The integrated scheduling of flexible manufacturing systems using Petri net modeling and heuristic searching

Mohammed Mohammed Magdy Mohammed
THE INTEGRATED SCHEDULING OF FLEXIBLE MANUFACTURING SYSTEMS USING PETRI NET MODELING 
& HEURISTIC SEARCHING

MOHAMMED MOHAMMED
MAGDY MOHAMMED

1997
THE AMERICAN UNIVERSITY IN CAIRO
SCHOOL OF SCIENCES AND ENGINEERING
ENGINEERING DEPARTMENT

THE INTEGRATED SCHEDULING OF FLEXIBLE
MANUFACTURING SYSTEMS USING PETRI NET
MODELING AND HEURISTIC SEARCHING

BY

MOHAMMED MOHAMMED MAGDY MOHAMMED

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science in Engineering

with concentration in:

(Industrial Engineering)

under the supervision of:

Dr. Mahmoud Younis

(The American University in Cairo)

June / 1997
Theses

Declaration to be signed by the author.

NAME: Mohammed Mohamed Magdy Mohammed

TITLE OF THESIS: THE INTEGRATED SCHEDULING OF FLEXIBLE MANUFACTURING SYSTEMS USING PETRI NET MODELING AND HEURISTIC SEARCHING

DEPARTMENT: Engineering

YEAR: 1997

Please sign and date ONE of the following paragraphs:

1. The thesis may be consulted in the Library and photocopied.

Signed: .......................................................... Date: ......................

OR

2. The thesis may be consulted in the Library, but may not be photocopied. The restriction on photocopying will cease two years after the date below, unless I apply for, and am granted, its renewal.*

Signed: .......................................................... Date: 5/6/1997

OR

3. The thesis may neither be consulted nor photocopied without written permission from me, or from the appropriate Head of Department if I cannot be contacted. This restriction will cease three years after the date below, unless I apply for, and am granted, its renewal.*

Signed: .......................................................... Date: ......................

* Application for renewal of restrictions will be considered by the Librarian, the appropriate Head of Department and the Chairman of the Academic Board or his nominee.
The American University in Cairo

School of Sciences and Engineering

The Integrated Scheduling of Flexible Manufacturing Systems Using Petri Net Modeling and Heuristic Searching

A Thesis Submitted by
Mohammed Mohammed Magdy Mohammed

To The Department of Engineering

June 1997

in partial fulfillment of the requirements for the degree of

Master of Science in Engineering with
Specialization in Industrial Engineering

has been approved by

Dr. Mahmoud A. Younis, (Supervisor)
Head, Mechanical Engineering Unit,
Engineering Department, AUC

Dr. Amin M. El-Kharboutly,
Professor, Mechanical Design & Production Dept.,
Faculty of Engineering, Ain Shams University

Dr. Nahed S. Abdel-Nour,
Professor, Engineering Department, AUC

Dr. Maher Y.A. Younan,
Head, Engineering Unit,
Engineering Department, AUC

Date

Department Chair/Program Director

Date

Dean

Date
DEDICATION

I dedicate this research to my beloved parents
ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Mahmoud Younis, for his constructive criticism and continuous supervision. In addition, I would like to express my sincere appreciation for his support and considerate help, without which this research would probably be non-existent.

I am also indebted to Dr. Frank DiCesare, Professor of Electrical, Computer, and Systems Engineering, at Rensselaer Polytechnic Institute, Troy, New York. The numerous papers and articles which he kindly sent to me, were by all means, the starting point of this research. Dr. DiCesare’s invaluable help was undoubtedly a key factor in bringing this research to the light.

I am also grateful to Dr. Abdallah Naim for the various important papers which he so kindly brought to my attention. Accordingly, I would like to thank Engineer Mohammed Askar for helping me find some sources in Alexandria that were directly related to this research topic.

I would also like to express my gratitude to Engineers Ahmed Abou-El-Sorour and Rania Mameesh for their help in developing the computer program involved in this research. I would like to thank them for the huge effort that they made in order to produce this computer program, on such a short notice.

Last, but definitely not least, I would like to thank my family. My parents and my sister, Mai, were a continuous support throughout this research. I can think of no words that can express or describe my feelings towards them. All I hope for, is that I can live up to their expectations. Thank you for bearing with me.
ABSTRACT

The goal of this research is to develop an integrated scheduling technique for flexible manufacturing systems (FMSs). This technique simultaneously considers both the machining and the transportation requirements involved in the FMS. The output of this proposed scheduling approach is an integrated schedule for both the jobs and the automated guided vehicles (AGVs). The objective of the scheduling function under consideration is the minimization of the makespan, or in other words the maximization of the production rate.

The scheduling approach, proposed in this research, is divided into two main parts. The first part is concerned with constructing a model of the FMS scheduling problem, whereas the second part is concerned with searching this model in order to reach an optimal or near optimal solution. The modeling tool that is used in this research is Petri nets, and the heuristic search function is based on the A* graph-search algorithm. The main type of Petri net model used in this approach is the timed-place Petri net (TPPN) model. In addition the colored Petri net (CPN) model is tested. The two heuristic search algorithms employed in this scheduling technique are the L1 and the Limited-Expansion A algorithms.

A computer program is developed to construct the Petri net model and perform the required search calculations to reach a final schedule. The proposed scheduling technique does not guarantee optimality. However, a sensitivity analysis study was conducted on the main parameters, to determine the impact of their variation in addition to their appropriate values that would render the best results. Several modifications are proposed to improve and enhance the efficiency of the search algorithm. The results obtained demonstrate that these proposals are successful. Finally the results obtained from using the CPN model indicate that this is a very promising field, that requires further investigation.
TABLE OF CONTENTS

LIST OF TABLES .................................................................................................................. viii
LIST OF FIGURES ........................................................................................................... x
LIST OF ABBREVIATIONS AND SYMBOLS................................................................. xiii

1 INTRODUCTION ............................................................................................................. 1

2 LITERATURE REVIEW ................................................................................................. 6
   2.1 PETRI NETS IN THE MODELING AND ANALYSIS OF MANUFACTURING SYSTEMS ... 6
       2.1.1 The Bottom-Up Synthesis Approach ................................................................. 12
       2.1.2 The Top-Down Synthesis Approach ................................................................. 17
       2.1.3 The Hybrid Synthesis Approach ....................................................................... 23
   2.2 APPLYING PETRI NETS IN THE SCHEDULING OF MANUFACTURING SYSTEMS ... 28
   2.3 OBJECTIVE ............................................................................................................. 40

3 THE PETRI NET MODEL ............................................................................................... 43
   3.1 THE ASSUMPTIONS ............................................................................................... 44
   3.2 THE QUESTION/ANSWER INPUT METHOD ........................................................... 47
   3.3 THE CONSTRUCTION APPROACH ......................................................................... 51
   3.4 THE BUILDING MODULES ................................................................................... 54
       3.4.1 The Machining Module ..................................................................................... 55
       3.4.2 The Transportation Modules .......................................................................... 62
           3.4.2.1 The Distributed AGVS .............................................................................. 63
           3.4.2.2 The Centralized AGVS ............................................................................ 83
           3.4.2.3 The Free Floating AGVS ......................................................................... 96
           3.4.2.4 The Robot ................................................................................................. 110
       3.4.3 The Remaining Modules ................................................................................... 112
   3.5 THE COLORED PETRI NET MODEL ..................................................................... 115
LIST OF TABLES

2.1 Scheduling / Decision Rules ................................................................. 32

3.1 A Process Plan Example ........................................................................ 49
3.2 The Description Of The General Machining Module .......................... 56
3.3 The Description Of The Transportation Module In Figure 3.9 ........... 67
3.4 The Description Of The Integrated Model In Figure 3.11 ................. 73
3.5 The Description Of The Integrated Model In Figure 3.12 ................. 75
3.6 The Description Of The Integrated Model In Figure 3.13 .................. 77
3.7 The Description Of The Transportation Module In Figure 3.15 ......... 78
3.8 The Description Of The Transportation Module In Figure 3.22 ......... 86
3.9 The Description Of The Integrated Model In Figure 3.24 ................. 90
3.10 The Description Of The Integrated Model In Figure 3.25 ............... 91
3.11 The Description Of The Integrated Model In Figure 3.26 ............... 91
3.12 The Description Of The Transportation Module In Figure 3.29 ....... 94
3.13 The Description Of The Transportation Module In Figure 3.30 ....... 95
3.14 The Description Of The Transportation Layout Unit ....................... 98
3.15 The Description Of The Transportation Layout Module In Figure 3.32 101
3.16 The Description Of The Movement Control Module In Figure 3.33 ... 104
3.17 The Description Of The Model Given In Figure 3.34 ....................... 108
3.18 The Description Of The Robot Transportation Module ................... 111

5.1 The Process Plan For Case Study 1 ..................................................... 138
5.2 The FMS Configuration For Case Study 1 .......................................... 139
5.3 The Operation Times For Case Study 1 .............................................. 139
5.4 The Results For Case Study 1 Using The Machining Module In Figure 5.1 142
5.5 The Results For Case Study 1 Using The Machining Module In Figure 5.2 143
5.6 The Results of Using the IL Ordering Strategy in the Case of the Large Petri Net Model .......................................................... 146
5.7 The Results of Using the IL Ordering Strategy in the Case of the Small Petri Net Model .......................................................................................... 147
5.8 The Comparison Between the IF and IL Ordering Strategies in the Case of the Large Petri Net Model .......................................................... 147
5.9 The Comparison Between the IF and IL Ordering Strategies in the Case of the Small Petri Net Model .......................................................... 148
5.10 The Effect of Varying b in the Case of IF Ordering .................................................. 154
5.11 The Effect of Varying b in the Case of IL Ordering .................................................. 154
5.12 The Results of Using b with w=1 in the Case of the Reduced Petri Net .......... 156
5.13 The Process Plan for Case Study 2 .......................................................................... 156
5.14 The FMS Configuration for Case Study 2 .............................................................. 157
5.15 The Operation Times for Case Study 2 .................................................................. 157
5.16 The Results for Case Study 2 Using Different Values of w and IF ...................... 158
5.17 The Results for Case Study 2 Using Different Values of w and IL ................. 158
5.18 The Results for Case Study 2 Using Different Values of b ................................. 159
5.19 The FMS Configuration for Case Study 2.1 ......................................................... 159
5.20 The Results for Case Study 2.1 Using Different Values of w ............................. 160
5.21 The FMS Configuration for Case Study 2.2 ......................................................... 161
5.22 The Results for Case Study 2.2 Using Different Values of w ............................. 161
5.23 The Comparison Between TPPN and CPN Models .............................................. 162

B.1 Conference Sites from 1980 to 1996 ................................................................. 186
# LIST OF FIGURES

1.1 *Research Area-Of-Application* ............................................................. 5

2.1 *Fusion of Series Places* ................................................................. 10
2.2 *Fusion of Series Transitions* ......................................................... 10
2.3 *Fusion of Parallel Places* ............................................................. 10
2.4 *Fusion of Parallel Transitions* ....................................................... 10
2.5 *Elimination of Self-Loop Places* .................................................... 10
2.6 *Elimination of Self-Loop Transitions* ............................................ 10
2.7 *An Example of a One-Way Merge* .................................................... 13
2.8 *An Example of Sharing Simple Elementary Paths* ............................ 15
2.9 *A Well-Formed Block and Its Associated Petri Net* ....................... 18
2.10 (a) *An Aggregate Petri Net* ........................................................ 19
2.10 (b) *A Well-Formed Block* ............................................................ 19
2.11 *The Refined Petri Net* ................................................................ 19
2.12 (a) *Aggregate Petri Net* ............................................................... 20
2.12 (b) *A 3-Well-Behaved Petri Net* .................................................. 20
2.13 *The Refined Petri Net* ................................................................ 21
2.14 (a) *Aggregate Petri Net* ............................................................... 21
2.14 (b) *The Refined Petri Net* ............................................................. 21
2.15 *The Four Basic Design Modules* ..................................................... 25
2.16 *Two Mutual Exclusion Examples* ...................................................... 27
2.17 *Shih and Sekiguchi’s Proposed Scheduling System* ....................... 30
2.18 *The Scheduling System Proposed by Oba and Ma* .......................... 34

3.1 *The Proposed Scheduling Approach* .................................................. 44
3.2 (a) *Initial State of a TPPN* ............................................................ 53
3.2 (b) THE EFFECT OF FIRING $t_2$ FIRST ................................................................. 53
3.2 (c) THE EFFECT OF FIRING $t_1$ FIRST ................................................................. 53
3.3 THE GENERAL MACHINING MODULE ................................................................. 55
3.4 A SPECIFIC MACHINING MODULE .................................................................. 60
3.5 THE MACHINING MODULE WITHOUT THE BUFFER CAPACITY CONSTRAINT ....... 61
3.6 A FLOWCHART OF THE DIFFERENT TRANSPORTATION MODULES .................. 64
3.7 THE FMS GENERAL LAYOUT (DISTRIBUTED AGVS CASE) ............................... 65
3.8 THE BI-DIRECTIONAL TWO PATH CONNECTION ............................................. 66
3.9 THE TRANSPORTATION MODULE FOR THE DISTRIBUTED AGVS (CASE 1) ........ 67
3.10 THE INTEGRATED MODEL FOR TRANSPORTING A PART WITH A DISTRIBUTED AGVS (CASE 1) ............................................................... 71
3.11 THE ‘1-2’ INTEGRATED MODEL FOR THE DISTRIBUTED AGVS (CASE 1) .......... 72
3.12 THE ‘2-1’ INTEGRATED MODEL FOR THE DISTRIBUTED AGVS (CASE 1) .......... 74
3.13 THE ‘2-2’ INTEGRATED MODEL FOR THE DISTRIBUTED AGVS (CASE 1) .......... 76
3.14 THE BI-DIRECTIONAL SINGLE PATH CONNECTION ....................................... 77
3.15 THE TRANSPORTATION MODULE FOR THE DISTRIBUTED AGVS (CASE 2) ....... 78
3.16 THE INTEGRATED MODEL FOR TRANSPORTING A PART WITH A DISTRIBUTED AGVS (CASE 2) ............................................................... 80
3.17 THE ‘1-2’ INTEGRATED MODEL FOR THE DISTRIBUTED AGVS (CASE 2) .......... 81
3.18 THE ‘2-1’ INTEGRATED MODEL FOR THE DISTRIBUTED AGVS (CASE 2) .......... 82
3.19 THE ‘2-2’ INTEGRATED MODEL FOR THE DISTRIBUTED AGVS (CASE 2) .......... 82
3.20 THE FMS GENERAL LAYOUT (CENTRALIZED AGVS CASE) ......................... 84
3.21 THE UNI-DIRECTIONAL TWO PATH CONNECTION .......................................... 85
3.22 THE TRANSPORTATION MODULE FOR THE CENTRALIZED AGVS (CASE 1) ...... 85
3.23 THE INTEGRATED MODEL FOR TRANSPORTING A PART WITH A CENTRALIZED AGVS (CASE 1) ............................................................... 88
3.24 THE ‘1-2’ INTEGRATED MODEL FOR THE CENTRALIZED AGVS (CASE 1) .......... 89
3.25 THE ‘2-1’ INTEGRATED MODEL FOR THE CENTRALIZED AGVS (CASE 1) .......... 89
3.26 THE ‘2-2’ INTEGRATED MODEL FOR THE CENTRALIZED AGVS (CASE 1) .......... 89
3.27 THE TRANSPORTATION MODULE FOR THE CENTRALIZED AGVS (CASE 2) ...... 92
3.28 THE CYCLIC LAYOUT ............................................. 93
3.29 THE TRANSPORTATION MODULE FROM M_2 TO M_3 (CYCLIC CASE) ............................................. 94
3.30 THE TRANSPORTATION MODULE FROM M_1 TO M_3 (CYCLIC CASE) ............................................. 95
3.31 THE TRANSPORTATION LAYOUT UNIT ............................................. 98
3.32 THE TRANSPORTATION LAYOUT MODULE ............................................. 100
3.33 THE MOVEMENT CONTROL MODULE ............................................. 103
3.34 THE INTERACTION BETWEEN THE MACHINING AND THE MOVEMENT CONTROL MODULES ............................................. 107
3.35 THE LAYOUT OF A MACHINING CELL ............................................. 110
3.36 THE ROBOT TRANSPORTATION MODULE ............................................. 111
3.37 THE SCHEMATIC DIAGRAM OF THE PETRI NET MODEL FOR JOB 1 ............................................. 114
3.38 A SECTION OF A CPN MODEL ............................................. 117

4.1 AN EXAMPLE PETRI NET MODEL ............................................. 122
4.2 THE REACHABILITY GRAPH OF THE PETRI NET IN FIGURE 4.1 ............................................. 123
4.3 THE L1 SCHEDULING ALGORITHM ............................................. 128

5.1 THE MACHINING MODULE WITH INFINITE BUFFER CAPACITY ............................................. 140
5.2 THE MINIMIZED MACHINING MODULE USED BY LEE AND DICE SARE ............................................. 141

A.1 PETRI NET MODELING COMPACTNESS ............................................. 174
A.2 A MARKED PETRI NET ............................................. 175
A.3 AN INITIAL MARKING OF A PETRI NET ............................................. 176
A.4 THE MATRIX REPRESENTATION OF A PETRI NET ............................................. 177
A.5 THE INCIDENCE MATRIX ............................................. 178
A.6 THE PETRI NET ENABLING AND FIRING RULES ............................................. 179
A.7 SELF-LOOP ELIMINATION ............................................. 182
LIST OF ABBREVIATIONS AND SYMBOLS

The main abbreviations that were used in this research are:

AGV : Automated Guided Vehicle
AGVS : Automated Guided Vehicle System
CPN : Colored Petri Net
FMS : Flexible Manufacturing System
TPPN : Timed Place Petri Net

The mathematical notation and symbols used in this research are:

A : The incidence matrix of a Petri net
A1 : The mean of all possible processing and travel times
AGV_x : AGV number x
b : The maximum number of markings that can be put in the OPEN list
c(n,n') : The cost of the arc heading from node n to node n'
dep (m) : The depth of the marking m in the reachability graph
E1 : The average number of transition firings per token from the initial to final place
f(M) : The estimate of the cost of an optimal path from the initial to the final marking which passes through marking M
g(M) : The current lowest cost obtained from the initial marking to marking M
h(M) : The estimate of the cost of the optimal path from marking M to the final marking
h^*(M) : The cost of the optimal path leading from marking M to the final marking
I : The input function defining the directed arcs from places to transitions
I(p_i,t_i) : The number of arcs connecting transition t_i to its input place p_i (a_i^t)
iff : if and only if
JN : Job number N (e.g. J2 = Job number 2)
JN (L) : Job number N with a required lot-size L
k : The total number of input places p for a transition t
m : The number of places
m(p) : The marking of the place p (i.e. the number of tokens in place p)
M₀ : The initial marking
Mᵢ : The i-th Petri net marking
MN : Machine number N (e.g. M1 = Machine number 1)
N : The set of all non-negative integers
n : The number of transitions
O : The output function defining the directed arcs from transitions to places
O(pᵢ,tᵢ) : The number of arcs connecting transition tᵢ to its output place pᵢ (aᵢ⁺)
P : The finite set of places
PN : The five tuple representation of a Petri net
∅ : The empty set
rₓ : The remaining time delay of the input place pᵢ of transition tᵢ for i = 1,2,...k
T : The finite set of transitions
uᵏ : The k-th firing vector
w : The weighting factor
Z : The four tuple representation of a Petri net structure
Chapter 1
INTRODUCTION

Production scheduling is concerned with the utilization and allocation of the available resources over time, in order to effectively manufacture a given set of products. The objective of scheduling is to assign and to sequence the use of these shared resources (e.g. machines, robots, transportation devices) such that production constraints are satisfied and production costs are minimized. Production scheduling problems are very complex, combinatorial in nature, and have been proven to be NP-hard problems (France 1982).

On the other hand flexible manufacturing systems (FMSs) are considered to be the ‘sharp end’ of the integration of automated industry, which is now recognized by the label computer integrated manufacturing (CIM). FMS is the latest level of automation along an evolutionary road to achieve ever more productivity and flexibility from manufacturing equipment (Parrish 1990). Thus an FMS is a computerized manufacturing system that is controlled and operated by a host computer, and it is normally composed of a collection of production equipment (e.g. CNC machines) that are physically connected by a transportation system (e.g. automated guided vehicles, AGVs). FMSs were developed to face the new global market and international competition, where rapidly changing customer demands forced the need for more flexible and efficient production systems. Therefore, FMSs were designed in an attempt to reach a compromise between the highly productive (but rigid) mass production systems and the highly flexible (but inefficient) job shops.

However, despite the advantages of using an FMS, the decision of including or installing such a system in the factory, has to be studied carefully by the management. The reason for this is that introducing an FMS requires a relatively large capital investment. Therefore, this investment has to be justified economically before being endorsed. In addition, the environment of an FMS is usually dynamic and complex.
Accordingly, the most important problem faced by the managers and the engineers in charge of an FMS, is ensuring that the FMS is operating at the best possible efficiency. In other words, one needs to assign the available resources of the FMS to the different processes that are required to produce individual products, such that the best possible efficiency is achieved (i.e. a scheduling problem).

Thus, the scheduling of an FMS plays a pivotal role in determining the efficiency and productivity of the production system. In addition the scheduling technique that is employed should offer the ability of quickly responding to the dynamic and continuously changing environment that is surrounding the FMS.

In the available FMS scheduling literature, the majority of the papers deal with the machine scheduling and the automated guided vehicle (AGV) routing as two separate issues. In other words the consideration of the material handling system and the part-processing facilities in the scheduling process has usually been conducted separately and treated independently. Obviously, this poses the serious problem of developing a part-schedule which cannot be applied due to the routing and material transportation constraints. However, in a manufacturing system where machining times are comparable in their amount to material transportation times, it is no longer realistic or acceptable to ignore the scheduling of the transportation system (or even treat it separately). Therefore, the consideration of the effective sequencing and scheduling of the material handling system can have a major impact on the productivity of the manufacturing system under consideration.

For this reason, a fairly recent field of research has been concerned with studying the issue of simultaneous job scheduling and routing of the FMS. Surely such a technique would greatly reduce the time spent in trying to harmonize between two separate schedules for parts and vehicles. In addition addressing the scheduling problem in such a manner would ensure the overall optimization of the performance of the FMS components/resources (i.e. machines and AGVs).
Thus, this research aims at introducing an integrated scheduling technique which considers both the processing and the routing constraints involved in an FMS. The objective of the proposed scheduling technique is to minimize the makespan of the schedule (i.e. schedule completion time). This objective is equivalent to maximizing the productivity of the FMS.

The modeling of the FMS system will be conducted using Petri nets. This choice was made due to the fact that Petri nets manage to capture and model several features that are exhibited by flexible manufacturing systems such as concurrency, conflict, mutual exclusion, routing flexibility, and shared-resources. In addition, Petri nets offer an elaborate graphical representation, besides their mathematical formalism, which can be easily communicated and understood. The model will be constructed using a modular bottom-up approach. Several Petri net modules will be developed and introduced to model the various FMS subsystems.

The scheduling process will be performed using the Petri net model. A heuristic search algorithm will be applied on the model to generate the required schedule. This heuristic function will generate a sequence of firings from the initial marking of the Petri net to the final marking. This sequence can be regarded as the schedule given in terms of the order of events that are to take place.

Another goal which this research seeks to accomplish, is the utilization of colored Petri nets as the modeling tool instead of the commonly used timed place Petri net. Accordingly the results of this study will indicate whether colored Petri nets offer any improvements on the proposed scheduling technique.

Thus the importance of this research topic is that it deals with a relatively new field which is the combined scheduling of both parts and material handling devices for an FMS. Evidently this will lead to a more efficient scheduling procedure, which will
eliminate the possibility of having to reschedule or alter developed job schedules to accommodate the routing constraints. Accordingly, a more efficient scheduling technique is developed which avoids any unnecessary extra costs and development time.

**Problem Statement**

Thus, the problem which this research is considering can be stated as follows: - Find a schedule for the given set of jobs (tasks) along with the AGVs, such that when this schedule is performed on the given FMS with the given configuration, it satisfies the given objectives.

A computer program will be developed to assist in the solution of the above problem. Based on the input entered by the user, the program will construct the appropriate Petri net model, and then it will apply the heuristic search algorithm to generate the required schedule. In this research, minimizing the makespan will be the only objective under consideration. The term ‘configuration’ was used in the problem definition to indicate the fact that the user will have several options from which he/she can choose the suitable material handling system and the FMS layout.

**Scope of Work**

The proposed scheduling technique focuses on the concept of integrated scheduling, which combines between parts and AGVs. However, the developed computer program also offers the user the option of creating a job-schedule which excludes the material transportation system. In addition, the colored Petri net model is limited to job-schedules and does not consider the AGVs.

The main assumptions and constraints which define the scope of this research are listed and discussed in section 3.1 of Chapter 3.
Figure 1.1 provides a general overview of the area of application of this research.

This thesis work and research lie at the intersection of these three fields:

- **Scheduling**
- **Flexible Manufacturing Systems**
- **Petri nets**
Chapter 2
LITERATURE REVIEW

This literature survey focuses on the application of Petri nets in the modeling and scheduling of industrial-related systems, and specifically concentrates on flexible manufacturing systems (FMSs). The use of Petri nets in modeling, and the different modeling techniques are discussed, along with a critique of each modeling approach. In addition the application of Petri nets in the scheduling process is considered and the different approaches that have been developed are reviewed in detail. Finally a concise description of how this study intends to contribute to the available research in this field is presented.

2.1 Petri Nets in the Modeling and Analysis of Manufacturing Systems

Petri nets were originally introduced by Carl Adam Petri in 1962, in his doctoral dissertation, to model and study communication with automata. Ever since that date Petri nets have been growing and expanding in terms of their applications and different types. Petri nets are a graph theoretic as well as a visual graphical tool that is suitable and well-suited for the modeling, analysis, performance evaluation and control of interacting concurrent discrete event dynamic systems (DEDS). However, in this section one will only consider the application of Petri nets as a modeling tool.

Zurawski and Zhou (1994) conclude that the main reason for the confinement of Petri nets to the academic and research institutions is the difficulty involved in the construction of Petri net models. This fact becomes apparent when the system to be modeled is complex, which is usually the case in medium or large scale systems. They claim that there is no methodology which enables a fully automatic construction of the Petri net model from the provided system specifications. In order for Petri nets to be more commonly used, especially by industry, methods and techniques allowing for automatic or semi-automatic construction of Petri net models from the requirement specifications have to be developed. Moreover, they state that,
in most cases Petri nets are constructed in an ad hoc manner. However, several attempts have been recently made to make the modeling process more systematic.

One specific example of these attempts was presented by DiCesare and Desrochers (1991). Their modeling methodology, which is designated for manufacturing systems, was based on the process plans. These process plans should describe in detail all the activities required for each product; including a listing of all the resources that should be utilized in addition to their relative order. From the available process plans the authors developed their modeling methodology which was presented in a stepwise fashion. The first step in this modeling technique is the identification of the operations and resources required for the production of individual items of each product. The activities should be ordered according to the technological precedence relations provided in the process plans. The next step is to create and label a place that represents the status of each activity and two transitions depicting the start and the end of this activity. The following step is to consider each activity and create a place for each resource that is required for performing this operation, and connect it to the start and end transitions of this particular activity. Finally, one has to specify the initial marking of the resulting Petri net model.

According to Murata (1989), one of the advantages of Petri nets is that as an elaborate graphical tool, they can be easily used as a visual aid in communicating the modeling structure and the underlying system information. In addition, the use of tokens in Petri nets can simply model the dynamics of the system under consideration. Another advantage of Petri nets is that there is only one rule pertaining to understanding their application, which is namely the enabling and firing rule.

Zurawski and Zhou (1994) state that Petri nets as a graphical and mathematical tool provide a uniform environment for the modeling, validation, analysis, design and control of discrete event systems. They believe that the major advantage of Petri nets is that using the same model one can conduct an analysis of the system properties, evaluate the performance of the system, and even design
discrete-event simulators and controllers for the system. They account for the
development of Petri nets in the modeling fields of production, computer and
communication systems; to their ability to model properties such as concurrent
operations, conflicts, precedence relations, process synchronization, and resource
sharing. In addition, Petri nets can be described by a set of mathematical
equations which reflect the behavior of the underlying system. Thus any modeled
system can be formally analyzed for behavioral properties such as precedence
relations amongst processes and events, concurrent operations, appropriate
synchronization, freedom from deadlock, and mutual exclusion of shared resources.
Comparing this aspect with the simulation-based model validation, one can clearly
see the drawback of simulation which only produces a limited set of the system
states and therefore can only indicate the presence, not the absence of errors in the
model and accordingly the system. On the other hand Petri net models can be
analyzed for significant system properties and can reveal whether problems, such
as deadlock or overflow, are absent or not. Nevertheless, as mentioned earlier, in
the case of very complex Petri net models, the analytical examination which
ascertains the absence of errors becomes infeasible and one has to resort to
simulation.

Murata (1989) states that the major weakness of Petri nets is the complexity
problem. Jeng and DiCesare (1993) affirm the fact that using Petri nets to represent
concurrency, conflict, and mutual exclusion is simple, straightforward, and
convenient. However, they admit that numerous problems arise when the system to
be modeled is very complex. In these cases, the final model becomes very large and
the huge state space of the system causes the task of analyzing the model to be
very difficult. The size of the Petri net model renders the two fundamental methods
of analysis (reachability tree and invariants) intractable and impractical. Thus, in
attempting to solve the problem of model complexity and model analysis, two
different approaches have been developed and are still being researched. These
two approaches, namely, model reduction and model synthesis, are considered in
detail.
The first approach which is model reduction, tries to reduce the Petri net structure while maintaining the logical properties of interest. Thereby, when the analysis is conducted on the reduced Petri net, the results obtained are similar in their outcome to those which would have been reached if the analysis was performed on the original net. Therefore, it is extremely important to develop methods for transforming a Petri net in a stepwise fashion whilst preserving the system properties that will be analyzed. Suzuki and Murata (1983) discuss such an approach, where subnets are reduced to single transitions or places while retaining properties such as liveness and/or boundedness.

Zurawski (1994) proposes a method for the systematic construction of functional abstractions of Petri net models. His approach is based on the replacement of the constituent components of the Petri net model with their functional abstractions. These functional abstractions, which are smaller in size (i.e. number of transitions and places) than the original subnets, represent the external behavior of the components. They can be easily verified to ensure the correct interaction between the different modules of the Petri net model. However, the problem with this systematic approach is that it is restricted to a specific class of Petri net models. Moreover, this method relies on temporal Petri nets, which were introduced by Suzuki (1985), rather than ordinary Petri nets.

Silva (1985) presented six basic reduction rules. This reduction kit is illustrated below starting from Fig. 2.1 to Fig. 2.6. Figures 2.1 and 2.2 depict the rules of fusion of series places and fusion of series transitions; respectively. Figures 2.3 and 2.4 depict the rules of fusion of parallel places and fusion of parallel transitions; respectively. Figures 2.5 and 2.6 depict the rules of elimination of self-loop places and elimination of self-loop transitions; respectively. These transformations preserve the Petri net properties of boundedness, liveness, and safeness. Assuming that $(Z, M_o)$ and $(Z', M'_o)$ denote the Petri net before and after any of these transformations; one can be assured that $(Z', M'_o)$ is bounded, live or safe iff $(Z, M_o)$ is bounded, live or safe, respectively.
The first approach which has been discussed up to this point, namely the reduction approach, is an attempt to simplify the analysis process of the underlying Petri net model. On the other hand, the second approach which is the synthesis approach addresses the modeling construction problem in addition to the analysis problem. The construction of Petri nets, especially for large scale systems, is by no means a simple task and it requires a great deal of experience (Zurawski and Zhou, 1994). In the past two decades several synthesis approaches have been researched and developed in order to achieve a systematic construction technique of Petri net models. Since in a moderately sized manufacturing system the complexity of both the design and analysis at the implementation level of detail is impractical and unreasonable; the challenge to researchers became the ability to develop a systematic technique for the construction of Petri net models that possesses and retains the desirable set of properties that ensure proper functioning in a manufacturing system.

The different synthesis approaches that have been researched can be classified as either bottom-up, top-down, or hybrid approaches. All these synthesis approaches have the common goal of seeking to simplify the design stage of Petri net models. In addition, they tackle the size and complexity problem, and accordingly the analysis issue, which arises when the system to be modeled is complex (e.g. flexible manufacturing systems). Therefore, one can view the importance of designing Petri nets with desirable properties for manufacturing systems from two different aspects. First, such an approach helps in avoiding the costly analysis of the model properties prior to implementing a Petri net-based controller for practical manufacturing systems. Second, performance evaluation and time analysis can be conducted by assigning appropriate time variables to the relevant places or transitions. Due to the importance of each synthesis method and the extensive research that has been conducted, they will be discussed separately in the following sections.
2.1.1 The Bottom-Up Synthesis Approach

The bottom-up approaches are characterized by starting with the construction of subnets for component processes and reaching the final net through the merging and/or linking of all these subnets. Thus, this technique calls for the decomposition of the system under consideration, into separate subsystems which will be modeled independently ignoring any interaction between them. The properties of these subsystems are usually verified easily. These subsystem models may have places and/or transitions in common which represent the interaction between them. At each synthesis step, these interactions are considered, and the corresponding places and/or transitions are either merged or linked together, in order to combine the overlapping and interrelated subsystems. The power of this technique lies in the fact that the analysis of the resulting larger subsystem (i.e. combined net) can be derived from the constituent subsystems. Thus, the analysis of the relevant properties is conducted after each combination (i.e. synthesis) step and this results in the simplification of the analysis of the final model. It remains to be said, that the merging or linking operation can be performed on either places, transitions, or a combination of both.

Agerwala and Choed-Amphai (1978) were the first to notice that calculating the invariants of a global net model through the invariants of the subnets from which it is formed, could be used to synthesize concurrent systems modeled by Petri nets. They proposed a set of synthesis rules for the bottom-up construction of a complete Petri net from subnets which share common places. At each synthesis step, the different subnets were combined through the merging of a set of places into a single place. This was called a one-way merge, and the theorem developed by the authors provided a means for calculating the invariants of the resultant Petri net model from the invariants of the subnets out of which it was formed. This in turn facilitated the analyzing process of the final Petri net, and reduced the effort through the proposed stepwise design methodology. An example of the one-way merge technique is presented in Fig. 2.7(a) and 2.7(b). It can be seen, that the two subnets are combined through the merging of places $p_3$ and $p_5$ into the new place $p_3'$. 
Narahari and Viswanadham (1985) continued along the same line of work, and extended this systematic bottom-up synthesis approach to include the merging of transitions. Their research was performed in the manufacturing field as they were investigating the problem of modeling and analyzing flexible manufacturing systems. In such systems, a product might require more than one machine operation, and the

![Diagram of Petri nets]

(a) Two subnets  
(b) Combined Petri net

Figure 2.7 An example of a one-way merge

flexibility of the system may provide several alternatives as to which machine should be selected to perform this operation. In addition, several different types of products could be manufactured simultaneously in the system. Therefore, the authors suggested that the building blocks or the fundamental subnets should depict the system operations. Thus, in their approach, a Petri net was first created for each and every basic operation such as an intermediate machining operation for a product. Accordingly, the subnet that represents a single product could be constructed through the merging of the places that denote the relevant machine operations. In the same way, the Petri net representing the whole system was constructed from the merging of the places of the subnets that represent the products. Narahari and Viswanadham (1985) provided two theorems related to their synthesis approach. One theorem dealt with calculating the P-invariants for the nets obtained by merging places. This theorem could be viewed as a development of the work of Agerwala and Choed-Amphai (1978), since it allowed for more than one set
of places to be merged at each synthesis step. The other theorem dealt with the T-invariants for the nets obtained by merging transitions. Both theorems were used by the authors to verify important properties in the context of manufacturing systems.

Krogh and Beck (1986) developed a different bottom-up synthesis technique in which safeness and liveness of the resulting Petri net models was guaranteed. They extended the concept of merging a common place or transition to the sharing of common paths. Their method shares simple elementary paths in which no transition or place appears more than once. In their work they introduced two elementary paths for the synthesis procedure. The first was the solitary transition path (STP) in which a simple elementary path terminated on both ends by a place and for each transition in the path there was only one input and one output place. The second was the solitary place path (SPP) in which a simple elementary path terminated on both ends with a transition and for each place in the path, it was either an input place or an output place for exactly one transition. In their procedure the starting subnets are all simple elementary circuits (SECs), which are finite-length paths with the same initial and terminal places. One SEC is chosen as the initial Petri net. Another SEC is combined to the initial one through the common STP or SPP which they share. This merging is continued until all the SECs are included and the final Petri net model has been reached. The authors developed a theorem stating that the final Petri net will be live and safe for any initial marking provided that there is only one token for each P-invariant of the system. In addition they showed that after each synthesis step the P-invariants of the intermediate Petri net can be easily calculated. An example of the proposed synthesis method is presented in Fig. 2.8(a) and 2.8(b). It can be seen that the two SECs share the common SPP \( t_1, p_2, t_2 \), which upon merging created the final Petri net in Fig. 2.8(b).

Lastly, one needs to mention that the results of Krogh and Beck were extended by Koh and DiCesare (1990) to model nets with bounded places (i.e. there is a maximum limit for the number of tokens that a place can hold) and generalized Petri nets (i.e. Petri nets with multiple arcs). In addition Valavanis (1990) introduced the extended Petri net version in which each token differed from the other so that more information could be carried by the net model.
Jeng and DiCesare (1992) proposed another synthesis technique which they called the "general approach". However, one can argue that this approach should be classified under the category of the bottom-up synthesis methods. This will be clarified during the explanation of this specific technique. The authors claim that their approach helps in designing a class of Petri nets that can model shared-resource automated manufacturing systems. With respect to other approaches, this

![Diagram of two simple elementary circuits](image)

(a) Two simple elementary circuits

![Diagram of a combined Petri net](image)

(b) Combined Petri net

Figure 2.8 An example of sharing simple elementary paths

This technique imposes minimal restrictions in order to give the modeler more freedom during the representation of the interactions among the different subsystems. To modularize this synthesis process, Resource Control Nets (RCNs) are defined as the initial generic modules, and the shared-resource system under consideration is formulated as the processes that control the resources and the interactions among the processes. The system model is built by merging these modules through their common transitions and common transition subnets, which denote the interaction among the processes (Jeng and DiCesare, 1992). Thus, it is evident that this technique follows the bottom-up approach since it decomposes the system into several subsystems (i.e. modules) and combines the independent models for each subsystem in a stepwise fashion until the final model is reached. Furthermore, the
resultant net has been proven to be conservative and bounded. In addition an algorithm was developed to check the structural liveness of the net. Finally, this technique was applied on the modeling of a flexible manufacturing system in which machines, tools, automated guided vehicles (AGVs), and buffers were acting as shared resources in order to produce two different product types.

After describing the bottom-up synthesis methodology and discussing the previous different approaches, one can indicate the advantages and the disadvantages of this technique. In the simple terms of Jeng and DiCesare (1993), the bottom-up synthesis approach exhibits one main advantage which is the simplicity of the modeling of the different subsystems. This is due to the fact that these subsystems usually have direct real-life correspondence (e.g. machines, robots, buffers, AGVs, etc.), and do not encompass any complicated interactions within their description and specification. On the other hand, the main disadvantage of bottom-up synthesis approaches is that final net model may not exhibit the same properties that were present in the initial subsystems. Nevertheless, the effect of this disadvantage is partially reduced due to the fact that after each synthesis step the properties of the resultant net can be easily derived from the properties of the subnets out of which it was composed. Another disadvantage of this synthesis approach is that the analysis is often based on the analytical calculation of the net invariants which poses the problem of not being able to determine the existence of important properties such as liveness and reversibility. In addition, some of the methods that have been discussed do not guarantee the preservation of the important properties.

Jeng and DiCesare (1995) criticize the bottom-up synthesis approach from within the manufacturing systems context. They affirm the fact that automated manufacturing systems (e.g. flexible manufacturing systems) display a highly complex behavior due to the presence of several shared resources and the complicated interactions that take place between the different system components. Therefore, it becomes evident that shared resources play a vital role in the field of automated manufacturing systems. It is argued that the work of Krogh and Beck (1986) in addition to Koh and DiCesare (1990) is mainly concerned with nonshared-
resource systems (i.e. resource sharing is absent). On the other hand the work of Koh and DiCesare (1991) deals with systems where there is only one shared resource in each subsystem. Nevertheless, bottom-up methods do provide some freedom for the modeler in specifying the system and the interaction among the concurrent processes. In the next section we discuss a complementary synthesis approach.

2.1.2 The Top-Down Synthesis Approach

The top-down synthesis approach is characterized by starting with an aggregate model of the system under consideration, which initially neglects the low level details. This model is then refined in a stepwise manner to include the intricate details of the modeled system. The power of the top-down synthesis technique is that it starts with a global view of the system and then provides a certain hierarchical advancement methodology to the design phase. In addition several researchers have attempted to introduce specific refinement routines that guarantee the preservation of the important net model properties after each refinement step. It is interesting to note that top-down synthesis approaches are related to reduction methods for analysis, since the reverse of any reduction rule could be seen as a refinement to a Petri net model whilst maintaining the desirable properties. Thus, it can be confirmed that similar to the bottom-up synthesis techniques, the top-down approach strives to simplify both the design and analysis of the final Petri net model. Researchers have investigated and developed two different schemes for the refinement and expansion of the Petri net model. These two approaches, are the refinement of transitions and the refinement of places; and are discussed in detail in the following paragraphs.

Valette (1979) was the first to use the step-by-step top-down refinement techniques. He proposed a method for the stepwise refinement of transitions for the synthesis and analysis of Petri nets. This refinement was based on the assumption that the firing of a transition is not instantaneous, which actually is a relaxation of the original definition of Petri nets and the Firing Rule. In this case the firing event is
composed of two steps, which renders the possibility of replacing the transition with a subnet representing any sort of complex operation. In addition, Valette presented the conditions under which this refinement step could take place whilst preserving the significant net properties such as liveness and boundedness. He proved that a well-formed block could be substituted for a transition which is not 2-enabled, without losing the properties of boundedness, safeness, and liveness. In order to understand this statement and the proposed refinement technique one needs to define a well-formed block and a 2-enabled transition. First, a block is interpreted as a net with one input and one output transition, called the initial and final transitions, respectively. The associated Petri net of a block is formed when a place, called the idle place is added such that its input and output transitions are the final and initial transitions of the net, respectively. An illustration of a block and its Petri net is given in the following Fig. 2.9.

![Figure 2.9 A well-formed block and its associated Petri net](image)

A well-formed block exists when the corresponding block is live; has an initial marking such that the idle place is not empty; and accordingly the only enabled transition is the initial transition. Second, a 2-enabled transition occurs if and only if there exists a marking $m$ reachable from the initial marking such that:

$$m(p) \geq 2 \times I(p,t), \forall p \in P; \text{ where}$$

**Equation 2.1**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m(p)$</td>
<td>is the marking of the place P</td>
</tr>
<tr>
<td>$l(p,t)$</td>
<td>is the weight of the arc connecting place p to transition t</td>
</tr>
<tr>
<td>$P$</td>
<td>is the finite set of places</td>
</tr>
</tbody>
</table>
The refinement technique proposed by Valette is demonstrated in Fig. 2.10 and 2.11. Figure 2.10(a) illustrates an aggregate Petri net where transition $t_2$ which is not a 2-enabled transition will be replaced by the well-formed block presented in Fig. 2.10(b). Figure 2.11 illustrates the resultant Petri net which is the outcome of the refinement step. This refinement process is continued until the required level of detail is reached.

(a) An aggregate Petri net

(b) A well-formed block

Figure 2.10

Figure 2.11 The refined Petri net
Suzuki and Murata (1983) developed and generalized the work of Valette. It should be noted that their work was previously referred to within the context of Petri net model reduction. This is due to the fact that their research and methods can be applied to either reduction or expansion of Petri nets. The authors extended the refinement technique proposed by Valette, and defined the new concepts of a k-well behaved Petri net which is very similar to the definition provided by Valette for a well-behaved Petri nets except for the fact that the idle place can hold up to k initial tokens, where k is any non-negative integer. Likewise, a k-enabled transition can only occur if and only if there is a marking m reachable from the initial marking such that:

\[ m(p) \geq k \times l(p,t), \forall p \in P \]  

(Equation 2.2)

Thus Suzuki and Murata suggested that a transition which is not k+1-enabled can be replaced by a k-well behaved Petri net, without losing the properties of boundedness, safeness, and liveness. The refinement technique is demonstrated below in Fig. 2.12 and 2.13. Figure 2.12(a) illustrates an aggregate Petri net where transition \( t_1 \) which is not a k-enabled transition for k\( \geq 4 \) but is 1-, 2-, and 3-enabled; will be replaced by 3-well-behaved Petri net presented in Fig. 2.12(b). Figure 2.13 illustrates the resultant Petri net which is the outcome of the refinement step. This refinement process is continued until the required level of detail is reached.

(a) Aggregate Petri net  
(b) A 3-well-behaved Petri net

Figure 2.12
Figure 2.13 The refined Petri net

The refinement of places was also discussed in the work of Suzuki and Murata (1983). They presented a technique where a transition $p_0$ can be replaced by two places $p_1$ and $p_2$ along with a transition $t_1$, whilst maintaining the properties of boundedness, safeness, and liveness. However, the following conditions had to be satisfied. First, the input transitions of $p_0$ should be the input transitions of $p_1$ and the output transitions of $p_0$ should be the output transitions of $p_2$. Second, the only input and output places of $t_1$ should be $p_1$ and $p_2$, respectively. Lastly, the initial marking of $p_1$ should be equal to the initial marking of $p_0$ whereas the initial marking of $p_2$ should be zero. This refinement technique is presented below in Fig. 2.14.

(a) Aggregate Petri net

(b) The refined Petri net

Figure 2.14
Zhou, McDermott, and Patel (1993) used another top-down refinement approach which was directly applied in the modeling and analysis of a flexible manufacturing system cell. The motivation behind this paper was that there was no sufficient literature related to the details of modeling and analyzing a practical real-life flexible manufacturing system. Therefore this work, according to the authors, was intended for industrial engineers and academic researchers as a real-life example and guide to the application of Petri nets in the modeling and analysis of flexible manufacturing cells. The modeling strategy that was followed was a simple combination of system decomposition, top-down refinement, and modular composition. The first step in the synthesis of the Petri net model for the flexible manufacturing cell was the system decomposition. The system was decomposed into two functional parts, which were the material handling section and the machining section. Furthermore, the machining section was divided into two parts corresponding to the two different operations that were conducted. Accordingly, three separate Petri net models were constructed for each division, and each model was inspected and verified for the properties of liveness, safeness, and reversibility. The following step was to combine the three models in order to obtain the final Petri net. This was accomplished through the refinement of the places in the material handling net model that were associated with the locations at which the machining operations took place. Thus, each of the two places in the material handling model was replaced with the corresponding Petri net model depicting the machining operation. This technique simplified the synthesis process, and achieved a final Petri net model with all the desirable properties such as liveness, safeness, and reversibility.

Thus, it is obvious that the top-down synthesis approach has the advantage of simplifying the synthesis process through providing a structured and systematic design methodology. In addition this approach can guarantee the preservation of several favorable properties in the context of manufacturing systems such as boundedness, liveness, and reversibility. However, this approach fails to generate or preserve the required properties when applied to concurrent systems where resources are shared. Jeng and DiCesare (1993) point out that the proposed techniques of Valette (1979) and Suzuki and Murata (1983) are limited to systems
that can only be described as independent modules. In other words the shared resources can only be represented within the well-formed blocks or well-behaved Petri nets, due to the fact that both have only one input and one output transition. Even Zhou and DiCesare (1990) admit that the top-down modular approach which they proposed in 1989 failed to deal with systems and environments where highly shared-resources existed. In the next section we discuss the most recent synthesis approach which was specifically developed to deal with this problem.

2.1.3 The Hybrid Synthesis Approach

Recognizing the drawbacks of the bottom-up and top-down approaches, researchers attempted to investigate and search for a new technique which could avoid the disadvantages encountered in both methods. This research resulted in the recent introduction of the hybrid synthesis approach. This new technique addresses, in particular, the problem of dealing with shared resources in the system being modeled. The hybrid approach, as implied by its name, is a combination of both the top-down and the bottom-up techniques. This approach divides the design process into two main stages (Zhou and DiCesare 1990). In the first stage there is the top-down approach where the designer starts with a first-level Petri net as an aggregate model of the system. This Petri net is constructed and verified for the properties of boundedness, liveness, and reversibility. Then, stepwise refinements of operation places and/or transitions are conducted until the required level of detail is reached. Then the second stage starts using the bottom-up approach to add the resource places to the Petri net model. The power of the hybrid approach lies in the fact that it combines the benefits of both the top-down and bottom-up approaches. Thus, the system specification and control details are introduced in an incremental manner, which simplifies the design stage. In addition the important properties of the Petri net model are guaranteed, which simplifies the analysis stage. In the following paragraphs the hybrid approach is discussed in detail.

Zhou and DiCesare (1990) were the first to introduce the new concept of hybrid synthesis. However, the work of Zhou, DiCesare, and Desrochers (1992)
provides a more detailed, mature and complete explanation of this new synthesis approach. Therefore, one will base this brief description on the latter research. The authors claimed that this systematic synthesis approach was especially developed to deal with automated manufacturing systems, due to the highly interactive nature of these manufacturing environments. Nevertheless, this approach can be applied to any system which exhibits the feature of shared-resources. However, before discussing their approach one needs to mention the tools which they used to simplify the analysis process. In the top-down refinement stage, the operation places were replaced by basic design modules. In addition, parallel and sequential mutual exclusions were used to model the shared resources in the bottom-up stage. Therefore, prior to describing the hybrid synthesis approach, one needs to study and understand the set of basic design modules, and the mutual exclusion structures that were proposed by the authors to facilitate the synthesis and analysis procedure. First, the basic design modules are briefly described.

The well-formed blocks proposed by Valette (1979) and the well-behaved Petri nets introduced by Suzuki and Murata (1983), were essential concepts for the description and application of the techniques and theorems proposed by the corresponding researchers. Similarly, Zhou, DiCesare and Desrochers (1992) state that the basic design modules which they had previously introduced in 1989, can act as an aid for their new Petri net synthesis approach. Four basic design modules are introduced and are applied instead of the synthesis modules that have been proposed by either Valette or Suzuki and Murata. The advantage of the new basic design modules is that they are specific, easily applicable in practical manufacturing design problems, and lastly the refinement theory using these modules can be extended to include the property of reversibility. These four basic design modules are: the sequence, the parallel, the choice, and the decision-free choice Petri net modules. It was proven that if a place or a transition in a Petri net is replaced (i.e. refined) by any one of these four basic design modules, then the properties of boundedness, liveness, and reversibility are preserved if they already exist. The four design modules are illustrated below in Fig. 2.15.
In the context of manufacturing systems, a sequence Petri net module represents a series of $n$ successive operations. A parallel Petri net module represents $n$ operations that begin execution at the same time. These $n$ operations can end at different time instants, yet no successive operation can start until all $n$ operations are completed. A choice Petri net module represents the situation where there are $n$ choices for conducting the successive operation. In the context of manufacturing, it depicts a conflict state where after a certain operation is completed

(a) A sequence Petri net module

(b) A parallel Petri net module

(c) A choice Petri net module

(d) A decision-free choice Petri net module

Figure 2.15 The four basic design modules
one has to choose one out of n alternative operations. A decision-free choice Petri net module models a situation where there are n choices for a successive operation, with a strict order among them, depending on the initial marking. It can be shown in Fig. 2.15(d) that after firing transition \( t_1 \), which is the only enabled transition, transition \( t_2 \) becomes the only enabled transition in the module. This order of transition enabling continues until transition \( t_n \). After \( t_n \) fires, \( t_1 \) is enabled again and the cycle can repeat itself in the same order.

After this brief overview of the basic design modules, one needs to describe the two mutual exclusion structures that are used in the hybrid synthesis approach. These are defined as the parallel mutual exclusion (PME) and the sequential mutual exclusion (SME). These two resource-sharing concepts were first introduced in the work of Zhou and DiCesare (1991). The authors formulated these structures in the context of Petri net theory and applied them to the modeling problems encountered in resource-sharing manufacturing environments. The PME structure was used to model a resource that is shared by different processes working in parallel. On the other hand, the SME structure was used to model a resource shared by sequentially related groups of processes. An example of these two mutual exclusion structures is given below in Fig. 2.16. Apart from formulating these two resource-sharing concepts, Zhou and DiCesare established two theorems concerning the preservation of the properties of a Petri net containing a parallel or a sequential mutual exclusion structure. These two theorems proved that adding any one of these two structures will not cause the loss of the net model properties of boundedness, liveness, and reversibility, given that they existed in the original Petri net. In addition, the authors applied their concepts on two simple manufacturing systems.

Following this brief description of the basic design modules and the mutual exclusion structures, one can start explaining the hybrid synthesis procedure. Zhou, DiCesare, and Desrochers (1992) consider the specification of a manufacturing system that depicts its intended behavior as a list of resources, a list of operations, and their precedence relationships. Operations and resources are distinguished as
two elementary components. The proposed hybrid methodology treats operation places and resource places separately, such that a Petri net can be constructed and refined in a top-down manner at the initial stage and in a bottom-up manner in the last stage. Thus the hybrid methodology can be divided into the following five steps. First, choose a bounded, live, and reversible Petri net as a first level model of the system. Second, decompose the system into several subsystems expressed as operation places using the basic design modules. Third, add the non-shared resource places whenever one or several operations require the resources. Fourth, add the shared resource places which form PMEs. Fifth, add the shared resource places which form the SMEs. Thus we find that the hybrid approach models and refines operation places in a top-down manner while it adds shared resources in a bottom-up manner. In both the works of Zhou and DiCesare (1990) and Zhou, DiCesare, and Desrochers (1992), the hybrid synthesis approach is applied on a manufacturing system which incorporates shared-resources such as machines, robots, automated guided vehicles, and buffers.

As mentioned earlier, the hybrid approach has several advantages. It simplifies the designing of the Petri net model, it preserves important model properties, and it can deal with shared-resource systems. Nevertheless, the hybrid technique has some limitations. One such limitation, is that when the interaction between subsystems is very complex, as in some manufacturing systems, the construction of the initial Petri net becomes a very difficult task. Another problem

Figure 2.16 Two mutual exclusion examples
appears when dealing with flexible manufacturing systems. In these systems, there are many distinct resources (e.g. machines, robots, AGVs), whose numbers are more than one and are shared among several processes. This poses a problem for the synthesis of a Petri net with the desirable properties. In other words, a highly shared-resource system is still not easily modeled using the hybrid synthesis approach. However, one needs to point out the fact that the hybrid approach has been introduced rather recently, and that future research in this direction could lead to a solution for these problems.

Zhou and DiCesare (1990) were the first to indicate the problem of modeling flexible manufacturing systems due to their complex and highly interactive nature. This led them to conclude their paper with the final statement that predicted that the new thrust for research in the field of Petri net synthesis will be how to incorporate scheduling into the current approaches. Thus we arrive at the last section of this literature survey which considers the research involved in the scheduling field using Petri nets.

2.2 Applying Petri Nets in the Scheduling of Manufacturing Systems

Petri nets were recognized as an appropriate tool for the modeling and analysis of manufacturing systems that exhibited properties such as concurrency and synchronization. The work in this field was demonstrated in the previous section. In addition, Petri nets were viewed as a promising tool in the field of scheduling automated manufacturing systems, including flexible manufacturing systems (FMSs). However, the actual generation of schedules has not been given much attention in the Petri net community (Lee and DiCesare 1994a). Therefore, as mentioned earlier, the application of Petri nets in the field of scheduling is a relatively young research area.

One believes that the following literature review covers the majority, if not all, the work that was conducted in the field of applying Petri nets to the scheduling of manufacturing systems (specifically FMSs). The relevant research will be presented
in chronological order; with the exception of a couple of papers (mainly the work of Lee and DiCesare) which will be discussed separately, due to their direct association with this study. The following survey displays the various approaches that were used to tackle the combinatorial optimization problem of scheduling. One can observe the different types of Petri nets, the different constraints, and the different scheduling techniques that were utilized in this field.

The work of Shih and Sekiguchi (1991) can be regarded as the first attempt to use Petri nets in the scheduling of flexible manufacturing systems. The authors claim that through their bibliographical research, they failed to find any work that proposed the use of Petri nets as a scheduling tool for FMSs. In addition, the authors state that they are the first to propose a scheduling approach which takes in consideration the routing flexibility, which is a fundamental characteristic of FMSs.\(^1\) The on-line FMS scheduling system, proposed by the authors, simulates the evolution of the system using a Timed Petri net (TPN) model of the production system. The TPN employed by the authors was a deterministic timed Petri net where the time delays were associated with the transitions. In addition, their scheduling approach employs a beam-search based decision method, to solve the scheduling conflicts such as the assignment of jobs to machines or the priority to use system resources such as machines or AGVs. The proposed system is illustrated as a flowchart in Fig. 2.17. The beam search used is an Artificial Intelligence (AI) technique which develops a small number of possible solutions in parallel in order to find a good solution (i.e. schedule) with minimum search effort. The authors present the concepts of beam-width and beam-depth which are used in their beam search algorithm to solve the scheduling conflicts. The scheduling objective set in this research is Just-in-time production, such that any earliness or tardiness is penalized. The scheduling approach is applied on an FMS example, which provided results better than those obtained by simple priority rules such as FIFO and dynamic slack.

\(^1\) Nakamura et al (1988) proposed a scheduling approach which utilized Petri nets but did not incorporate routing flexibility. Thus, it could not be used to schedule FMSs.
Thus, one can see that the proposed scheduling system performs a beam search routine whenever there is a conflict. This routine constructs partial schedules within a specified beam-depth and uses an evaluation function to choose the best solution. This cycle is repeated until the final schedule is reached. However, the main problem with this method is that it is based on partial schedules and therefore it does not guarantee global optimization.

The work of Hatono, Yamagata, and Tamura (1991) was also concerned with the topic of on-line scheduling of FMSs. However, the authors used Stochastic Petri nets (SPN) as the modeling tool of the manufacturing system. They developed continuous-time and discrete-time stochastic Petri nets, in order to describe the uncertain events in the FMS such as machine tool failure, repair time, and even processing time. In addition, a heuristic dispatching rule was introduced into the model to represent the "heuristic" behaviors in the FMS. In order to obtain an efficient schedule of the FMS in an on-line real-time basis, a rule base was constructed and its performance was evaluated using the FMS simulation model that was proposed. The scheduling objective was JIT production which is similar to the work of Shih and Sekiguchi. It should be noted, that the Petri net model was constructed in a hierarchical manner in order to simplify the simulation task. They divided the SPN model into two levels. The first level was the transporting level and it was represented by discrete-time stochastic Petri nets. The second was the
processing level which was considered as a submodel of the transporting level, and was represented using continuous-time stochastic Petri nets.

Obviously, the main advantage of the scheduling approach developed by Hatono, Yamagata, and Tamura is that it takes into account the uncertainty involved in the manufacturing process. Moreover, their proposed technique takes place online rather than off-line which provides a quicker reflex to any problems that could take place whilst processing and allows for the system to adapt to any unexpected changes. Nevertheless, this scheduling technique has a major disadvantage, which is considering the FMS to possess fixed routings. In other words for each part/job there is a fixed sequence of machines and machining operations that are required to produce it. Obviously, the feature of fixed routings drastically reduces the efficiency and productivity of the FMS. Moreover, this constraint eliminates the very concept of flexibility, and deprives the FMS form one of its major advantages.

Raju and Chetty (1993) developed Priority nets for both the modeling and simulation of flexible manufacturing systems. A dynamic scheduling algorithm, employing the Priority net aided simulation, is used to determine the operating policies under varying conditions. The concept of Priority nets is regarded as an extension to timed Petri nets, such that dynamic decision-making and flexibility are added to the modeling structure. These Priority nets were allowed to handle both ordinary tokens as in timed Petri nets, and colored tokens as in colored Petri nets. In order to deal with the conflicts that occur among the competing elements during the operation of the FMS, decision places and decision transitions were introduced to implement certain algorithms to resolve these conflicts. Time was only associated with transitions in the Priority net model, and it was included in the form of deterministic values. Furthermore, Raju and Chetty suggest a two-level hierarchical approach for the modeling phase. The higher level, called the system net, considers the operations of the machining centers/cells. The lower level, called the logistic net, considers the transportation system (e.g. AGVs) and the FMS layout. The two subnet models are linked through the load/unload transitions. Thus this modeling technique follows the bottom-up approach. A software package, named
Priority Net Aided Flexible Simulator (PAFS), was developed to generate the model and simulate the system.

The on-line scheduling method proposed by Raju and Chetty was divided into a four-level hierarchical structure of decision making. The first level considers the loading strategy, whereas the second level considers the selection of the appropriate machine tool from amongst the candidate machine tools. The third level deals with the selection of the job for processing from amongst the competing jobs, whereas the fourth level considers the assignment of transport devices and the selection of the job to be transported. For each level there are several decision rules which could be applied. For example, the rules considered for the machine selection (level 2) and job selection (level 3) are given below in Table 2.1.

**Table 2.1 Scheduling / Decision Rules**

<table>
<thead>
<tr>
<th>Machine Selection Rules</th>
<th>Job selection Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Least utilized machine (LUM)</td>
<td>First-in first-out (FIFO)</td>
</tr>
<tr>
<td>Machine with minimum remaining input queue size (MRIQS)</td>
<td>Shortest processing time (SPT)</td>
</tr>
<tr>
<td>Machine with shortest processing time (MSPT)</td>
<td>Total processing time (TPT)</td>
</tr>
<tr>
<td>Machine with minimum input queue processing time (MIPT)</td>
<td>Remaining processing time (RPT)</td>
</tr>
<tr>
<td>Random machine (RM)</td>
<td>Random job (RJ)</td>
</tr>
</tbody>
</table>

Raju and Chetty demonstrate their Priority net based on-line scheduling technique through its application on an FMS with six machining centers, a cell, and a load/unload station. Several combinations of decision rules (one at each level) are evaluated using their proposed software (PAFS) which offers several indicative measures such as the makespan, the average machine utilization, and the average vehicle utilization. In addition the effect of different lot sizes and different product mixes is studied.

The advantages of the scheduling tool provided by Raju and Chetty lies in the fact that it takes place on-line, provides a hierarchical modeling methodology,
provides flexible decision-making capability, and provides an evaluation mechanism of the different scheduling rules. Another advantage is that their proposed software can accommodate variable processing times and can handle machine failures. Nevertheless, this approach utilizes decision rules which are simple heuristics that do not guarantee optimality. In addition there exists the problem of determining the best possible combination of rules. This difficult task requires an enormous number of simulation trials. Moreover, for each situation (i.e. product mix or volume mix) the same combination of rules will not perform with equal efficiency and effectiveness. The last problem is that some rule-combinations result in deadlocks, and the system can only be retrieved manually.

Oba and Ma (1994) proposed a similar technique for the scheduling of advanced manufacturing systems. The model utilizes Colored Petri nets (CPN) and some dispatching rules for the scheduling process. The manufacturing system considered is composed of several versatile machines and a material handling system using AGVs. The authors considered two types of problems which need to be resolved. The first problem/conflict occurs when one has to choose from several parts that require a single resource such as a machine or an AGV. The second conflict occurs when one has to choose from between several available resources (machines or AGVs) to process or transport a certain part. In their work, Oba and Ma used the dispatching rules for scheduling to resolve these conflicts. The proposed modeling, simulation and scheduling system is depicted below in Fig. 2.18.

It should be noted that Oba and Ma used a simple basic module, representing the configuration of a manufacturing cell, to build the CPN model. These basic modules are linked together to form the whole system model. Thus, the authors used a bottom-up modeling approach. The objective of the scheduling process in their work was to meet the given due dates (i.e. minimize tardiness). One of the advantages of this technique is that utilizing CPNs renders very compact system models. Another advantage is the independence of the net structure from the part-file and the AGV-file, which offers the flexibility of changing information
related to parts and AGVs without having to modify the net structure. However, due to the fact that dispatching rules and simulation were used, there remains the same problems that were previously mentioned regarding the technique provided by Raju and Chetty (1993). In addition, the work of Oba and Ma has another disadvantage which is not considering the problem of AGV collision.

DiCesare, Graves, and Gile (1996) present an interactive multimedia learning module for manufacturing scheduling. The authors offered an interactive multimedia tutorial on manufacturing scheduling that included a simulation of an electronics board manufacturing shop. Although this is an educational tool and is not involved
directly in FMS scheduling, yet the decision to consider this research is based on
the fact that the modeling, simulation, and scheduling processes were based on a
Petri net model. The proposed module is divided into three components which are
the scheduling concepts, the scheduling methods, and the electronic board factory
simulation. The first two components are simple interactive educational sections that
introduce the students/users to the common terms used in scheduling problems. In
addition these two components provide descriptions and examples of the possible
scheduling objectives and solution techniques that could be used. Although the Petri
net model that is used is a timed-transition Petri net (TTPN), yet the simulation
phase allows for stochastic processing times and machine breakdowns. The Petri
net model is constructed in a hierarchical manner where macro-places or macro-
transitions are replaced by subnets. Thus, the authors utilize the top-down
approach. The Petri net based simulator is “run” after the user decides on the
dispatching rules which will be used. Therefore, the simulation process helps the
user in determining whether the scheduling rules that were chosen achieved an
acceptable performance or not. The main disadvantages of this scheduling
technique stem from the fact that it does not deal with FMSs. Therefore, one finds
that this technique does not consider the material handling system (e.g. AGVs) or
transportation times. In addition heuristic dispatching rules are used, which do not
guarantee optimality.

The work and research of Lee and DiCesare (1992) was deferred up to this
point in order to consider separately the several advances they have made in their
proposed technique. Lee and DiCesare introduced a scheduling technique for
flexible manufacturing systems through the combination of Petri net modeling and
heuristic search. They recognized Petri nets as the ideal tool for formulating an FMS
scheduling problem, and used a modular bottom-up approach for the synthesis of
their timed-place Petri net (TPPN) model. In order to circumvent the problem of state
explosion, Lee and DiCesare propose the application of a heuristic search algorithm
which will only generate the necessary portion of the reachability graph such that it
finds the optimal schedule or a near-optimal schedule. This schedule will be given in
the form of a firing sequence of transitions that form a path from the initial marking to
the final marking. The proposed search algorithm was called the L1 algorithm and
was derived from the graph search algorithm A*. The L1 algorithm used three functions which were f, g, and h. Due to the fact that the scheduling objective of the authors was to minimize the makespan, the function \( f(m) \) was defined as the estimate of time for an optimal path starting from the initial marking and ending at the final marking which passes through the marking \( m \). The value of \( f(m) \) was calculated through the summation of \( g(m) \) and \( h(m) \), where \( g(m) \) was the minimum cost (time) from the initial marking to the current marking \( m \), and \( h(m) \) was the estimate of the time remaining to reach the final marking from the current marking \( m \).

In their work Lee and DiCesare used several heuristic functions, \( h(m) \), to find an acceptable schedule for two jobs on a simple FMS composed of three machines. They reached the result that the best heuristic function, in terms of reducing the search time (i.e. schedule generation time) was the following:

\[
\text{h (m) = - dep (m) ; where (Equation 2.3)}
\]

\[
\text{dep (m) = The depth of the marking m in the reachability graph}
\]

Obviously, the main advantage of the scheduling technique provided by Lee and DiCesare (1992) is that it addresses the global optimization of the schedule problem. This global perspective was not considered in the previous techniques due to the fact that the beam-search method and heuristic dispatch rules only consider local optimization. In addition, their proposed scheduling method and search algorithm can accommodate several performance criteria and objectives. However, Lee and DiCesare did not consider the material handling problem in their model and did not incorporate any stochastic measures to account for machine failure and variable processing time.

Lee and DiCesare (1993) extended their previously mentioned scheduling technique to include machine setup time in the Petri net model. They realized that ignoring setup times could cause significant errors in their results, in the case of having setup times that are comparable in duration to the machine processing times. The authors modified their TPPN model to incorporate setup times, in addition to the other features such as routing flexibility, shared resources, and concurrency. The
same scheduling objective and algorithm were used. However, the heuristic function which was used in their previous paper was modified. The new heuristic function is given below:

$$ h(m) = - w \cdot \text{dep}(m); \text{ where} $$  

(Equation 2.4)

\[ \text{dep}(m) = \text{The depth of the marking m in the reachability graph} \]

\[ w = \text{The weighting factor} \]

The authors use this new heuristic function and the algorithm L1 on the same example which they had in their previous paper. Although the authors added the inclusion of setup times to their previous work, yet the same disadvantages are existent.

In 1994 Lee and DiCesare had two more papers published regarding their proposed scheduling technique. The first paper by Lee and DiCesare (1994a) can be viewed as a more detailed treatment of their approach. Several examples are solved using their approach, including more complex and larger FMS configurations. Their heuristic function is implemented on several problems with either fixed or variable lot sizes and the conclusion that is reached is that the search algorithm performs favorably when the value of the weighting factor, \( w \), is equal to, or greater than 5. This result was reached on the basis of the ability of the heuristic search to compromise between the time required to reach a solution and the quality of that solution (i.e. the resultant schedule). In addition, they claim that their approach eliminates the analytical overhead required to guarantee the liveness of the model and the absence of deadlock in the system. However, the paper is concluded with the statement that the model is deficient due to the fact that it does not consider the material handling system within the FMS structure, and that this issue is under research.

Accordingly, in their second paper, Lee and DiCesare (1994b) introduced their conception of the integrated scheduling of flexible manufacturing systems. The authors extended their previously described scheduling approach to include the material handling system. Thus this approach was used to combine the part
processing facilities and the material handling system in one coherent model. Automated guided vehicles (AGVs) were considered to be the material handling devices used in the FMS scheduling problems. In addition the authors specified the centralized and the distributed automated guided vehicle systems (AGVS). These two types of AGVS were integrated into the Petri net model for part processing. The same type of Petri net model, namely the TPPN model; and the same scheduling objective, namely minimizing the makespan, were used. However, new places were added to the model to represent AGV availability, path availability, and transportation activities. In addition the modification to the Petri net model in the case of having insignificant transportation times was presented. Lee and DiCesare used the same search algorithm, namely L1, yet they utilized a different heuristic function. This new heuristic function is given in the following equation form:

\[
h(m) = w \cdot A_l \cdot [E_l - \text{dep}(m)] ; \text{where} \quad (\text{Equation 2.5})
\]

- \text{dep}(m) = \text{The depth of the marking } m \text{ in the reachability graph}
- w = \text{The weighting factor}
- A_l = \text{The mean of all possible processing and travel times}
- E_l = \text{The average number of transition firings/token from the initial to final place}

This heuristic function was used on a given problem and several values of the weighting factor, w, were experimented. In addition a comparison was conducted between the results obtained from the proposed scheduling approach and those achieved using heuristic dispatch rules such as shortest queue length, shortest waiting time, and shortest processing time. In all three cases the proposed scheduling method renders better results in terms of production rate than the corresponding heuristic dispatch rule. Certainly, the main advantage of this proposed scheduling method is that it includes the material handling system, and therefore offers a schedule which takes full advantage of the resources and the flexibility of the manufacturing system. However, the two proposed AGVS offer a limiting constraint. This is due to the fact that in the case of the centralized AGVS each AGV is devoted to a particular job until it is completed, whereas in the case of the distributed AGVS each AGV is devoted to a particular machine to transfer any
finished parts and return. Another limitation is the assumption that there are direct paths between all the machines, in addition to the simple traffic control mechanisms that are used. The final disadvantage is that the Petri net model remains without any probabilistic functions to account for the possible variability in the machining and transportation times.

The work of Sun, Cheng and Fu (1994) is the last research to be considered in this literature survey. Their work is directly related to the work of Lee and DiCesare. Sun, Cheng and Fu use the same scheduling approach which combines between the use of a Petri net model and a heuristic search algorithm. In addition the authors provide a modeling technique which takes into account the material handling system in their TTPN model. Nevertheless, the research of Sun, Cheng and Fu has several features which differentiate their work from that of Lee and DiCesare(1994b). Firstly, the authors use the bottom-up synthesis approach, where they divide their model into two submodels which are the stationary transportation model and the variable process flow model. The transportation model which describes the motion of the AGVs is further decomposed into three micro-models which are the transportation layout module, the movement control module, and the push control module. The aim of the transportation layout module is to model the layout of the transportation system. The purpose of the movement control module is to model the AGV routing, whereas the purpose of the push control module is to prevent any AGV traffic jams from occurring. The process flow model deals with the part routing and processing. After constructing the Petri net model the limited-expansion A algorithm, which is a modification of the L1 algorithm, was utilized to generate the schedule. The heuristic function given in Equation 2.4 was used, and the scheduling objective was set to be the minimization of the makespan.

Comparing the work of Sun, Cheng and Fu (1994) to that of Lee and DiCesare (1994b), one finds several advantages. The first advantage is that Sun, Cheng, and Fu did not presume any particular type of AGVS. Obviously, this resulted in a more complex model which required several control mechanisms to ensure that AGV collision and deadlock did not occur. In addition, their modeling technique allowed for different layouts of the FMS without having to assume that
there are routes between each and every machine. Finally, their search algorithm required less computational resources and memory. Nevertheless, their work does have some disadvantages. The first disadvantage is that the use of the limited-expansion A algorithm results in a trade-off between the quality/optimality of the resultant schedule and the schedule generation time. Another disadvantage is that their approach/model is based on deterministic time, and does not consider any stochastic measures that are more realistic in representing a manufacturing system such as an FMS.

Thus one has completed the survey of the available research involved in the field of Petri net synthesis, and the application of Petri nets in the field of scheduling manufacturing systems. The next section will describe the goals of this study and relate it to the previous research.

2.3 Objective

One finds it appropriate to restate the problem which is being treated by this study at this stage. This study aims at finding a schedule which minimizes the makespan of a given set of jobs that are to be processed on a given FMS employing a given AGVS. The modeling of the problem will be conducted using Petri nets as the modeling tool. After the modeling phase is completed, a heuristic search algorithm will be applied on the Petri net model in order to render the required schedule.

Thus it can be seen that the scheduling approach proposed in this study is mainly based on the method developed by Lee and DiCesare (1992). However, in this study several modifications and advancements are introduced. In the modeling phase, which is discussed in detail in Chapter 3, a modular bottom-up approach is utilized. This approach constructs the Petri net model in a modular form and connects the different modules through their common transitions or places. The Petri net model will have the flexibility to accommodate different types of AGVS. These will include the two forms introduced by Lee and DiCesare (1994b) in
addition to the less restricted AGVS proposed by Sun, Cheng, and Fu. Furthermore, the proposed model will have the flexibility of accommodating machine buffers of any size. The option of not considering AGVs or buffers will be provided to the user. In addition the Petri net model will not presume any fixed layout which will offer more flexibility for the user.

As for the heuristic search algorithm, which will be described in Chapter 4, the user will have the option of using the L1 algorithm or the Limited-Expansion A algorithm. Thus the user can compare the results of the two algorithms in terms of the quality of the resultant schedule and the required generation time. In addition, in the case of the Limited-Expansion A algorithm the user will have the option of changing the value of b, which is the maximum capacity of the OPEN list (explained in Chapter 4). The heuristic function which will be used for the search is given in Equation 2.4. A sensitivity analysis regarding the value of w, the weighting factor will be conducted. Through this analysis, an appropriate value of the weighting factor will be suggested to the user along with the option of changing this value. Finally, the schedule will be given in the form of transition firings which will define the order of events that are to take place.

Due to the computational and modeling complexity that is involved in this scheduling technique, a computer program will be developed to assist in the construction of the Petri net model, and to perform the heuristic search. The program will ask the user for the necessary input regarding the FMS configuration and the required information related to the specified set of jobs. The program will then construct the Petri net model, and apply to it the specified heuristic search algorithm. After reaching a final schedule, which is aimed at minimizing the makespan, the program will display the schedule along with the total time duration to complete the schedule. The program will also display the time that was required to generate the schedule and the number of iterations that were performed.

In addition to being a scheduling tool, one can utilize this program as a design tool. This option is available due to the fact that the user can specify the specific type of AGVS which is to be used in addition to the specific layout. Thus
through changing these parameters the user can determine the best AGVS and layout that satisfy his requirements. However, one has to keep in mind that the criterion for judgment in this study will be the schedule completion time, or in other words the production rate that is achieved from the resultant schedule.

The development of the computer program with all the features that were mentioned above can be regarded as an addition offered by this study. However, this study also attempts to explore a totally new field. It was shown that all the work of Lee and DiCesare, along with the work of Sun, Cheng, and Fu, was based on the application of timed-place Petri nets (TPPN) as the modeling tool. However, after examining and analyzing the heuristic search algorithm, one believes that the application of Colored Petri nets (CPN) will help in generating the final schedule in a shorter period of time. Although CPNs offer no additional modeling power yet their advantage in the modeling field is known because of their ability to create a more structured and compact model. One believes that the compactness of the model could help in increasing the speed of the search algorithm. This point will be elaborated during the description of the heuristic search algorithm in Chapter 4. However, at this point one cannot ascertain whether this reduction in the schedule generation time will take place or not. Moreover, if this reduction does occur, one should determine if it is of significant value. In addition, one needs to consider the total amount of time which is the summation of the time required to construct the model and the time required to generate the schedule.

Thus, this study will attempt to construct two models. One using TPPNs and the other using CPNs. After applying the search algorithm a comparison between the two approaches will be conducted. Obviously, the criterion on which this comparison will be held is the summation of the model-construction and schedule-generation time. Finally, on the basis of this comparison one will be able to conclude whether the application of CPNs in this scheduling approach is a viable alternative or not.
Chapter 3
THE PETRI NET MODEL

This research has several goals which it seeks to accomplish. The main goal is to provide a scheduling technique for both jobs and AGVs in a given flexible manufacturing system (FMS). The proposed scheduling approach relies on Petri nets for modeling the manufacturing system under consideration. The Petri net model takes into account both the part-processing facilities (e.g. machines) and the material handling system (AGVs). After the Petri net model is constructed, a heuristic search algorithm is implemented on the model in order to provide the schedule that satisfies the given objectives. Thus the final schedule will offer a combined solution for the efficient handling of both the machining resources and the AGVs.

In this research two different types of Petri nets are used to perform the modeling process. The first type is the timed place Petri net (TPPN) whereas the second type is the colored Petri net (CPN). TPPNs have been used as the modeling tool in similar scheduling techniques. They have been utilized in the work of Lee and DiCesare (1994b) in addition to the work of Sun, Cheng, and Fu (1994). On the other hand, CPNs have been used by Oba and Ma (1994) to model FMSs. However, despite the fact that the work of Oba and Ma provided a scheduling approach (based on heuristic dispatch rules), yet CPNs have not been used with the heuristic search algorithm that is described in this research. Therefore, the other main goal of this research is to determine if the decision of using CPNs, rather than TPPNs, as the modeling tool offers favorable results. The comparison between the two models will be based on the time required to generate the final schedule.

Thus, this research can be divided into two main parts, which are the model construction section and the heuristic search section. However, in order to implement the suggested scheduling approach using a computer program, this research is divided into three inter-related sections which can be illustrated by the flowchart depicted in
Fig. 3.1. This chapter is concerned with the first two blocks in Fig. 3.1, which are the definition/specification of the problem and the construction of the corresponding Petri net model.

![Diagram]

**Figure 3.1 The proposed scheduling approach**

In this chapter one will attempt to describe the modeling approach that will be used to build the Petri net. First, a listing of the questions that will be used to specify and define the scheduling problem will be given. The possible/allowable answers will be discussed along with their implications. This will be followed by the discussion of the methodology that will be used to construct the TPPN or the CPN model. The necessary steps will be described in detail along with the proposed building modules. However, before starting the explanation one has to present the fundamental assumptions on which this modeling approach has been based.

### 3.1 The Assumptions

Several assumptions have been made in order to determine the limits and boundaries within which this scheduling approach will/can be implemented. Obviously,
these assumptions affect the modeling approach and have a direct impact on the set of constraints that will be considered and accordingly on those that will be neglected. Therefore these assumptions play an important role in shaping this work and setting the area of research on which it will focus. The main assumptions are listed and explained as follows:

1. Only machining and transportation operations are considered to take place within the FMS. Thus assembly operations are not within the scope of this modeling / scheduling approach.

2. Set-up times are considered to be negligible compared to the machine processing times. In the case of FMSs, this assumption is usually valid.

3. Machining tools are assumed to be abundantly available. In other words the tool constraint is neglected and it is assumed that there is no resource competition occurring regarding the tools. This assumption is justifiable if every machine has its own tool magazine or if there is a large centralized tool storage area.

4. Job preemption is not allowed. In other words if a certain machine starts processing a certain part, then this operation has to be completed before the machine can start processing another part. Thus a machine can only release a part after finishing the required operations.

5. Processing and transportation times are constant. The Petri net models can only incorporate deterministic values. Thus no variation or probabilistic functions can be associated with the processing and transportation times.

6. Machine failure/repair is not considered. Thus in this scheduling approach all machines are assumed to be operating and functioning properly. No probabilities are associated with their working condition or status.

7. At the initial state of the system all resources are available (i.e. machines, AGVs, buffer places, robots). In other words the FMS is not pre-loaded with any jobs. In addition all jobs are ready at the inlet of the FMS and can be allowed to enter the system at any time-instant.
8. Blockage cannot occur at the unloading (exit) area of the FMS. In other words, finished products are immediately removed from the unloading area.

9. Buffer capacities are fixed and are independent of part types and part sizes. For example, if a buffer capacity is \( c \), then this buffer can accommodate \( c \) parts of any type, or any combination of parts as long as their sum is less than or equal to \( c \).

10. Each machine in the FMS has its own input and output buffer area. This assumption is applied in the case of independent machines (i.e., stand-alone machines) and in the case of machining cells. Thus, no common buffer is allowed for the machines that form a machining cell.

11. The transportation of parts between machines residing in the same machining cell is performed using robots. Thus, in this research, robots are used as a transportation tool. In addition, each machining cell can have only one robot.

12. The job lot-sizes are fixed, which means that the specified lot-sizes that are initially inputted by the user cannot be altered during the operation of the FMS (i.e., during the schedule time duration).

13. Each path connecting any two machines to each other can accommodate only one AGV at a time, regardless of the length of that path. This precautionary measure helps in avoiding the danger of AGV collision.

14. The transportation time along any path is assumed to be constant and fixed. It is independent of the type or size of the part that is being transported. In other words, the speed of the AGVs is assumed to be constant.

15. The time required for loading and unloading a part is added to the corresponding transportation time.

16. Each AGV can carry only one part at a time, and has to deliver this part to its destination before attempting to transport another part. Thus the carrying capacity of the AGV is not affected by the different part types or the different part sizes.

17. In the case of a centralized automated guided vehicle system (AGVS), each AGV belongs to the job/part that it is carrying. In other words, the AGV remains with the part from the instant that it enters the FMS until it becomes a final product and is
delivered to the unloading area (i.e. leaves the system). After delivering the part to the unloading area the AGV can carry another part and repeat the same cycle.

18. In the case of a distributed AGVS each AGV belongs to a certain machine or the loading (input) area. In such a case, whenever a machine M finishes processing a certain part, the AGV that belongs to that machine transports that part to its next destination and then returns (empty) to the machine M. Similarly, when a part (raw material) enters the FMS an AGV belonging to the loading area transports the part to its destination and then returns to the loading area.

3.2 The Question/Answer Input Method

The Petri net model of the scheduling problem under consideration will be constructed using a modular step-by-step bottom-up approach. A computer program will be developed to facilitate this construction task. This computer program transforms the problem input data into a Petri net model which can then be used in combination with the heuristic search algorithm to generate the required schedule. Thus, the starting point for constructing the Petri net model is to allow for the specification of the manufacturing system under consideration. In order to ensure that the input data is given in a systematic manner, a set of questions, directed from the program to the user, have been proposed. These questions help in determining all the required information pertaining to the FMS configuration and the scheduling problem. Moreover, through this question/answer input methodology the user can specify the set of constraints that he/she wishes to be applied during the construction of the Petri net model and accordingly the generation of the schedule. A simple listing of the proposed questions is given below:

1. What is the number of machines involved in this FMS?
2. What is the input/output buffer capacity of each machine?
3. What is the number of jobs (i.e. different tasks/parts) involved in this scheduling problem?
4. What is the lot-size for each job-type?
5. How many processes are required for each job-type?
6. Which resource(s) is (are) required to perform each process (i.e. operations)?
7. What is the duration time required for each operation?
8. Are automated guided vehicles (AGVs) involved in this scheduling problem?
9. Which type of automated guided vehicle system (AGVS) is used in the FMS?
10. What is the number of AGVs involved in this FMS?
11. Which type of AGV track layout is used in the FMS?
12. Are robots involved in this scheduling problem?
13. What is the number of robots involved in this FMS?
14. Which machines are served by each robot?
15. What is the transportation time from each machine to the other?

Operations and resources are the two main components that constitute any manufacturing system. Thus the specification of a manufacturing system can be achieved if one obtains a list of the available resources, and a list of the required operations, along with their technological precedence. The first two questions along with the questions ranging from 9 to 14 are aimed at acquiring a description of the FMS configuration. In other words these eight questions help in creating a list of the available resources along with an image of how they are interconnected (i.e. the FMS layout). The remaining questions, apart from number 8, are aimed at creating the list of operations. In the case of considering AGVs in the scheduling problem, operations can be divided into two classes. There can be either transportation operations or machining operations. The last question is aimed at creating the list of the transportation operations along with their time durations. On the other hand, questions 3 to 7 attempt to create a list of the machining operations that are required to produce each product, along with the technological precedence that is involved amongst these particular operations. In addition these questions provide the machining operation times.
CHAPTER 3

In fact, these five questions (i.e. questions 3 to 7) can be viewed as a systematic questioning technique that allows the user to enter to the program the process plan of each job (product) that is included in the scheduling problem. To clarify this point and accordingly introduce the modeling terminology that is used in this research, one needs to consider the information given below in Table 3.1. The information given in Table 3.1 is concerned with a manufacturing system that is composed of five machines (i.e. M1 - M5). Correspondingly, the process plans for three different products is given (i.e. Job 1, Job 2, Job 3) as follows:

Table 3.1 A Process Plan Example

<table>
<thead>
<tr>
<th>Process</th>
<th>J1</th>
<th>J2</th>
<th>J3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>M1 / M2</td>
<td>M2 / M5</td>
<td>M3</td>
</tr>
<tr>
<td>2</td>
<td>M3</td>
<td>M1 / M3</td>
<td>M1 / M4</td>
</tr>
<tr>
<td>3</td>
<td>M1 / M4</td>
<td>-</td>
<td>M2 / M5</td>
</tr>
</tbody>
</table>

It can be seen that the data conveyed in Table 3.1 represents a certain set of activities that have to be performed in a particular order in order to produce the finished product(s). In other words, Table 3.1 provides a sequence of processes, and each process sequence defines a certain product. It can be seen that Job 1 and Job 3 require three processes, whereas in order to produce Job 2 only two processes are required. It should be noted that at the level of processes (e.g. turning process, milling process, drilling process, etc.), one does not consider the machining resources. In the example given in Table 3.1 one finds that several processes can be conducted at more than one machine. For example, the first process of Job 2 can be conducted at either machine M2 or machine M5. This is a characteristic feature of FMSs which depicts the routing flexibility that is inherent in these manufacturing systems.

49
In this research the term “operation” is used for the consideration of the resource requirements needed to perform a specific process. Thus performing the first process of Job 2 at machine M2 is an operation, and similarly conducting the same process at machine M5 is considered to be a different operation.

Thus, questions 3 to 7 are simply aimed at transferring to the program the data that is present in the process plans (e.g. Table 3.1) of the products/jobs that are to be manufactured in the given FMS. Moreover, these five questions manage to obtain additional information such as the required production quantities (i.e. lot-size) for each product, along with the time duration required to perform each operation.

One of the main advantages of the proposed question list is that it offers the user several options regarding the set of constraints that will be used and applied during the solution of the given scheduling problem. This feature is apparent in questions 2 and 8. In question 2 the user has the option of setting all or some of the machine buffer capacities to infinity. Thus, the user has the option of totally or partially neglecting the buffer capacities’ constraint. In question 8 the user has the option of choosing to include or neglect the material handling system (i.e. the AGVs and robots) in his description and specification of the FMS. Thus the user can exclude or include the material handling constraint whilst solving the scheduling problem. Obviously, in the case of exclusion, the generated schedule will be a normal part (job) schedule. On the contrary, in the case of inclusion, the generated schedule will be in the form of an integrated schedule that considers both parts and AGVs.

Another advantage of the proposed question/answer methodology is that it can be used as aid in evaluating different design alternatives. The user has several design options to choose from. For example the user can see the effect of including machining cells within the FMS under consideration. In addition the user has the option of selecting the number of AGVs in addition to the specific track layout which is to be used for connecting the machines to each other. However, one has to point out that the
criterion that will be used to grade or judge the different design alternatives is the productivity. This is due to the fact that the scheduling objective of this research is minimizing the makespan (i.e. increasing the production rate). Thus, if the user places more weight on other measures such as machine utilization or meeting due dates, then this inputting method and program will be of no assistance.

In addition to being used as a design evaluation tool, this scheduling methodology can help in determining the best or most appropriate batch size to be used. In other words the scheduling program can assist in deciding on the part loading strategy that is to be used during the operation of the FMS. This advantage is offered through question 4, where the user is asked to input the lot-size for each part. Thereby the user can run several trials with different lot-sizes for each part and accordingly a comparison between the different product mixes can be conducted. Obviously, the criterion for comparison remains to be the scheduling objective that is set in this research, which is the makespan (i.e. total completion time).

After discussing the goals and advantages of the inputting method, one needs to consider the model construction stage. In this stage two different types of Petri net models can be constructed, namely, the timed place Petri net (TPPN) model or the colored Petri net model (CPN). Obviously, the treatment and the representation of the input data is different in the case of the two models. Nevertheless, there are some common features between the TPPN and the CPN models. These common features are considered in the following section.

3.3 The Construction Approach

In the proposed modeling strategy time is associated with places. Accordingly all transitions can fire immediately. This feature is inherent in the definition of TPPN, and in this research it is also applied in the case of the CPN models. Thus, places can assume certain time delays which affect the availability of the tokens that they carry. In
the case of timed places a token is only available for firing after the predetermined time delay passes. On the other hand, in the case of places with no time duration, it is assumed that the tokens they carry are readily available for the firing operation. An example depicting these characteristics is given below in Fig. 3.2.

In Fig. 3.2(a) the initial state of a TPPN is given. All the transitions and the places except for $p_1$ are assumed to have no time delays. As for place $p_1$, a time delay of two minutes is associated with it. It can be seen that there is no graphical distinction between TPPN models and ordinary Petri net models. In addition, one cannot distinguish from the graphical representation of the TPPN between immediate places and time-delayed places. This distinction is apparent from the interpretation (i.e. semantics) of the Petri net places and transitions. Therefore in the case of the Petri net given in Fig. 3.2 (a), only transition $t_2$ is enabled. This is due to the fact that the tokens in the input places of $t_2$ are all readily available, whereas in the case of transition $t_1$, the token in place $p_1$ will be available for firing after two minutes according to the provided interpretation. Thus in order to reach the marking given in Fig. 3.2 (b) one has to fire transition $t_2$. There is no time consumed in order to transform the initial marking to the given marking. However, in the case of deciding to fire transition $t_1$, first one has to wait two minutes for the token in place $p_1$ to be available. Thus the marking in Fig. 3.2 (b) can be reached instantaneously while the marking in Fig. 3.2 (c) requires a delay period of two minutes.

Thus, one can provide the general interpretation of transitions and places that will be used in the proposed Petri net models. Transitions will be used to represent events while places will represent conditions. Therefore, transitions will represent events such as the start (initiation) or the end (termination) of the involved activities. Places will represent conditions such as the performance of activities, availability of resources, and the readiness of parts for machining processes. Obviously, the activities under consideration are either machining operations or transportation operations.
(a) Initial state of a TPPN

(b) The effect of firing $t_2$ first

(c) The effect of firing $t_1$ first

Figure 3.2
Available resources include several items such as machines, AGVs, buffer places, robots, and parts. Thereby, time will only be associated with places representing activities. All other places will be immediate and the tokens they carry will be readily available.

Since the modeling approach that will be used is the bottom-up approach, the FMS under consideration will be divided into several subsystems. Each subsystem will be represented by a specific module. After the creation of all the required modules, the final Petri net model will be constructed through either merging or linking these modules. The merging process will take place through the sharing of common transitions or common places. On the other hand the linking process will take place through the addition of extra places or extra transitions. This bottom-up modular approach will be used for the construction of both types of Petri nets (TPPN and CPN). The proposed building modules (i.e. building blocks) are presented in the following section along with a detailed description of each module.

3.4 The Building Modules

In order to simplify the modeling stage of the given scheduling problem, the FMS under consideration is decomposed into two main subsystems. The first subsystem is concerned with the machining section whereas the second subsystem is concerned with the transportation section. In the machining subsystem one considers the buffer space resources and the machine resources. In the transportation subsystem one considers the available AGVs, robots and paths (i.e. AGV tracks). Basic building modules are proposed to be the fundamental blocks out of which the Petri net model will be constructed. Obviously, the shape of the different modules will be modified according to the data inputted by the user. The different modules and the possible structural modifications that could take place regarding each type are considered in the following sections.
3.4.1 The Machining Module

The general machining module, and the description of the involved places and transitions are given in Fig. 3.3 and Table 3.2, respectively.

![Diagram of the general machining module]

Figure 3.3 The general machining module
Table 3.2 The description of the general machining module

<table>
<thead>
<tr>
<th>Places</th>
<th>Description of places</th>
<th>Transitions</th>
<th>Description of transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>p₁</td>
<td>Input buffer occupied slots</td>
<td>t₁</td>
<td>Part entering input buffer</td>
</tr>
<tr>
<td>p₂</td>
<td>Input buffer empty slots</td>
<td>t₂</td>
<td>Start of machine operation</td>
</tr>
<tr>
<td>p₃</td>
<td>Machine operating</td>
<td>t₃</td>
<td>End of machine operation</td>
</tr>
<tr>
<td>p₄</td>
<td>Machine available</td>
<td>t₄</td>
<td>Part leaving output buffer</td>
</tr>
<tr>
<td>p₅</td>
<td>Output buffer occupied slots</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p₆</td>
<td>Output buffer empty slots</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The general machining module given in Fig. 3.3 is common to both the TPPN and the CPN model. However, the difference between the two models lies in the treatment and definition of the tokens that will be used. Obviously the main difference is that the tokens in the TPPN model are unidentifiable, whereas the tokens in the CPN model are identifiable (using the colors). This identification property which characterizes CPN models will have no effect on the structure of the machining module and therefore the logic behind the module is the same in both models.

The logic which sets the dynamics of the machining module is achieved through the directed arcs that connect the module components to each other. The connections between the transitions and places (and vice-versa) of the module define their relations and mutual dependence. Assuming that a part (represented as a token) needs to visit a certain machining module for the performance of a specific operation. First, this part has to enter the input buffer of the corresponding machine. The occurrence of this event is represented by the firing of transition \( t₁ \) in the machining module. However, this event cannot occur if the input buffer of the machining module is full. Therefore, the model has to check if the input buffer has any empty places before allowing the part to enter. This precondition is denoted in the model by the arc connecting place \( p₂ \) to
transition \( t_1 \). Place \( p_2 \) is defined in Table 3.2 as the place representing the availability of empty input buffer slots. The absence of tokens in place \( p_2 \) depicts the fact that there are no empty spaces in the input buffer of the machine, while on the other hand the presence of tokens means that there are empty slots in the input buffer equal to the number of present tokens. Using the Enabling and Firing rules, transition \( t_1 \) cannot fire unless there is at least one token in place \( p_2 \). Thus the model has ensured that no part can enter the machining module unless there is at least one empty place in the input buffer of the machine under consideration.

When transition \( t_1 \) fires, one token is removed from place \( p_2 \) and equally one token is added to place \( p_1 \). Place \( p_1 \) is defined as the place representing the occupied input buffer slots. Accordingly, the absence of tokens in place \( p_1 \) denotes the fact that there are no parts occupying the input buffer (i.e. the input buffer is empty), whereas the presence of tokens means that there are parts occupying the input buffer equal to the number of present tokens. Thus, the firing of \( t_1 \) causes an increase and a reduction of one token in the places \( p_1 \) and \( p_2 \), respectively. In physical manufacturing terms, this means that the entrance of a part in the input buffer of a specific machine reduces the number of empty places in that buffer by one. On the other hand, this entrance also causes the number of parts occupying the input buffer to increase by one.

Place \( p_4 \) represents the availability of the machine. The presence of a token in this place means that the condition is true (i.e. the machine is available). Accordingly, the absence of the token means that the condition does not hold (i.e. the machine is not available). On the other hand place \( p_3 \) represents a contradictory condition which is the machine operation (i.e. working). Similarly, the presence of a token in place \( p_3 \) means that the machine is working whereas the absence of a token means that it is idle. Obviously, no machine can be operating and available at the same time. Therefore, the setting of the initial marking of the model has to ensure that these two places (i.e. \( p_3 \) and \( p_4 \)) cannot be marked (i.e. carry tokens) at the same time instant. In other words there should be no marking, either initial or any subsequent marking, in
which both of these places are marked. This mutual exclusion relationship can be achieved by marking either \( p_3 \) or \( p_4 \) with one token in the initial marking. Due to the fact that it is assumed that in the initial state of the FMS all machines are available, one places the token in place \( p_4 \). For this reason, the only place that was marked in the general machining module given in Fig. 3.3 was \( p_4 \). This is to indicate the fact that this place is initially marked, without taking any consideration of the user's input. Although places \( p_5 \) and \( p_6 \) will also be marked in the initial marking, yet in their case the marking depends on the input of the user.

Transition \( t_2 \) is defined as the transition representing the start of the machining operation. In other words, a part is taken from the input buffer and placed on the machine for processing. From the arcs connected to this transition, one discovers that \( t_2 \) cannot fire unless there is a token in place \( p_1 \) and place \( p_4 \). Thus the module has set the preconditions of this event to be having a part in the input buffer and having the machine available. Upon the firing of transition \( t_2 \), one token is removed from places \( p_1 \) and \( p_4 \), and one token is deposited in places \( p_2 \) and \( p_3 \). In other words, the effect of the occurrence of this event (i.e. start of machining) is that the number of parts in the input buffer is reduced by one \((p_1)\) and accordingly the number of empty slots is increased by one \((p_2)\). In addition, this event causes a change in the status of the machine from being available and free \((p_4)\) to operating and busy \((p_3)\). It should also be noted that the module's topology along with the initial marking ensure that the machine will only be able to operate on one part at a time.

Transition \( t_3 \) is defined as the transition representing the end of the machining operation. It is assumed that this event also includes the placement of the machined part in the output buffer. On the other hand places \( p_5 \) and \( p_6 \) are similar to places \( p_1 \) and \( p_2 \). The former couple are designated for the output buffer using the same modeling interpretations that were used for assigning the latter couple for the input buffer. Thus, place \( p_5 \) is defined as the place representing the number of occupied slots in the output buffer, and place \( p_6 \) is defined as the place representing the number of
empty slots in the output buffer. For transition $t_3$ to fire there has to be a token in places $p_3$ and $p_6$. In other words, for the event represented by transition $t_3$ to take place there has to be a part that is being machined and there has to be at least one empty place in the output buffer.

Thus, after the machining operation is completed, transition $t_3$ is fired. The firing of $t_3$ causes the addition of one token to places $p_4$ and $p_5$, along with the removal of one token from $p_3$ and $p_6$. In other words, the effect of the occurrence of this event (i.e. end of machining) is that the number of parts in the output buffer is increased by one ($p_5$) and accordingly the number of empty slots in the output buffer will be decreased by one ($p_6$). In addition, this event causes a change in the status of the machine from being operating and busy ($p_3$) to being available and free ($p_4$).

Finally, transition $t_4$ is defined as the transition representing the event where a part leaves the output buffer of the corresponding machining module. In other words the part is leaving this specific machine to follow the route that was set for it by the proposed schedule. Concerning the machining module, transition $t_4$ is connected to two places. It has $p_5$ as an input place and $p_6$ as an output place. Therefore transition $t_4$ can only fire if there is a part in the output buffer (i.e. $p_5$ is marked). When this transition fires, a token is removed from place $p_5$ and correspondingly a token is added to place $p_6$. This denotes the fact that when a part leaves the machining module, the number of parts in the output buffer is reduced by one ($p_5$) and accordingly the number of empty spaces in the output buffer is increased by one ($p_6$).

Since the transitions in the proposed Petri net models have no time delays, all the transitions in the machining modules can fire immediately. As for the places, a time delay which is equal to the corresponding operation time is associated with place $p_3$. The other five places are assumed to be immediate places, in which tokens are readily available. In the case of place $p_3$ the token which it carries becomes available after a period of time equal to the machining operation time. Therefore, transition $t_3$ can only
fire after the inputted time delay period has passed, provided that all other preconditions are satisfied. For example, the machining operation could be completed but there is no empty space in the output buffer. The fact that the output buffer is full will be reflected by having no tokens in place $p_6$, and this would disable transition $t_3$. Thus, the part will be delayed and will remain on the machine until an empty slot becomes available in the output buffer. Two examples of a specific machining module are given in Fig. 3.4(a) and (b). The first presents the initial marking of a specific module where the capacities of the input and output buffers are set to four and three, respectively. The second presents an intermediate marking where the machine is operating; two places are occupied in the input buffer and one place is occupied in the output buffer.

(a) Initial marking

(b) Intermediate marking

Figure 3.4 A specific machining module
The general machining module given in Fig. 3.3 represents the general case that could be considered in the proposed modeling approach. As for the specific machining modules given in Fig. 3.4 (a) and (b), both cases follow the general module structure. However, there are several other cases where the general machining module will have to be modified. The case that will be considered at this stage is the machining module that will be formed when the user chooses to neglect the buffer capacity constraint. In this case, the general machining module is modified to the Petri net structure given in Fig. 3.5.

![Petri Net Diagram]

Figure 3.5 The machining module without the buffer capacity constraint
The difference between the machining module given in Fig. 3.5 and the general machining module is that the original places depicting the empty buffer slots have been removed. The role of places $p_2$ and $p_6$ in the general machining module was to set a maximum for the number of tokens that could reside in the input and output buffers (i.e., buffer capacity). Thus, with the removal of places $p_2$ and $p_6$ from the general machining module given in Fig. 3.3, one dismisses the buffer constraint. In Fig. 3.5 this is apparent from the fact that the input and output buffer places, $p_1$ and $p_4$, can now carry an unlimited number of tokens.

Thus one has examined the two main forms of the machining module. The other modifications that could take place on the general machining module occur as a result of the interaction of the machining subsystem with the transportation subsystem. Therefore, one needs to present the transportation modules before considering the modifications which they incur on the general machining module.

### 3.4.2 The Transportation Modules

In the case of the transportation subsystem, one cannot make a generalization that can be applied to the Petri net model. This is due to the fact that there could be two transportation devices available in the FMS. These two transportation devices are the AGVs and the robots. However, robots are only involved in the case of the presence of machining cells within the FMS. Each machining cell is assumed to have one robot, which is considered to be the transportation means between the machines included that specific machining cell.

In addition, the AGV transportation system to be used depends on the input provided by the user. The proposed modeling approach offers the option of choosing from three different AGVS (automated guided vehicle systems). Moreover, the flexibility of the modeling approach is increased by allowing one to choose from different types of track layouts, which define how the FMS machines are connected to each other. Thus
the AGV transportation module that will be used in the specified scheduling problem depends on the type of AGVS and the shape of the layout that will be chosen.

In this modeling approach three different AGVS are considered. The first type is the distributed AGVS where each AGV either belongs to a machine or to the loading (input) area. The second type is the centralized AGVS where each AGV belongs to a part. The third type is the “free floating” AGVS where the AGVs are left to move around the FMS freely. Obviously, the third case is the least-constrained and the most commonly used AGVS. For each one of the three different AGVS, different layouts are suggested by the program. Thus, the user has the freedom to pick the most suitable match of AGVS and track layout. However, for every particular match a different modeling module will be developed. A flowchart depicting the different transportation modules, along with the corresponding section/case number indicating where they will be presented and explained, is given in Fig. 3.6.

The flowchart in Fig. 3.6 displays all the transportation modules that are proposed in this research. In addition the flowchart presents the different layouts that are considered in this work. The full description and explanation of these layouts is given during the discussion of the corresponding AGVS.

3.4.2.1 The Distributed AGVS

In the distributed AGVS each machine has its own AGV. In addition one or more AGV is located at the loading (input) area where the raw material is stored. The decision of how many AGVs are to be located at the loading area is made by the user. The role of the AGVs in the FMS is to transfer the parts (whether in the raw-material, in-process, or finished product state) to the next destination point. Therefore, there are no AGVs located at the unloading (output) area. After the AGV performs the transportation process it immediately returns to its original location.
Figure 3.6 A flowchart of the different transportation modules

The general layout of the system is illustrated in Fig. 3.7. The arrows in the layout diagram depict the direction of flow of the AGVs through the FMS. The dashed arrows represent the flow of the AGVs that belong to the loading area, whereas the solid arrows represent the flow of the AGVs that belong to the machines. The direction of the flow is indicated by the arrow head. Thus AGVs at the loading area transfer the unprocessed parts (raw material) to the machines and return immediately back to their positions. On the other hand the AGVs that belong to the machines transport the finished parts (final product) to the unloading area and then return immediately to their original locations. The curved arrow within the block representing the machines of the FMS denotes the flow of the AGVs between the different machines. The parts at this
stage (in-process) are transported by several AGVs. After each machine completes the required processing operations the AGV that belongs to it transports the machined part to the following machine's input buffer. Thus AGVs deal with the outgoing, not the incoming, parts.

![Diagram](image)

**Figure 3.7 The FMS general layout (distributed AGVS case)**

It is assumed that each machine is connected directly to the other machines, if required. In other words if a part needs to be transported between two specific machines, then there will exist a direct path between them for the AGV to move along. Obviously, if no transportation operation occurs between any two machines, then there will be no AGV path connecting them to each other. However, the shape and number of connections between the different machines, loading area, and unloading area has a direct impact on the structure of the Petri net module that will be constructed. Therefore, two different cases of AGV track layouts are considered in the distributed AGVS.

**Case 1: Bi-directional two path layout**

In this case, if parts need to be transported between two specific machines, it is assumed that these two machines are connected to each other, as shown in Fig. 3.8. This particular layout (connection) is called the bi-directional two path layout. Each line indicates a separate AGV track. In this research the term path is equivalent to, and will be used interchangeably with, the term track.
In addition the arrow heads on each line indicate the direction of flow. Therefore, in this case the AGVs are allowed to move in both directions on the same track. Thus this layout was called “bi-directional” due to the fact that the AGVs can move in both directions on the same path. The term “two path” was used to indicate that two different paths exist between the connected machines, if required. In other words each machine AGV has its own path which can only be used by it. Therefore, in Fig. 3.8 one of the two tracks connecting machine $M_i$ to machine $M_j$ will be used by the AGV belonging to machine $M_i$. The other path will be utilized by the AGV belonging to machine $M_j$.

Assuming that a part needs to be transported from $M_i$ to $M_j$. The AGV located at machine $M_i$ will transport that part along its designated track and will return on the same track (bi-directional). If a part has to be transported from $M_j$ to $M_i$, then the transportation process can take place simultaneously with the previous one, due to the fact that the AGV belonging to machine $M_j$ can perform the process along the other path that is allocated for it. The transportation module depicting this particular system (i.e. AGVS + layout) is given in Fig. 3.9. A description of the places and transitions used in the module is provided in Table 3.3.

Similar to the machining modules, all the transportation modules will be used in both the TPPN and CPN models. Whether identifiable tokens are used or not, will have no effect on the structure of the transportation modules. Therefore the interpretation of the transitions and places remains the same in both Petri net models. A description of the transitions and places of the transportation module (Fig. 3.9) along with an explanation of the dynamics of the model, is given.
Figure 3.9 The transportation module for the distributed AGVS (Case 1)

Table 3.3 The description of the transportation module in Figure 3.9

<table>
<thead>
<tr>
<th>Places</th>
<th>Description of places</th>
<th>Transitions</th>
<th>Description of transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>AGV available</td>
<td>$t_1$</td>
<td>Start of transportation operation</td>
</tr>
<tr>
<td>$p_2$</td>
<td>AGV transporting</td>
<td>$t_2$</td>
<td>End of transportation operation</td>
</tr>
<tr>
<td>$p_3$</td>
<td>AGV returning</td>
<td>$t_3$</td>
<td>End of return operation</td>
</tr>
</tbody>
</table>

Transition $t_1$ is defined as the transition denoting the event of starting the transportation operation. Obviously, for this event to occur an AGV has to be available. In the case of the distributed AGVS, this AGV is either an AGV belonging to a machine or to the loading area. Therefore a directed arc is used to connect place $p_1$, which represents the availability of the AGV under consideration, to transition $t_1$. Thus by setting place $p_1$ as an input place to transition $t_1$ one ensures that the Petri net model will not start a transportation operation (fire $t_1$) unless the required AGV is available. In other words, if place $p_1$ is not marked, then the required AGV is not available (i.e. 67
engaged in another transportation process) and the transportation operation will have to be delayed.

The firing of transition \( t_1 \) results in the removal of one token from place \( p_1 \) and the deposition of one token in place \( p_2 \). Place \( p_2 \) is defined in Table 3.3 as the place representing the transportation operation. In other words the model manages to represent the fact that when a transportation operation commences, the AGV involved (i.e. being used) is no longer available. Thus, the model structure guarantees that none of the AGVs will be engaged in more than one transportation process at the same time.

Transition \( t_2 \) is defined as the transition representing the end of the transportation operation. In addition place \( p_3 \) is defined as the place depicting the returning operation of the AGV (i.e. heading back to its original location). After the completion of the transportation operation, transition \( t_2 \) is allowed to fire. The firing of transition \( t_2 \) results in the removal of one token from \( p_2 \) and the deposition of one token in place \( p_3 \). In other words the firing of transition \( t_3 \) marks the occurrence of two events that actually take place at the same instant. The first event is the end of the operation of transporting the part to the necessary destination. The second event is the beginning of the operation of returning, with the AGV being empty, to the original location. In this specific case the AGV returns along the same track which is dedicated for its use.

Transition \( t_3 \) is defined as the transition representing the end of the returning operation. Thus, the AGV performing the transportation operation has returned to its original location, which could be either a certain machine or the loading area. The firing of transition \( t_3 \) can only take place after the end of the returning operation. From the directed arcs of the transportation module, one finds that the result of firing \( t_3 \) is the removal of one token from place \( p_3 \) and the deposition of one token in place \( p_1 \). This results in the restoration of the initial marking (i.e. initial state) of the transportation module. The results of firing transition \( t_3 \) indicate that when the returning operation is
completed the AGV under consideration is available once more. Thus the AGV is ready to perform the next transportation operation assigned to it by the developed schedule.

In the transportation module given in Fig. 3.9, both places $p_2$ and $p_3$ have time delays associated with them. These time delays are equal to the time period required to perform the transportation and the returning operations, respectively. Thus a token in place $p_2$ becomes available after the time required for the transportation operation elapses. Similarly, a token in place $p_3$ becomes available after the time required for the returning operation elapses. On the other hand, place $p_1$ has no time delay associated with it. Therefore any token present in place $p_1$ is assumed to be readily available.

The merging of the transportation module given in Fig. 3.9 with the general machining module given in Fig. 3.3 is a very simple task. Obviously, this merging task is essential in order to combine the machining and transportation subsystems into one coherent model. Thus, the resulting model will be able to handle the problem of scheduling both jobs and AGVs. In this case the merging process will take place through the sharing of common transitions. It can be seen that transition $t_1$ in the general machining module and transition $t_2$ in the transportation module (Fig. 3.9) are common. The reason for this, is that the event of entering the input buffer of a machine and the event of ending the transportation operation, are coinciding events. Likewise, transition $t_4$ in the machining module and transition $t_5$ in the transportation module are common. Obviously, the events of leaving the output buffer of a machine and the commencement of a transportation operation take place simultaneously.

Assuming that a part needs to be transported to machine $M_i$ after visiting machine $M_j$. The integrated model representing the flow of the part is given in Fig. 3.10. It can be seen in Fig. 3.10 that transitions $t_4$ and $t_5$ are formed as a result of the merging of transitions in the machining module with transitions in the transportation module. Thus, transition $t_4$ represents the event of a part leaving the output buffer of machine $M_i$ and the commencement of the transportation operation from $M_i$ to $M_j$. 

69
Moreover, transition $t_8$ represents the events of ending the transportation operation, the beginning of the return operation, and the entrance of a part into the input buffer of machine $M_i$. It should be noted that the Petri net model given in Fig. 3.10 did not specify the buffer capacities of machines $M_i$ and $M_j$ for the purpose of generalization. It should be noted that time delays are only associated with places $p_3$, $p_8$, $p_9$, and $p_{12}$. These time delays are the machine operation time on $M_i$, the transportation time from $M_i$ to $M_j$, the return time from $M_j$ to $M_i$, and the machine operation time on $M_j$, respectively. Finally, it should be noted that the time to load the part at machine $M_i$ and to unload it at machine $M_j$ is added to the transportation time delay associated with place $p_8$.

The example considered in Fig. 3.10 considered the problem of modeling a part being transported from one machine to another. This case occurs when a process of a certain part can be conducted on only one machine, and similarly the following process can only be performed on a single machine. In other words, the problem that has just been considered does not include any form of routing flexibility. In the case that routing flexibility is included, some modifications have to take place with regards to the machining and transportation modules.

The first example of routing flexibility to be considered is when a process being performed on a single machine is followed by a process that can be performed on two machines. For example, process 1 of a certain part is performed on machine $M_i$, whereas process 2 can be performed either on machine $M_i$ or machine $M_k$. The modified integrated model (i.e. the combination of the machining and transportation modules) that represents such a situation is given in Fig. 3.11. The description of the corresponding places and transitions is given in Table 3.4. The notation ‘1-2’ is used to indicate that the first process is performed on a single machine while the second process can be performed on two different machines.
Figure 3.10 The integrated model for transporting a part with a distributed AGVS (Case 1)
Due to the space limitation, Fig. 3.11 only illustrates the portion of the integrated model that has been modified. Therefore the machining modules of machines $M_i$ and $M_k$ along with the first part of machine $M_i$ are not included. It can be seen that the module of machine $M_i$ has been modified at the output buffer section. Instead of having only one output transition indicating that a part has left the machine buffer, two transitions have been used. Accordingly, the places representing the output buffer occupied and empty slots have been connected to both transitions, as shown in Fig. 3.11. The reason for using two transitions is to indicate the fact that a part has the choice of going to either machine $M_i$ or machine $M_k$.

![Diagram of a Petri Net Model](image)

*Figure 3.11 The '1-2' integrated model for the distributed AGVS (Case 1)*
Table 3.4 The description of the integrated model in Figure 3.11

<table>
<thead>
<tr>
<th>Places</th>
<th>Description of places</th>
<th>Transitions</th>
<th>Description of transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p_1 )</td>
<td>Output buffer occupied slots (( M_i ))</td>
<td>( t_1 )</td>
<td>Part entering output buffer</td>
</tr>
<tr>
<td>( p_2 )</td>
<td>Output buffer empty slots (( M_i ))</td>
<td>( t_2 )</td>
<td>Start of transportation (( M_i ) to ( M_j ))</td>
</tr>
<tr>
<td>( p_3 )</td>
<td>AGV available (belongs to ( M_i ))</td>
<td>( t_3 )</td>
<td>Start of transportation (( M_i ) to ( M_k ))</td>
</tr>
<tr>
<td>( p_4 )</td>
<td>AGV transporting (( M_i ) to ( M_j ))</td>
<td>( t_4 )</td>
<td>End of return (( M_i ) to ( M_i ))</td>
</tr>
<tr>
<td>( p_5 )</td>
<td>AGV transporting (( M_i ) to ( M_k ))</td>
<td>( t_5 )</td>
<td>End of return (( M_k ) to ( M_i ))</td>
</tr>
<tr>
<td>( p_6 )</td>
<td>AGV returning (( M_i ) to ( M_i ))</td>
<td>( t_6 )</td>
<td>End of transportation (( M_i ) to ( M_i ))</td>
</tr>
<tr>
<td>( p_7 )</td>
<td>AGV returning (( M_k ) to ( M_i ))</td>
<td>( t_7 )</td>
<td>End of transportation (( M_i ) to ( M_k ))</td>
</tr>
</tbody>
</table>

As for the transportation module, it is apparent that two modules are used to indicate the transportation operations from \( M_i \) to \( M_j \) and from \( M_i \) to \( M_k \). However, the two modules share a common place, which is \( p_3 \). In Table 3.4, place \( p_3 \) is defined as the place representing the availability of the AGV that belongs to machine \( M_i \). Thus, place \( p_3 \) forms a mutual exclusion relationship between places \( p_4 \) and \( p_6 \). In other words, the integrated model in Fig. 3.11 guarantees that place \( p_4 \) and \( p_6 \) cannot be marked simultaneously. Thus, only one transportation operation, either from \( M_i \) to \( M_j \) or from \( M_i \) to \( M_k \), can be taking place at certain moment. This is due to the fact that there is only one AGV available at machine \( M_i \). After the AGV performs the required transportation operation and returns to machine \( M_i \), it restores the marking of place \( p_3 \). Thus, the AGV restores its original status and becomes available for performing the next transportation operation. Finally, it should be noted that transition \( t_8 \) indicates the events of ending the transportation operation from machine \( M_i \) to machine \( M_j \), the start of the returning operation, and the entrance of a part into the input buffer of machine \( M_i \). The same representation is applied with transition \( t_7 \) but with respect to machine \( M_k \).
The second example of routing flexibility to be considered occurs when a process of a certain part can be performed on one of two machines, and is then followed by a process that can be performed on only one machine. For example, process 1 can be performed on machine $M_i$ or machine $M_j$, whereas process 2 is performed on machine $M_k$. The modified integrated model (i.e. the combination of the machining and transportation modules) that represents such a situation is given in Fig. 3.12. The description of the corresponding places and transitions is given in Table 3.5. The notation ‘2-1’ is used to indicate that the first process can be performed on two different machines while the second process is performed on a single machine.

Two separate transportation modules are used in Fig. 3.12 to indicate the transportation operations from machines $M_i$ and $M_j$ to machine $M_k$. No common places exist between the two modules since each machine has its own AGV.

Figure 3.12 The ‘2-1’ integrated model for the distributed AGVS (Case 1)
Table 3.5 The description of the integrated model in Figure 3.12

<table>
<thead>
<tr>
<th>Places</th>
<th>Description of places</th>
<th>Transitions</th>
<th>Description of transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>p₁</td>
<td>AGV available (belongs to M₁)</td>
<td>t₁</td>
<td>Start of transportation (M₁ to Mₖ)</td>
</tr>
<tr>
<td>p₂</td>
<td>AGV available (belongs to M₁)</td>
<td>t₂</td>
<td>Start of transportation (M₁ to Mₖ)</td>
</tr>
<tr>
<td>p₃</td>
<td>AGV transporting (Mᵢ to Mₖ)</td>
<td>t₃</td>
<td>End of return (Mₖ to Mᵢ)</td>
</tr>
<tr>
<td>p₄</td>
<td>AGV transporting (Mᵢ to Mₖ)</td>
<td>t₄</td>
<td>End of return (Mₖ to Mᵢ)</td>
</tr>
<tr>
<td>p₅</td>
<td>AGV returning (Mₖ to Mᵢ)</td>
<td>t₅</td>
<td>End of transportation (Mᵢ to Mₖ)</td>
</tr>
<tr>
<td>p₆</td>
<td>AGV returning (Mₖ to Mᵢ)</td>
<td>t₆</td>
<td>End of transportation (Mᵢ to Mₖ)</td>
</tr>
<tr>
<td>p₇</td>
<td>Output buffer occupied slots (Mₖ)</td>
<td>t₇</td>
<td>Start of machining operation (Mₖ)</td>
</tr>
<tr>
<td>p₈</td>
<td>Output buffer empty slots (Mₖ)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The machining modules of machines Mᵢ and Mⱼ are not illustrated in Fig. 3.12 due to the fact that no changes are imposed on their structure. However, the machining module of machine Mₖ is modified in order to account for the possibility of receiving a part from either machine Mᵢ or machine Mⱼ. In Fig. 3.12 one finds that two transitions are used instead of one, to represent the event of a part entering the input buffer of machine Mₖ. These two transitions are t₅ and t₆. Accordingly, the places representing the occupied and empty input buffer slots are connected to these two transitions as shown in Fig. 3.12.

The third and last example of routing flexibility to be considered in Case 1 of the distributed AGVS layout occurs when each of the two consecutive processes can be performed on two different machines. For example, process 1 can be performed on machine Mᵢ or machine Mⱼ, whereas process 2 can be performed on machine Mⱼ or machine Mₖ. The modified integrated model (i.e. the combination of the machining and transportation modules) that represents such a situation is given in Fig. 3.13. The description of the corresponding places and transitions is given in Table 3.6. The notation ‘2-2’ is used to indicate that the first process can be performed on two different
machines and similarly the second process can also be performed on two different machines.

This example can be viewed as a combination of the previous two examples (i.e. ‘1-2’ and ‘2-1’). The routing flexibility is apparent from the four different transportation operations that one can choose from. In addition, the shared resource feature is evident from the common places. Therefore, one finds that the AGVs in addition to the input and output buffer spaces are considered to be shared resources. Finally, one finds that these three examples can be easily expanded to include several machines. Thus, the models that have been proposed can be easily modified to account for ‘1-n’, ‘n-1’, and ‘n-n’ situations, where n is the number of machines and can assume any value greater than or equal to two.

Figure 3.13 The ‘2-2’ integrated model for the distributed AGVS (Case 1)
Table 3.6 The description of the integrated model in Figure 3.13

<table>
<thead>
<tr>
<th>Places</th>
<th>Description of places</th>
<th>Transitions</th>
<th>Description of transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$ ($p_3$)</td>
<td>Out. buff. occupied slots $M_h (M_i)$</td>
<td>$t_1$ ($t_2$)</td>
<td>End of machining oper. $M_h (M_i)$</td>
</tr>
<tr>
<td>$p_2$ ($p_4$)</td>
<td>Out. buff. empty slots $M_j (M_k)$</td>
<td>$t_3$ ($t_4$)</td>
<td>Start of trans. $M_h$ to $M_i$ ($M_i$ to $M_k$)</td>
</tr>
<tr>
<td>$p_5$ ($p_6$)</td>
<td>AGV available for $M_h$ ($M_i$)</td>
<td>$t_5$ ($t_6$)</td>
<td>Start of trans. $M_i$ to $M_k$ ($M_i$ to $M_j$)</td>
</tr>
<tr>
<td>$p_7$ ($p_9$)</td>
<td>AGV transport $M_h$ to $M_i$ ($M_k$ to $M_i$)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_9$ ($p_{10}$)</td>
<td>AGV transport $M_i$ to $M_k$ ($M_i$ to $M_i$)</td>
<td>$t_9$ ($t_{10}$)</td>
<td>End of return $M_k$ to $M_i$ ($M_i$ to $M_i$)</td>
</tr>
<tr>
<td>$p_{11}$ ($p_{12}$)</td>
<td>AGV returning $M_j$ to $M_h$ ($M_k$ to $M_h$)</td>
<td>$t_{11}$ ($t_{12}$)</td>
<td>End of trans. $M_h$ to $M_i$ ($M_i$ to $M_k$)</td>
</tr>
<tr>
<td>$p_{13}$ ($p_{14}$)</td>
<td>AGV returning $M_j$ to $M_i$ ($M_i$ to $M_i$)</td>
<td>$t_{13}$ ($t_{14}$)</td>
<td>End of trans. $M_i$ to $M_k$ ($M_i$ to $M_i$)</td>
</tr>
<tr>
<td>$p_{15}$ ($p_{16}$)</td>
<td>Inp. buff. occupied slots $M_k$ ($M_j$)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Case 2: Bi-directional one path layout

In this case, it is assumed that any two machines are connected to each other as shown in Fig. 3.14. This single bi-directional track exists between any two machines, if there is a possibility that a transportation operation could take place from one machine to the other.

![Figure 3.14 The bi-directional single path connection](image)

Figure 3.14 The bi-directional single path connection

Obviously, the flow in this particular case is constrained relative to the previous case where two tracks connected every machine couple. Thus, if a part needs to be
transported from machine $M_i$ to machine $M_j$, one has to ensure that there is no flow of parts in the opposite direction. This control measure has to be inserted in the transportation module, in order to ensure that no AGV collision occurs between two AGVs moving in opposite directions on the same path. Thus, the transportation module depicting this system is given below in Fig. 3.15. In addition a description of the corresponding places and transitions is given in Table 3.7.

![The transportation module for the distributed AGVS (Case 2)](image)

**Figure 3.15** The transportation module for the distributed AGVS (Case 2)

<table>
<thead>
<tr>
<th>Places</th>
<th>Description of places</th>
<th>Transitions</th>
<th>Description of transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>AGV available</td>
<td>$t_1$</td>
<td>Start of transportation operation</td>
</tr>
<tr>
<td>$p_2$</td>
<td>Path available</td>
<td>$t_2$</td>
<td>End of transportation operation</td>
</tr>
<tr>
<td>$p_3$</td>
<td>AGV transporting</td>
<td>$t_3$</td>
<td>End of return operation</td>
</tr>
<tr>
<td>$p_4$</td>
<td>AGV returning</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Evidently, the difference between the two transportation modules for the distributed AGVS is that an extra place was added in the second case. In Fig. 3.15, one finds that the added place is $p_2$, which is defined in Table 3.7 as the place representing
the availability of the path connecting the two machines. If this place is marked, then the path is available (i.e. vacant). Accordingly, if place $p_2$ is unmarked then this means that the path is not available, or in other words an AGV is moving along the path. Obviously, there is no time delay associated with this place, and therefore any token in this place is assumed to readily available for the enabling and firing rules.

The addition of place $p_2$ affects the firing of transition $t_1$. In order for transition $t_1$ to fire, both places $p_1$ and $p_2$ have to be marked. Thus the model ensures that in order for a transportation operation to take place both the required AGV and the required path have to be available. If only one of the preconditions is available, then the transportation operation will be delayed until the other becomes available. When the transition fires, the tokens in places $p_1$ and $p_2$ are removed. This indicates that the AGV and the path are being currently used.

The addition of the path availability place also affects the firing of transition $t_3$ which represents the end of the AGV returning operation. It is seen that when transition $t_3$ fires, the initial marking is restored, by the deposition of one token in places $p_1$ and $p_2$. Thereby, the model considers the path to be available only after the AGV returns to its original position. This control mechanism helps the model to ensure that no path can have more than one AGV moving along it at any given time instant.

The integration of this transportation module with the machining modules is performed using the same methods applied in Case 1. Therefore, Fig. 3.16 presents the integrated model which depicts the machining and transportation of a part from machine $M_i$ to machine $M_j$. The only difference between this model and the corresponding model developed in Case 1 is that the number of places has increased by one due to the addition of the place representing the path availability.
Figure 3.16 The integrated model for transporting a part with a distributed AGVS (Case 2)
The development of the modified integrated models that incorporate routing flexibility will be very similar to those discussed in Case 1. The only difference will be the presence of the additional places that represent the availability of the paths. Thus, the '1-2', '2-1', and '2-2' integrated models are given directly in Fig. 3.17, 3.18, and 3.19, respectively.

![Diagram of the '1-2' integrated model for the distributed AGVS (Case 2)](image)

*Figure 3.17 The '1-2' integrated model for the distributed AGVS (Case 2)*
Figure 3.18 The ‘2-1’ integrated model for the distributed AGVS (Case 2)

Figure 3.19 The ‘2-2’ integrated model for the distributed AGVS (Case 2)
The track availability places in Fig. 3.17 are places $p_s$ and $p_9$, whereas in Fig. 3.18 they are places $p_6$ and $p_{10}$. In the case of Fig. 3.19 there are four path availability places which are $p_{17}$, $p_{18}$, $p_{19}$, and $p_{20}$. It should be noted that all these places are not shared in the given modules due to the fact that each transportation operation is performed on a different path. These models can be easily extended to include any number of machines.

Finally, one needs to point out that the loading area is treated as a machine with one or more AGVs located at its site. Thus, the corresponding transportation modules are exactly the same as those developed in Case 1 and Case 2. However, the only difference is that the AGV availability place can carry more than one token. The number of these tokens indicates the number of AGVs located at the loading area.

### 3.4.2.2 The Centralized AGVS

In the centralized AGVS each AGV assumes responsibility for a certain part. The AGV remains with the part from the moment that it enters the FMS until it leaves the system in the form of a final product. In this particular AGVS all the AGVs are located at the loading (input) area. Whenever a part is released into the system, an AGV is assigned to it. The AGV acts as the pallet that moves around the system with the part. After the part reaches the final product status, the AGV transports it to the unloading area (exit). Thus, the AGV is available once more, and heads back to the loading area to repeat the same cycle. Therefore, the maximum number of parts that could be in the processing stage is equal to the number of AGVs available in the loading area. Thus the number of parts being machined in the FMS is limited by the number of AGVs.

The general layout of the system is given in Fig. 3.20. The direction of the arrows indicate the direction of the flow. The curved arrow within the block representing the machines is used to indicate that the AGV moves to several machines (according to
the process plan and schedule) in order to transform the part into a final product. In this layout it is assumed that the loading and unloading areas are adjacent to each other. This assumption will be used throughout the entire model development stage of the centralized AGVS.

![Diagram of FMS general layout](image)

**Figure 3.20 The FMS general layout (centralized AGVS case)**

Similar to the distributed AGVS, the shape and number of connections between the different machines has a direct impact on the structure of the Petri net transportation module that will be constructed. However, in the case of a centralized AGVS one is not restricted by the assumption that any two machines have to be directly connected by a path, if a transportation operation occurs between them. Although this research offers the possibility of modeling such a situation, yet an alternative layout is suggested for the user to choose from. Therefore, three different cases of AGV track layouts are considered in the centralized AGVS.

Case 1: Uni-directional two path layout

In this case, if a part needs to be transported between two specific machines, it is assumed that these two machines are connected to each other, as shown in Fig. 3.21. Each arrow in Fig. 3.21 indicates a separate AGV track. This particular layout
(connection) is called the uni-directional two path layout. The term uni-directional is used to indicate that an AGV can move in only one direction along a certain path.

![Diagram of the uni-directional two path connection]

**Figure 3.21 The uni-directional two path connection**

Assuming that a part needs to be transported from $M_i$ to $M_j$. The corresponding AGV located at machine $M_i$ will transport that part along the track heading from $M_i$ to $M_j$. If another part has to be transported from $M_j$ to $M_i$, then the transportation process can concurrently take place with the previous one. This is due to the fact that the AGV carrying the part located at machine $M_j$ can perform the transportation process along the other path that is pointing from $M_j$ to $M_i$. The transportation module depicting this particular system (i.e. AGVS + layout) is given in Fig. 3.22. A description of the places and transitions used in the module is provided in Table 3.8.

![Diagram of the transportation module for the centralized AGVS (Case 1)]

**Figure 3.22 The transportation module for the centralized AGVS (Case 1)**
Table 3.8 The description of the transportation module in Figure 3.22

<table>
<thead>
<tr>
<th>Places</th>
<th>Description of places</th>
<th>Transitions</th>
<th>Description of transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>p₁</td>
<td>Path available</td>
<td>t₁</td>
<td>Start of transportation operation</td>
</tr>
<tr>
<td>p₂</td>
<td>AGV transporting</td>
<td>t₂</td>
<td>End of transportation operation</td>
</tr>
</tbody>
</table>

Transition t₁ is defined in Table 3.8 as the transition representing the event of starting the transportation operation. In order for transition t₁ to fire (i.e. the occurrence of the event), its input place p₁ has to be marked. Place p₁ is defined as the place representing the availability of the path connecting two specific machines to each other. In other words, for a transportation operation to take place (in a certain direction) the required path has to be available. When transition t₁ is fired, one token is removed from place p₁ and equally one token is placed in p₂. Place p₂ is defined as the place representing the transportation operation. Thus on firing t₁ the module displays the state where a transportation operation is being conducted and accordingly the path being utilized is considered to be unavailable to prevent any other AGV from moving across it. This ensures that no path will have more than one AGV moving along it, and thereby the danger of having AGV collision is avoided.

Transition t₂ is defined as the transition representing the event of ending the transportation operation. Therefore this transition can only fire after the transportation operation is completed (i.e. after the expiring of the time delay associated with place p₂). The effect of firing transition t₂ is the removal of one token from place p₂ and the deposition of one token in place p₁. Thus after the transportation operation is performed the path that has been used is restored to its original status (i.e. available).

The merging of this transportation module with the machining module is very similar to the method used in the case of the distributed AGVS. One finds that both transitions t₁ and t₂ of the transportation module given in Fig. 3.22 are common between both the transportation and the machining modules. The reason for this is that
transition $t_1$ can be used to represent the two events of a part leaving the output buffer of a machine, and the beginning of the transportation operation. Similarly, transition $t_2$ can be used to represent the two events of the ending of a transportation operation, and the entrance of a part into the input buffer of a machine. Thus, the integrated (merged) model representing the transportation of a part from machine $M_i$ to machine $M_j$ using a centralized AGVS with the given track layout is given in Fig. 3.23.

However, the system modeled in Fig. 3.23 does not demonstrate any form of routing flexibility. This is due to the fact that for the first and second machining processes only one machine can be used. Obviously, in an FMS this case rarely happens due to the highly flexible nature of the manufacturing system. Therefore three cases are considered where either the first, second, or both of the machining processes can be conducted on two different machines. The ‘1-2’, ‘2-1’, and ‘2-2’ modified integrated models for Case 1 of the centralized AGVS are illustrated in Fig. 3.24, 3.25, 3.26, respectively. The interpretation of the places and transitions involved in these three Figures is given in Tables 3.9, 3.10, and 3.11, respectively.

It is obvious that the modifications introduced by the centralized AGVS transportation module are identical to those produced by the distributed AGVS module. Therefore one notices that the machining modules are modified in a similar manner with respect to the two transportation systems. Thus if after a part is machined on $M_i$ it can visit either machine $M_j$ or machine $M_k$, one finds that the output (exiting) transition of the machining module of $M_i$ is divided into two transitions. Conversely, if a part at machine $M_i$ can come from either machine $M_j$ or machine $M_k$, one finds that the input (entrance) transition of the machining module of $M_i$ is divided into two transitions. Depending on which AGVS is being used and the corresponding track layout, one inserts in between of the machining modules the appropriate transportation module.
Figure 3.23 The integrated model for transporting a part with a centralized AGVS (Case 1)
Figure 3.24 The ‘1-2’ integrated model for the centralized AGVS (Case 1)

Figure 3.25 The ‘2-1’ integrated model for the centralized AGVS (Case 1)
Figure 3.26 The ‘2-2’ integrated model for the centralized AGVS (Case 1)

Table 3.9 The description of the integrated model in Figure 3.24

<table>
<thead>
<tr>
<th>Places</th>
<th>Description of places</th>
<th>Transitions</th>
<th>Description of transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>p₁</td>
<td>Output buffer occupied slots (M₁)</td>
<td>t₁</td>
<td>Start of trans. oper. (M₁ to M₁)</td>
</tr>
<tr>
<td>p₂</td>
<td>Output buffer empty slots (M₁)</td>
<td>t₂</td>
<td>Start of trans. oper. (M₁ to Mₖ)</td>
</tr>
<tr>
<td>p₃</td>
<td>Path available (M₁ to M₁)</td>
<td>t₃</td>
<td>End of trans. oper. (M₁ to M₁)</td>
</tr>
<tr>
<td>p₄</td>
<td>AGV transporting (M₁ to M₁)</td>
<td>t₄</td>
<td>End of trans. oper. (M₁ to Mₖ)</td>
</tr>
<tr>
<td>p₅</td>
<td>Path available (M₁ to Mₖ)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p₆</td>
<td>AGV transporting (M₁ to Mₖ)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 3.10 The description of the integrated model in Figure 3.25

<table>
<thead>
<tr>
<th>Places</th>
<th>Description of places</th>
<th>Transitions</th>
<th>Description of transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p_1 )</td>
<td>Path available (( M_i ) to ( M_k ))</td>
<td>( t_1 )</td>
<td>Start of trans. oper. (( M_i ) to ( M_k ))</td>
</tr>
<tr>
<td>( p_2 )</td>
<td>AGV transporting (( M_i ) to ( M_k ))</td>
<td>( t_2 )</td>
<td>Start of trans. oper. (( M_i ) to ( M_k ))</td>
</tr>
<tr>
<td>( p_3 )</td>
<td>Path available (( M_i ) to ( M_k ))</td>
<td>( t_3 )</td>
<td>End of trans. oper. (( M_i ) to ( M_k ))</td>
</tr>
<tr>
<td>( p_4 )</td>
<td>AGV transporting (( M_i ) to ( M_k ))</td>
<td>( t_4 )</td>
<td>End of trans. oper. (( M_i ) to ( M_k ))</td>
</tr>
<tr>
<td>( p_5 )</td>
<td>Input buffer occupied slots (( M_k ))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( p_6 )</td>
<td>Input buffer empty slots (( M_k ))</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 3.11 The description of the integrated model in Figure 3.26

<table>
<thead>
<tr>
<th>Places</th>
<th>Description of places</th>
<th>Transitions</th>
<th>Description of transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p_1 ) (( p_2 ))</td>
<td>Output buff. occupied slots ( M_h (M_i) )</td>
<td>( t_1 ) (( t_2 ))</td>
<td>Start of trans. ( M_h ) to ( M_i ) (( M_h ) to ( M_k ))</td>
</tr>
<tr>
<td>( p_3 ) (( p_5 ))</td>
<td>Path available ( M_h ) to ( M_i ) (( M_h ) to ( M_k ))</td>
<td>( t_3 ) (( t_4 ))</td>
<td>Start of trans. ( M_i ) to ( M_k ) (( M_i ) to ( M_i ))</td>
</tr>
<tr>
<td>( p_4 ) (( p_6 ))</td>
<td>AGV transport ( M_h ) to ( M_i ) (( M_h ) to ( M_k ))</td>
<td>( t_5 ) (( t_6 ))</td>
<td>End of trans. ( M_h ) to ( M_i ) (( M_h ) to ( M_k ))</td>
</tr>
<tr>
<td>( p_7 ) (( p_9 ))</td>
<td>Path available ( M_i ) to ( M_k ) (( M_i ) to ( M_i ))</td>
<td>( t_7 ) (( t_8 ))</td>
<td>End of trans. ( M_i ) to ( M_k ) (( M_i ) to ( M_i ))</td>
</tr>
<tr>
<td>( p_9 ) (( p_{10} ))</td>
<td>AGV transport ( M_i ) to ( M_k ) (( M_i ) to ( M_k ))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( p_{11} ) (( p_{12} ))</td>
<td>Input buff. occupied slots ( M_k (M_i) )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The above modeling strategy and the integrated models that have been presented can be very easily extended to account for any combination of machines. Thus the Petri net model manages to fully capture the routing flexibility feature.

**Case 2 : Bi-directional one path layout**

The bi-directional single path layout which was considered as Case 2 in the distributed AGVS, can also be applied to the centralized AGVS. The proposed track
layout and connection between the different machines was given in Fig. 3.14. Similar to the previous case the AGV carrying the part has to check if the required path is available before starting the transportation process. However, in this particular situation both the AGVs at machines \( M_1 \) and \( M_i \) need to perform this inquiry, since there is only one path connecting the two machines to each other.

However, although additional constraints have been imposed on the flow of the AGVs, yet the same transportation module developed for Case 1 is used to model this problem. The transportation module is redrawn in Fig. 3.27 for simplicity.

![Figure 3.27 The transportation module for the centralized AGVS (Case 2)](image)

Evidently, there is no difference in the topology and structure of the transportation module used for Case 2 of the centralized AGVS from the one used in Case 1. However, the difference lies in the interpretation and the physical significance of place \( p_1 \). The definition of the place \( p_1 \) pictured in Fig. 3.27 is that it is the place representing the path availability. However, in this case there exists only one path between any two machines. In Case 1 two places were required for any machine-couple to define the two tracks connecting them to each other. However, in Case 2 only one place is required since only one path is available. Thus, an AGV moving from machine \( M_i \) to machine \( M_j \), and similarly an AGV requesting to travel in the opposite direction, has to check first if place \( p_1 \) is marked (i.e. the path is available). Obviously, in Case 2 a fewer number of places need to be created to represent the paths of the FMS. However, the sharing of these places will be increased.
All the integrated and modified models of Case 2 have exactly the same structure as those provided in Case 1. The only difference lies in the definition of the places representing the path availability.

Case 3: The cyclic layout

The last type of layout considered for the centralized AGVS is the cyclic layout. Fig. 3.28 depicts the direction of flow of the AGVs along with the arrangement of the FMS components.

![Diagram of cyclic layout](image)

**Figure 3.28 The cyclic layout**

\( M \) is assumed to be the number of machines in the FMS. Therefore machine \( M \) is the last machine on the cyclic layout. As indicated by the arrow directions, the AGV is only allowed to move in one direction along the cycle. Thus an AGV carrying the same part can rotate around this cycle more than once, depending on the provided schedule.

In this case one cannot design a generic transportation module that is applicable in all the possible situations. The problem is tackled using a different strategy. Firstly one creates a place representing the availability of each path in the FMS. The number of places will be equal to \( M+1 \) due to the fact that the loading and unloading areas are assumed to be at the same location. Accordingly, \( M+1 \) places will also be developed to
represent the transportation operation across each corresponding path. Each one of these places will have a certain time delay associated with it.

Assuming that a part needs to travel from machine 2 to machine 3. The transportation module that represents this activity is given in Fig. 3.29. The definition of the corresponding places and transitions is given in Table 3.12.

![Diagram](image)

Figure 3.29 The transportation module from $M_2$ to $M_3$ (Cyclic Case)

<table>
<thead>
<tr>
<th>Places</th>
<th>Description of places</th>
<th>Transitions</th>
<th>Description of transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>Path available ($M_2$ to $M_3$)</td>
<td>$t_1$</td>
<td>Start of transportation operation</td>
</tr>
<tr>
<td>$p_2$</td>
<td>AGV transport ($M_2$ to $M_3$)</td>
<td>$t_2$</td>
<td>End of transportation operation</td>
</tr>
</tbody>
</table>

The transportation module drawn in Fig. 3.29 is identical in its structure to the modules used in cases 1 and 2. The reason for this similarity is that the transportation operation is between two adjacent machines. However, in the case of modeling the transportation of a part from machine 1 to 3, one finds that the transportation module becomes as shown in Fig. 3.30.
Figure 3.30 The transportation module from M₁ to M₃ (Cyclic Case)

Table 3.13 The description of the transportation module in Figure 3.30

<table>
<thead>
<tr>
<th>Places</th>
<th>Description of places</th>
<th>Transitions</th>
<th>Description of transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>p₁</td>
<td>Path available (M₁ to M₂)</td>
<td>t₁</td>
<td>Start of trans. oper. (M₁ to M₂)</td>
</tr>
<tr>
<td>p₂</td>
<td>AGV transport (M₁ to M₂)</td>
<td>t₂</td>
<td>Start of trans. oper. (M₂ to M₃)</td>
</tr>
<tr>
<td>p₃</td>
<td>Path available (M₂ to M₃)</td>
<td>t₃</td>
<td>End of trans. oper. (M₂ to M₃)</td>
</tr>
<tr>
<td>p₄</td>
<td>AGV transport (M₂ to M₃)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Since machine 1 and machine 3 are not adjacent nor directly connected the transportation operation is divided into several sequential activities. In this case the activities are firstly moving from machine 1 to machine 2 and then from machine 2 to machine 3. Each activity is then represented by the transportation module given in Fig. 3.29. Finally the modules are merged by sharing their common transitions to form the model representing the transportation operation. For example in Fig. 3.30 transition t₂
represents the event of ending the transportation from machine 1 to machine 2 and the starting of the transportation from machine 2 to machine 3.

Thus the transportation module is dependent on the relative location of the machines under consideration. However, in general the model is formed as a combination of several identical modules, that are merged together.

3.4.2.3 The Free Floating AGVS

In the free floating AGVS all the AGVs are assumed to move around the FMS freely without any predetermined constraints (i.e. full freedom). Thus the AGVs are not restrained or forcefully assigned to a particular machine or a certain part. In this case only one type of track layout is proposed, which is the cyclic layout. Therefore the layout and direction of flow in the FMS is identical to the one illustrated in Fig. 3.28.

The modules used to model the transportation operations in the free floating AGVS are mainly based on the work of Sun, Cheng, and Fu (1994). However, some modifications have been introduced on their modules in order to enable the application of the search algorithm. In addition some other modifications have been introduced in order to maintain the logic of the modeled system, and in order to prevent the model from allowing certain activities to occur. Therefore, the first modification was to remove the self-loops produced in their model in order for the Petri net to be pure and correctly represented by the A-matrix which is generated in the modeling stage and used by the scheduling search algorithm. In addition some additional control mechanisms have been introduced to ensure that an AGV carrying a certain part cannot be requested to transport another part. This guarantees that no AGV will carry more than one part at a time and that each AGV will immediately deliver the part that it carries to the required destination.
The transportation of a part in the free floating AGVS is divided into three main stages. The first stage is concerned with a part taking control of a specific AGV. Obviously, if no AGVs are available (i.e. free) the part will have to wait. In addition this control mechanism ensures that each AGV can only be involved in one transportation task at a time. The second stage is calling on the AGV to come to the location (or stop) where the part is residing. Therefore, the Petri net model has to determine the relative location of the AGV with respect to the part, and then a method has to be devised for moving (i.e. directing) the AGV to the part’s current location. After the AGV reaches the part and loads it, the third and final stage is to transport the part to the desired destination. When the AGV delivers and unloads the part at the required destination place, it is assumed that the AGV remains at that location (i.e. free and available) until it is summoned for a new transportation task.

In order to model this AGVS two interactive modules have to be developed. These two modules were named in the work of Sun, Cheng and Fu (1994) as the transportation layout module and movement control module. In this research both modules will be modified although the structure of the Transportation Module remains intact. In addition the integration of these modules with the machining module is performed differently in this research.

The transportation layout module is used to describe the layout of the specific FMS under consideration. This module indicates the different locations (stops) in the FMS and shows how these different locations are connected to each other. In this research, the transportation layout module will only have to describe the cyclic layout since it is the layout associated with the free floating AGVS. The transportation module is formed as a combination (or series) of several transportation layout units, where each unit models the flow between two directly connected locations. Therefore, in the case of the cyclic layout these units represent the flow between two adjacent locations. A generic transportation layout unit is given in Fig. 3.31, and the description of the corresponding places and transitions is given in Table 3.14.
In the given transportation layout unit the two adjacent locations belong to machines $M_i$ and $M_j$. The transportation device is called $AGV_x$. If $AGV_x$ is currently residing at machine $M_i$ then the place $p_2$ should be marked, where place $p_2$ is defined as the place indicating the presence of $AGV_x$ at $M_i$.

Transition $t_1$ is defined as the transition representing the start of the movement of $AGV_x$ from machine $M_i$ to machine $M_j$. Obviously, one of the preconditions for this
transition to fire is that the AGV has to be located at machine $M_i$ (i.e. $p_2$ is marked). The other preconditions are depicted by the transition's input places, which are places $p_1$ and $p_7$. Place $p_1$ is defined as the place representing the arrival of an order to move the corresponding AGV from machine $M_i$ to machine $M_j$. On the other hand place $p_7$ depicts the availability of the stop (i.e. location) at machine $M_i$. Thus, the model states that in order for an AGV located at machine $M_i$ to move to the adjacent machine $M_j$, there has to be an order for this motion and the stop at $M_j$ has to be free.

The firing of $t_1$ results in the removal of one token from places $p_1$, $p_2$, and $p_7$. In addition one token is added to places $p_3$ and $p_4$, where the first is defined as the place representing the availability of the stop at $M_i$ and the second is the place depicting the movement activity. Thus, the model confirms the fact that when AGV starts moving to $M_j$ ($t_1$ fired) the stop of $M_i$ is no longer occupied. In addition the removal of the token from $p_7$ indicates that the stop at $M_j$ is reserved (in this case for AGV), and accordingly no AGV is allowed to move to, or assume control of that stop.

Transition $t_2$ is defined as the transition representing the end of the movement of AGV from machine $M_i$ to machine $M_j$. Thus this transition can only fire after the completion of the movement activity. On firing $t_2$ one token is removed from place $p_4$, and one token is deposited in places $p_5$ and $p_6$. Place $p_5$ is defined as the place representing the signal indicating that the ordered movement has been accomplished. On the other hand place $p_6$ represents the presence of AGV at the stop of machine $M_j$.

Considering the interaction of the transportation layout unit with its environment (i.e. other modules), one finds that places $p_1$ and $p_5$ are used to serve this purpose. Place $p_1$ is used to model an order which the unit receives from its environment, whereas place $p_5$ models the answer (response) of the unit confirming that it has carried out the order.
Obviously, all the places of the transportation unit are immediate places except for place \( p_4 \), which has the time required for moving from machine \( M_i \) to \( M_j \) associated with it. The construction of the complete transportation layout module comes as a result of assembling several transportation layout units with each other. The transportation layout module for an FMS with a loading/unloading area and three machines is given in Fig. 3.32\(^1\).

\[ \text{Figure 3.32 The transportation layout module} \]

\(^1\) Place \( p_4 \) is drawn in four different locations. This repetition helps in clarifying the Figure and reducing the arc intersections.
It should be noted that the given transportation layout module is concerned with one specific AGV which is called AGV\textsubscript{x}. This point is clarified from the definition of the corresponding places and transitions that is given in Table 3.15.

**Table 3.15 The description of the transportation layout module in Figure 3.32**

<table>
<thead>
<tr>
<th>Places</th>
<th>Description of places</th>
<th>Trans.</th>
<th>Description of transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>p\textsubscript{1}</td>
<td>AGV\textsubscript{x} occupying M1 stop</td>
<td>t\textsubscript{1}</td>
<td>Start of moving from M1 to M2</td>
</tr>
<tr>
<td>p\textsubscript{2}</td>
<td>Order to move AGV\textsubscript{x} from M1 to M2</td>
<td>t\textsubscript{2}</td>
<td>End of moving from M1 to M2</td>
</tr>
<tr>
<td>p\textsubscript{3}</td>
<td>AGV\textsubscript{x} moving from M1 to M2</td>
<td>t\textsubscript{3}</td>
<td>Start of moving from M2 to M3</td>
</tr>
<tr>
<td>p\textsubscript{4}</td>
<td>Move order fulfilled</td>
<td>t\textsubscript{4}</td>
<td>End of moving from M2 to M3</td>
</tr>
<tr>
<td>p\textsubscript{5}</td>
<td>AGV\textsubscript{x} occupying M2 stop</td>
<td>t\textsubscript{5}</td>
<td>Start of moving from M3 to L/U</td>
</tr>
<tr>
<td>p\textsubscript{6}</td>
<td>Stop M2 available</td>
<td>t\textsubscript{6}</td>
<td>End of moving from M3 to L/U</td>
</tr>
<tr>
<td>p\textsubscript{7}</td>
<td>Order to move AGV\textsubscript{x} from M2 to M3</td>
<td>t\textsubscript{7}</td>
<td>Start of moving from L/U to M1</td>
</tr>
<tr>
<td>p\textsubscript{8}</td>
<td>AGV\textsubscript{x} moving from M2 to M3</td>
<td>t\textsubscript{8}</td>
<td>End of moving from L/U to M1</td>
</tr>
<tr>
<td>p\textsubscript{9}</td>
<td>AGV\textsubscript{x} occupying M3 stop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p\textsubscript{10}</td>
<td>Stop M3 available</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p\textsubscript{11}</td>
<td>Order to move AGV\textsubscript{x} from M3 to L/U</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p\textsubscript{12}</td>
<td>AGV\textsubscript{x} moving from M3 to L/U</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p\textsubscript{13}</td>
<td>AGV\textsubscript{x} occupying L/U stop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p\textsubscript{14}</td>
<td>Stop L/U available</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p\textsubscript{15}</td>
<td>Order to move AGV\textsubscript{x} from L/U to M1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p\textsubscript{16}</td>
<td>AGV\textsubscript{x} moving from L/U to M1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p\textsubscript{17}</td>
<td>Stop M1 available</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
It is apparent from Fig. 3.32 and the corresponding definitions given in Table 3.15 that a transportation layout module is composed of several transportation layout units linked to each other in series. In addition the last unit is linked to the first one in order to represent the cyclic (closed-loop) layout of the FMS paths. However, in the case that there are more than one AGV in the FMS, the number of transportation layout modules that need to be constructed is equal to the number of AGVs present in the system. All the places will be repeated for each AGV transportation module, except for the places representing the availability of the locations or stops. Thus, in the FMS example being considered, places \( p_6, p_{10}, p_{14}, \) and \( p_{17} \) will be common for all the different transportation modules. Through the sharing of these places the model ensures that no stop will have more than one AGV occupying it at a time. However, in order for the L/U area to have the capacity of holding more than one AGV, the number of tokens located in its corresponding place (in this case \( p_{14} \)) is initially set to be equal to the number of AGVs in the system. This is due to the fact that one assumes that in the initial state all the AGVs are located at the L/U area (i.e. input/output area).

Obviously the interaction between the transportation layout module and its environment is very similar to that of the transportation units. In the case being considered, each AGV can receive any one of four different orders \( (p_2, p_7, p_{11}, \) and \( p_{15}) \), and after fulfilling that order it responds with a message confirming the completion of the order \( (p_4) \). This interaction of the transportation layout module takes place with the other module involved in describing the transportation system. Thus, the transportation layout module does not interact directly with the machining module.

The other module describing the AGV transportation system, is called the movement control module. This module helps in transforming the requests for AGV movement issued by the machining module into orders that are sent to the transportation layout module. In addition this module helps in determining the current location of an AGV, and in directing the AGV along the appropriate route that leads it to the required destination stop. For the case being considered, the movement control
module concerned with the movement of AGV$_x$ to the location of machine M1 is depicted in Fig. 3.33. The definition of the corresponding places and transitions is given in Table 3.16.

Figure 3.33 The movement control module
Table 3.16 The description of the movement control module in Figure 3.33

<table>
<thead>
<tr>
<th>Places</th>
<th>Description of places</th>
<th>Trans.</th>
<th>Description of transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>AGV$_x$ request to move to M1</td>
<td>$t_1$</td>
<td>Dummy transition</td>
</tr>
<tr>
<td>$p_2$</td>
<td>AGV$_x$ occupying M1 stop</td>
<td>$t_2$</td>
<td>Dummy transition</td>
</tr>
<tr>
<td>$p_3$</td>
<td>AGV$_x$ occupying L/U stop</td>
<td>$t_3$</td>
<td>Dummy transition</td>
</tr>
<tr>
<td>$p_4$</td>
<td>AGV$_x$ occupying M2 stop</td>
<td>$t_4$</td>
<td>Dummy transition</td>
</tr>
<tr>
<td>$p_5$</td>
<td>AGV$_x$ occupying M3 stop</td>
<td>$t_5$</td>
<td>Move request fulfilled</td>
</tr>
<tr>
<td>$p_6$</td>
<td>Dummy place</td>
<td>$t_6$</td>
<td>Initiate order to move L/U to M1</td>
</tr>
<tr>
<td>$p_7$</td>
<td>Dummy place</td>
<td>$t_7$</td>
<td>Initiate order to move M2 to M3</td>
</tr>
<tr>
<td>$p_8$</td>
<td>Dummy place</td>
<td>$t_8$</td>
<td>Initiate order to move M3 to L/U</td>
</tr>
<tr>
<td>$p_9$</td>
<td>Dummy place</td>
<td>$t_9$</td>
<td>Moving order fulfilled</td>
</tr>
<tr>
<td>$p_{10}$</td>
<td>AGV$_x$ at M1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_{11}$</td>
<td>Order to move AGV$_x$ from L/U to M1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_{12}$</td>
<td>Order to move AGV$_x$ from M2 to M3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_{13}$</td>
<td>Order to move AGV$_x$ from M3 to L/U</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_{14}$</td>
<td>Move completed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_{15}$</td>
<td>Intermediate waiting place</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Firstly, it should be noted that several places in the movement control module are actually the same places used in the transportation layout module. Thus one finds that the places indicating the location of AGV$_x$ are the same (i.e. places $p_2$, $p_3$, $p_4$, and $p_5$ in the movement control module and $p_1$, $p_5$, $p_9$, and $p_{13}$ in the transportation layout module, respectively). In addition, obviously the places representing the interface between the two modules are identical. Thus the places representing the orders for movement and the place representing the confirmation of completing the order are identical in both models.
As for the dummy places and transitions, they have been added to remove the self-loops that were present in the original module introduced by Sun, Cheng, and Fu (1994). The reason for having to remove the self-loops is to enable the A-matrix from correctly representing the Petri net structure. Otherwise, the A-matrix would have given a false representation and the proposed scheduling algorithm would have been unable to reach a final solution.

In order to understand the logic behind the proposed movement control module in Fig. 3.33, the following scenario should be studied. Assume that in the current state, AGV_x is at the stop of machine M2 (i.e. place p_4 is marked), and a request for moving the AGV to the stop of machine M1 has been received (i.e. place p_1 is marked). Thus transition t_7 is the only enabled transition in the movement control module. The firing of transition t_7 results in the removal of one token from places p_1 and p_4. In addition it causes the deposition of one token in places p_8, p_12, and p_15. In other words, upon receiving the request the movement control module checks where the AGV is residing, and since in our case it was at machine M2, the module issued an order to the transportation layout module stating that AGV_x should move from machine M2 to machine M3 (place p_12). However, since the firing of transition t_7 removed the token in place p_4 which represents the presence of AGV_x at machine M2, the order for movement cannot be performed in the transportation layout module. Thus, one resorted to the addition of the dummy place p_8 and the dummy transition t_3 to restore the token in place p_4 and enable the transportation module to perform the movement order. Place p_15 is defined as an intermediate waiting place. Actually, the sole purpose of this place is to represent the fact that the movement control module is waiting for the response of the transportation layout module. Obviously, this response is that the movement order has been performed. Accordingly, after the movement operation is performed the transportation module will deposit a token in both places p_14 and p_8. In other words the result of performing this operation is that AGV_x will be in the stop of machine M3 (i.e. p_5 marked in the movement control module), and that a signal confirming the completion of the operation will be sent to the movement control module (i.e. p_14 will be marked).
Thus transition $t_6$ will be enabled. The firing of transition $t_6$ will result in the removal of one token from places $p_{14}$ and $p_{15}$, and the deposition of one token in place $p_1$. In other words, the movement control module has updated its marking and representation of the system. The current marking is very similar to the initial one with the difference that $AGV_x$ is currently residing at the stop of machine $M3$. Thus the movement control module has managed to transfer the concerned AGV from $M2$ to $M3$. The same sequence of events is repeated until the AGV reaches the required destination which is machine $M1$. In other words the movement control module has moved the AGV along each path segment until it reached the required destination.

When $AGV_x$ reaches the stop of machine $M1$, one will find that places $p_1$ and $p_2$ are marked. Thus transition $t_5$ is enabled. The firing of transition $t_5$ removes one token from places $p_1$ and $p_2$, and delivers one token to places $p_6$ and $p_{10}$. The role of the dummy place $p_6$ is similar to the other dummy places and it is to restore the token in place $p_2$ to indicate that $AGV_x$ is at the stop of machine $M1$. Place $p_{10}$ is defined as the place representing the signal that indicates that $AGV_x$ has arrived at $M1$. In other words it is a confirmation that the move request made in $p_1$ has been fulfilled.

Thus, the movement control module receives the request from the machining module for moving a certain AGV to a particular location. It performs this request in a sequence of steps that are marked by its interaction with the transportation module through the issuing of movement orders and the receiving of confirmations. After the AGV reaches the required destination the movement control module sends a signal back to the machining module indicating that its request has been fulfilled.

Accordingly the movement control module needs to be constructed for each stop present in the FMS (i.e. number of machine plus one). In addition each module has to be specified for a certain AGV. Thus the total number of movement control modules is equal to the product of the number of stops and the number of AGVs in the system.
Finally an illustrative example indicating the interaction between the machining module and the movement control module is given in Fig. 3.34. The description of the corresponding places and transitions is given in Table 3.17. Figure 3.34 models the case where a part is at machine $M_i$ and can be transported to either machine $M_j$ or machine $M_k$ to undergo its next machining process. The FMS is assumed to have two free floating AGVs, namely, AGV$x$ and AGV$y$.

Figure 3.34 The interaction between the machining and the movement control modules

---

2 Place $p_0$ is drawn in three different locations. This repetition helps in clarifying the Figure and reducing the arc intersections.
Table 3.17 The description of the model given in Figure 3.34

<table>
<thead>
<tr>
<th>Places</th>
<th>Description of places</th>
<th>Trans.</th>
<th>Description of transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>Output buffer occupied slots of $M_i$</td>
<td>$t_1$</td>
<td>Start of AGV$_x$ request for $M_i$</td>
</tr>
<tr>
<td>$p_2$</td>
<td>Output buffer empty slots of $M_i$</td>
<td>$t_2$</td>
<td>Start of AGV$_y$ request for $M_i$</td>
</tr>
<tr>
<td>$p_3$</td>
<td>AGV$_x$ available</td>
<td>$t_3$</td>
<td>Start of trans. from $M_i$ to $M_j$ (AGV$_x$)</td>
</tr>
<tr>
<td>$p_4$</td>
<td>AGV$_x$ request to move to $M_i$</td>
<td>$t_4$</td>
<td>Start of trans. from $M_i$ to $M_k$ (AGV$_x$)</td>
</tr>
<tr>
<td>$p_5$</td>
<td>Intermediate waiting place</td>
<td>$t_5$</td>
<td>Start of trans. from $M_i$ to $M_k$ (AGV$_y$)</td>
</tr>
<tr>
<td>$p_6$</td>
<td>AGV$_x$ at $M_i$</td>
<td>$t_6$</td>
<td>Start of trans. from $M_i$ to $M_j$ (AGV$_y$)</td>
</tr>
<tr>
<td>$p_7$</td>
<td>AGV$_x$ request to move to $M_i$</td>
<td>$t_7$</td>
<td>End of trans. from $M_i$ to $M_j$ (AGV$_x$)</td>
</tr>
<tr>
<td>$p_8$</td>
<td>AGV$_x$ request to move to $M_k$</td>
<td>$t_8$</td>
<td>End of trans. from $M_i$ to $M_k$ (AGV$_x$)</td>
</tr>
<tr>
<td>$p_9$</td>
<td>Intermediate waiting place</td>
<td>$t_9$</td>
<td>End of trans. from $M_i$ to $M_k$ (AGV$_y$)</td>
</tr>
<tr>
<td>$p_{10}$</td>
<td>Intermediate waiting place</td>
<td>$t_{10}$</td>
<td>End of trans. from $M_i$ to $M_j$ (AGV$_y$)</td>
</tr>
<tr>
<td>$p_{11}$</td>
<td>AGV$_x$ at $M_j$ (AGV$_x$ at $M_k$)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_{12}$</td>
<td>AGV$_y$ available</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_{13}$</td>
<td>AGV$_y$ request to move to $M_i$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_{14}$</td>
<td>Intermediate waiting place</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_{15}$</td>
<td>AGV$_y$ at $M_i$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_{16}$</td>
<td>AGV$_y$ request to move to $M_k$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_{17}$</td>
<td>AGV$_y$ request to move to $M_i$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_{18}$</td>
<td>Intermediate waiting place</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_{19}$</td>
<td>Intermediate waiting place</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_{20}$</td>
<td>Intermediate waiting place</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_{21}$</td>
<td>AGV$_y$ at $M_k$ (AGV$_y$ at $M_j$)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_{22}$</td>
<td>Input buffer occupied slots of $M_k$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_{23}$</td>
<td>Input buffer occupied slots of $M_i$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

108
In Fig. 3.34 two transitions, \( t_1 \) and \( t_2 \), are used to represent the exit of the output buffer of machine \( M_i \). The reason for using two transitions is to enable the machine to call on either or both the AGVs that are available (i.e. AGV\(_x\) and AGV\(_y\)). Place \( p_3 \) and \( p_{13} \) are defined as the places representing the availability of AGV\(_x\) and AGV\(_y\), respectively. These two places ensure that an AGV can only be involved in one transportation operation at a time. One will consider the case of AGV\(_x\) only since the other is identical. Obviously the whole transportation process is initiated when a part enters the output buffer of machine \( M_i \).

The machining module (i.e. \( M_i \)) sends a request for bringing AGV\(_x\) to machine \( M_i \). This is indicated by place \( p_4 \). The request is sent to the movement control module and the machining module waits (place \( p_5 \)) until the movement control module replies saying that AGV\(_x\) has arrived at the machine stop (place \( p_6 \)). The part is loaded to the AGV and then it can start to be transported to either machine \( M_i \) or machine \( M_k \) (transition \( t_3 \) or \( t_4 \)). Thus the firing of either transitions marks the removal of a part from the output buffer of machine \( M_i \), which is modeled by the arc connecting these transitions to place \( p_2 \). Place \( p_2 \) is defined as the place representing the empty slots in the output buffer. Thus the firing of either transition \( t_3 \) or \( t_4 \) adds a token to place \( p_2 \), or in other words increases the number of empty spaces in the output buffer by one.

If transition \( t_3 \) fires, the machining module sends a request to the movement control module to move AGV\(_x\) to machine \( M_j \) (place \( p_7 \)). The machining module waits (place \( p_9 \)) until the movement control module replies saying that AGV\(_x\) has reached the stop of machine \( M_j \) (place \( p_{11} \)). Thus the transportation operation is completed and the part is unloaded into the input buffer of the corresponding machine (transition \( t_7 \)). The firing of \( t_7 \) results in the deposition of one token in places \( p_3 \) and \( p_{24} \). The deposition of a token in place \( p_3 \) represents the fact that AGV\(_x\) is available once more. In other words, it is free to perform any other transportation operation. The deposition of a token in place \( p_{24} \) represents the fact that the number of parts in the input buffer of machine \( M_i \) has increased by one. It should be noted that the places representing the empty
slots in the input buffers of machines $M_i$ and $M_k$ have not been included for the sake of simplifying the model and reducing the number of illustrated places. Obviously, the transportation of $AGV_x$ to machine $M_k$ follows the same strategy that has been discussed.

3.4.2.4 The Robot

The robot is a different transportation device that is also considered in this proposed modeling technique. In contrast to AGVs the robot is limited in the sense that it can only serve a specific set of machines. It is assumed in this research that the machines served by a robot have to be in the same machining cell. The layout of a machining cell that is composed of three machines is given in Fig. 3.35.

![Figure 3.35 The layout of a machining cell](image)

Only one robot can be assigned to each machining cell. The task of the robot is to transport parts between the different machines that are located within the same corresponding machining cell. Thus if a transportation process needs to take place between two machines in the same machining cell, the robot (not an AGV) is used. It is assumed that each machine has its own input and output buffer area. Thus the robot takes a part from the output buffer of a specific machine and delivers it to the input buffer of another machine lying in the same machining cell.
The Petri net module depicting the use of a robot in a transportation process is depicted in Fig. 3.36. The definition of the corresponding places and transitions is given in Table 3.18.

![Petri net diagram](image)

**Figure 3.36 The robot transportation module**

<table>
<thead>
<tr>
<th>Places</th>
<th>Description of places</th>
<th>Transitions</th>
<th>Description of transitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>p₁</td>
<td>Robot available</td>
<td>t₁</td>
<td>Start of transportation operation</td>
</tr>
<tr>
<td>p₂</td>
<td>Robot transporting</td>
<td>t₂</td>
<td>End of transportation operation</td>
</tr>
<tr>
<td>p₃</td>
<td>Robot returning</td>
<td>t₃</td>
<td>End of return operation</td>
</tr>
</tbody>
</table>

Table 3.18 The description of the robot transportation module

It is obvious that the structure of the robot transportation module is identical to the structure of the module that was used to model Case 1 of the distributed AGVS (Fig. 3.9). Therefore the integration of both modules with the machining module is identical.

Assuming that a transportation operation which utilizes a robot is to be conducted. Transition t₁ is defined as the transition representing the beginning of this
transportation operation. However, for this operation to begin (i.e. \( t_1 \) fires) the robot has to be free (i.e. place \( p_1 \) is marked). The firing of \( t_1 \) results in the removal of one token from \( p_1 \) and the deposition of one token in \( p_2 \), which is the place representing the transportation operation. After the time delay required to conduct the transportation operation passes, the token in place \( p_2 \) is considered to be available. In other words transition \( t_2 \) is enabled.

The firing of \( t_2 \) indicates that the transportation operation is completed. This results in the removal of the token in place \( p_2 \) and the deposition of a token in place \( p_3 \), which is defined as the place representing the robot returning operation. The robot is assumed to have an initial position at which it resides whenever it is free. In addition, it is assumed that the robot is centralized with respect to all the machines in the cell, which means that the time needed for the robot to return to its original location is a constant, regardless of the specific transportation operation.

After the robot returns to its initial position (i.e. after time delay), transition \( t_3 \) is enabled. This transition is defined as the transition representing the event of ending the return operation. The firing of \( t_3 \) results in the removal of the token in place \( p_3 \) and the deposition of one token in place \( p_1 \). In other words the module indicates that the robot is available once more and ready to perform another transportation operation.

### 3.4.3 The Remaining Modules

The machining and the transportation modules are the main modules out of which the Petri net model is constructed. However, there remains some other components that are required in order to form the whole Petri net model. Thus, one has to mention the initial and final places, which indicate the presence of the raw materials and the final products, respectively. In the initial marking, the initial places are marked with tokens. Each initial place corresponds to a certain part (job) and the number of tokens it carries corresponds to the lot size of that specific job. For each initial place
there is a final place which represents the final state of that job (i.e. the finished product form). Accordingly, in the final marking of the Petri net model, each final place should be marked with a number of tokens equal to that which was located in its corresponding initial place. In other words the final marking (state) of the FMS indicates that each job batch (lot) has been processed.

The last component which could be found in the Petri net model is the intermediate place. The intermediate place is used to indicate that a part has finished a certain process and is about to start its following process. It should be noted that the intermediate place is only utilized in the case of Petri net models that do not consider the transportation constraint. In other words intermediate places are only used in Petri net models that are composed of machining modules.

It remains to be said that the Petri net model is constructed in steps. In other words the model for each job is formed in a sequential order whilst taking into account the common places that represent the shared resources. A schematic example of the Petri net model for the first job in a certain scheduling problem is given in Fig. 3.37. It should be noted that the transportation constraint is neglected, and therefore the model is only composed of machining modules in addition to the initial, intermediate, and final places. In other words this model can only provide a job schedule.

In Fig. 3.37, it can be seen that job 1 goes through four different machining processes. Each process can be performed on two different machines, except for the second process which can only be performed on machine M2. Place $p_1$ and $p_5$ are the initial and the final places, respectively; whereas places $p_2$, $p_3$, and $p_4$ are the intermediate places. It can be seen that the lot size of Job 1 is equal to four since this is the number of tokens in the initial place assuming that the given marking is the initial marking. The blocks representing the different machine modules do not show whether there are any shared places. For the given subnet, there are three shared places which belong to the module of machine M2. These three places are $p_2$, $p_4$, and $p_5$ as given in
Figure 3.37 The schematic diagram of the Petri net model for job 1
the general machining module depicted in Fig. 3.3. The same subnet is formed for the rest of the jobs involved in the scheduling problem. If any machining module is repeated as was the case with M2 the same corresponding places are shared amongst these duplicate modules. The same construction strategy is followed in the case of including transportation modules. However, in this case the transportation modules replace the intermediate places.

### 3.5 The Colored Petri Net Model

In this research colored Petri net models will only be considered in the case of job schedules. In other words the colored Petri net model proposed in this research cannot accommodate AGVs and therefore does not include any transportation modules.

In the case of colored Petri nets the same general machining module, illustrated in Fig. 3.3, will be utilized. However, in the case of the proposed CPN model each job-type will be assigned a certain color set. Thus tokens representing job J1 will carry a different "color" from that which is carried by tokens belonging to job J2, and so on. This color set will carry information related to each token. Thus, each token will have a color set that defines its job-type, processing requirements (route), current processing status, and its corresponding operation times. In other words each token has all its relevant information built into it. Therefore, in the case of CPNs one does not need to model the route of each job, as in the case of TPPNs (as shown in Fig. 3.37), since each token knows where it is heading.

Therefore in the case of colored Petri nets each machine needs its module to be constructed only once. In CPNs places are not shared since each machining module is independent and its places can carry any job token with any color. However, it should be noted that places $p_2$, $p_4$, and $p_6$ of the general machining module will only carry non-colored tokens in the case of CPN models. Place $p_3$ will only carry one colored token at
a time since a machine can only be operating on one part at a time. However, places $p_1$ and $p_5$ will be allowed to carry more than one colored token at a time; since the input and output buffers can carry more than one part (if their capacity is greater than one). In addition places $p_1$ and $p_5$ can carry different colored tokens at the same time since the input and output buffer of any machine can accommodate different job-types (parts).

In addition each initial and final place in the CPN model can carry only one type of color which corresponds to the job-type that this place represents. Obviously the number of colored tokens in each initial place in the initial marking corresponds to the lot-size of that job. As for the intermediate places, they are treated similar to places $p_1$ and $p_5$ in the machining modules. The intermediate places can carry any number of different colored tokens. However, it should be noted that in the case of CPNs intermediate places are associated with the machines not consecutive processes. In other words each machining module has its own intermediate place.

Accordingly, the construction of a colored Petri net model is performed in a different manner. To construct a CPN model, each machine module is constructed independently (and only once), along with its intermediate place. The following step is the creation of the initial and final places. Finally, in order to bring all these components together into one coherent model, directed arcs are constructed between any two modules that need to be connected to each other. For example if a part needs to be transported from machine M2 to machine M5, then a directed arc connecting the intermediate place of M2 to the input transition of M5 is constructed. Similarly, if a part is finished on machine M3, then a directed arc connecting the output transition of M3 to the final place of the concerned job is constructed. Thus the complete CPN model is constructed.

Figure 3.38 is used to explain and illustrate the CPN construction technique along with the possible arc connections.
Figure 3.38 A section of a CPN model
Figure 3.38 demonstrates the fact that in the case of the CPN model the intermediate place is included within the machining module (in this case place $p_r$). In order to represent the colored tokens, one used different token shapes. Thus, one can assume that the rectangular tokens represent Job 1, and that the triangular tokens represent Job 2. The circular tokens are non-colored tokens and represent the availability of the system resources, such as the buffer places and the machines. Thus for the given marking (state) of machine $M_i$ in Fig. 3.38, one finds that the input buffer contains one part of Job 1 and one part of Job 2, and that there is an extra empty space in the buffer. Accordingly, the machine $M_i$ is processing a part that belongs to Job 1, whereas the output buffer has one part of Job 2 residing in it, and it has three empty spaces.

Figure 3.38 illustrates how the different modules in the CPN model are connected to each other. Thus, an initial place of any part ($p_{\text{initial}}$) is connected to any machining module (if required) as shown in Fig. 3.38. After a part leaves the output buffer of a machine, it enters the corresponding intermediate place. If the part is a finished product, then a directed arc is constructed to connect the last transition of the machining module ($t_5$ in Fig. 3.38) to the corresponding final place ($p_{\text{final}}$). On the other hand if the part has to visit other machines, then a directed arc is constructed to connect the intermediate place to the corresponding machines. Thus, in the last case directed arcs connect the intermediate place to the first transition in the required machining modules (i.e. transitions similar to $t_1$ in Fig. 3.38).

Thus the whole CPN model of the system can be constructed without having to duplicate or repeat any places or transitions. Therefore, the advantage of using identifiable colored tokens is the reduction of the size of the model.
Chapter 4
THE SCHEDULING ALGORITHM

The heuristic search algorithms that are proposed in this research are based on the well-known A* algorithm. The A* algorithm is defined as a graph search algorithm with the objective of finding a shortest path using a certain heuristic evaluation function. This heuristic evaluation function helps in guiding the search through the graph. Research in the area of graph-searching began in the late 1950s. At that time the use of heuristic information to increase the efficiency of graph-searching was being studied in the fields of artificial intelligence (AI) and operations research (Pearl 1985). In 1968 Hart, Nilsson, and Raphael introduced the A* algorithm and proved its applicability in finding an optimal solution for the graph searching problem. For this reason, the A* algorithm is currently considered to be a well-established search tool in the field of AI.

Before discussing the A* algorithm and how it can be applied in conjunction with a Petri net model to generate a schedule, one needs to introduce some basic concepts regarding the topic of graphs and graph searching methodology. These fundamental terms and definitions are given in accordance with the work of Pearl (1985). The following description will assist in the explanation of the proposed heuristic search algorithms.

4.1 The Fundamentals of Graph-Searching

A graph consists of a set of nodes, and for each graph there is a start node called node s. This start node defines the initial problem or state that is to be considered. Within the graph structure some nodes will be connected to each other by directed arcs (links). In AI literature these arcs are called operators or the production rules. If a directed arc is pointing from a certain node n towards another node n', then node n is called the parent (or father) of node n', and node n' is called the successor (or child) of node n. In the case of the graph considered in this study, one finds that the
arcs connecting the nodes to each other are assigned a certain cost which affects their role in determining the final solution (schedule). This cost is denoted as \( c(n, n') \) and is assigned to the corresponding arc heading from node \( n \) to node \( n' \).

If for a sequence of nodes \( n_1, n_2, n_3, \ldots, n_k \), each node \( n_i \) is the successor of node \( n_{i-1} \), then there exists a path of length \( k \) from node \( n_1 \) to node \( n_k \). In this case node \( n_k \) is called the descendant of node \( n_1 \), and accordingly node \( n_1 \) is called the ancestor of \( n_k \). In general (and in the case of this study) the cost of any path is equal to the summation of the costs of the individual arcs out of which this path is composed.

Thus one has presented the major and necessary terms to understand and describe graphs. Therefore, one can now consider the terms and steps involved in the field of graph searching.

The first and most elementary step in the process of graph searching is called node generation. The term node generation is used to describe the process of computing a node (i.e., its representation code) from the code of its parent. This new successor is said to be generated and its parent is said to be explored. The next step in the process of graph searching is called node expansion. The term node expansion describes the process of generating all the successors emanating from the parent node. Thus the parent node is said to be expanded. During the node expansion stage pointers are usually set up from each successor linking it to its parent. These pointers help in determining the path leading from any given node \( n \) to any of its ancestors or to the starting node. In our scheduling problem these pointers are extremely important for the sake of producing a final schedule.

The graph searching procedure (strategy) assists in determining the order in which nodes are to be generated. One has to distinguish between an uninformed (blind) search strategy and an informed (guided) search strategy. In the former, the order in which nodes are chosen to be expanded depends solely on the information
gathered by the search. On the other hand, in the latter search strategy the order of expansion is affected by the unexplored portion of the graph in addition to the nature of the goal of this search.

Thus one can divide the nodes involved in the graph under consideration into four different subsets (Pearl 1985). These four subsets are as follows:

1. Nodes that have been expanded
2. Nodes that have been explored but not yet expanded,
3. Nodes that have been generated but not yet explored,
4. Nodes that are still not generated.

Several graph searching procedures, including the A* algorithm, require a distinction between the nodes that have been expanded and the nodes that have been generated but not yet explored (i.e. subsets 1 and 3). Therefore, two separate lists called the CLOSED and the OPEN lists are used to distinguish between, and keep track of, these two different node classifications. The CLOSED list includes all the nodes that have been expanded, whereas the OPEN list has all the nodes that have been generated and are awaiting expansion. In other words the OPEN list maintains all the nodes that represent the boundaries that have been reached by the graph search algorithm.

4.2 The Reachability Graph

After discussing the various terms used to describe graphs and the graph searching process, one needs to introduce the specific graph that will be considered in this research. Obviously, this graph has to be related to Petri nets and has to offer a means for creating a production schedule. One finds that the reachability graph satisfies the above requirements. The reachability graph (or coverability tree) of a Petri net describes all the possible changes in the marking of the net. Since each marking
represents a certain state of the system being modeled, the reachability graph provides a complete record of all the possible behaviors that the system can assume. Thus in the case of modeling an FMS, the reachability graph provides the state space within which this FMS can operate. A Petri net example is given in Fig. 4.1, and its corresponding reachability graph is given in Fig. 4.2.

Figure 4.1 An example Petri net model
Figure 4.2 The reachability graph of the Petri net in Figure 4.1

It is obvious that each node of the reachability graph is in fact a specific marking of the Petri net that could be reached from the given initial marking. Thus each node in the reachability graph represents a specific state. The representation code which defines each node is the column vector representing the Petri net marking. It is obvious that the start node is in any reachability graph will be the initial marking $M_0$. With the initial marking given, one can construct the entire reachability graph by firing all the enabled transitions in all the markings that are reachable from $M_0$. Therefore, one finds that the arcs in the reachability graph are labeled with transition numbers. This indicates that for a certain marking (parent node) to be transformed into another marking (successor node) this particular transition has to be fired. Thus, for example,
the reachability graph in Fig. 4.2 indicates that for marking M3 to become marking M4, transition t5 has to be fired (i.e. the event modeled as transition t5 has to take place). Thereby, using graph terminology, one finds that the Firing rule of Petri nets is the operator (or the production rules) in the case of reachability graphs.

The reachability graph in Fig. 4.2 indicates that the corresponding Petri net can restore its initial state. This initialization of the system takes place when transition t6 is fired. In addition the reachability graph indicates that marking M4 can be reached from the initial marking M0 through two different paths. The first path is M0-M1-M2-M4 while the other path is M0-M1-M3-M4.

Thus all the possible patterns of behavior of a system are described in the reachability graph. Therefore, if one constructs the whole reachability graph of a Petri net model of an FMS, then a complete description of the possible states that the FMS can assume is provided. In addition a path from the initial marking (i.e. initial state) of the model to the final marking (i.e. final state), represents a particular schedule. Thereby, if one can find the optimal path connecting the initial and final markings, the optimum schedule is achieved. However, unlike the example given in Fig. 4.2, the construction of the entire reachability graph is a formidable task; especially in the case of a manufacturing system such as an FMS. The enumeration of all the possible markings becomes an infeasible and time-consuming task due to the problem of state space explosion.

Therefore, one resorts to the application of a heuristic graph search algorithm to assist in determining the optimal or near-optimal path that leads from the initial marking to the final marking. Thus the application of the proposed modified versions of the A* algorithm on the reachability graph helps in generating an optimal or near optimal schedule. The next section describes the proposed scheduling method and the corresponding graph search technique.
4.3 The Scheduling Algorithm Description

The search strategy employed by the general A* algorithm is considered prior to describing the proposed scheduling algorithms. The A* algorithm is classified as a heuristic best-first search procedure, whose goal is to find a path with minimum total cost. Thus the A* algorithm is placed in the group of best-first search procedures. Similar to other searching procedures, the algorithms within the best-first search category utilize heuristic information to decide which node should be expanded next. However, the best-first strategy is distinguished from the other search methods by its basic principle of choosing the best alternative from amongst ‘all’ the available nodes, no matter where they are located in the partially developed graph (Pearl 1985). This principle necessitates the fact that best-first algorithms store and preserve a memory of their past attempts (i.e. remember all the generated nodes).

Another common feature found amongst all the best-first search algorithms is the estimation of the promise/potential of each generated node. This estimation is performed numerically using a heuristic evaluation function, denoted as, \( f(n) \). This evaluation function offers a measure of the possibility of finding the solution path linking the initial state to the final state passing through the node under consideration. Usually, the lower the value of \( f(n) \) the higher the potential of the node \( n \). Therefore, the node that is selected for expansion is the one carrying the lowest value of \( f \).

In this research two similar and closely related scheduling algorithms are proposed to solve the scheduling problem on hand. The first algorithm is called the L1 search algorithm, whereas the second algorithm is called the Limited-Expansion A algorithm. The former was applied by Lee and DiCesare (1992), while the latter was introduced by Sun, Cheng, and Fu (1994). Both algorithms are based on the A* search algorithm. The L1 algorithm is identical to the general A* algorithm in terms of the steps that are performed. The only difference is that the terms used in the L1 algorithm are adapted to the representation form and the problem domain under consideration (i.e.
Petri nets and FMS). In the case of the Limited-Expansion A algorithm, a small modification is imposed on the A* algorithm. Both heuristic search algorithms are presented in the following sections.

### 4.3.1 The L1 Algorithm

The L1 algorithm is presented using both graphical and verbal means. The graphical representation of the algorithm is provided through the flowchart illustrated in Fig. 4.3. The verbal description is depicted in the following list of steps:

1. Place the initial marking $M_0$ on the list OPEN.
2. If OPEN is empty, terminate with failure.
3. Remove the marking $M$ with the minimum value of $f$ from the list OPEN and put it on the list CLOSED.
4. If $M$ is the final marking, construct the path from the initial marking to the final marking using the pointers, and terminate.
5. Otherwise, find all the enabled transitions in the marking $M$.
6. Generate the successor marking for each enabled transition, and set pointers from the successor marking to its parent marking $M$.
7. For each successor marking $M'$ do the following:
   a) If marking $M'$ is not already on list OPEN or list CLOSED, then calculate $f(M')$, and place $M'$ on the list OPEN.
   b) If marking $M'$ is already on OPEN, direct its pointer along the shorter path (i.e., the path yielding the smallest value of $g(M')$).
   c) If marking $M'$ is already on CLOSED, direct its pointer along the shorter path and if $M'$ requires pointer redirection, move it from list CLOSED to list OPEN.
8. Go to step 2.

---

1 For the ease and clarity of representation, Fig. 4.3 has been divided into two halves. The two parts are linked through the corresponding directed arcs.
PN Model & Mo & Mn

Put Mo on the list OPEN

Is OPEN empty?

Remove first marking M from OPEN and put it on CLOSED

Is M the final marking?

Find the enabled transitions in Marking M

Construct the optimal path from the initial to the final marking

Optimal/Near Optimal schedule has been generated

Terminate with failure
Figure 4.3 The L1 scheduling algorithm
The only difference between the L1 algorithm and the general A* algorithm is that the L1 is specifically developed for searching the reachability graph of a Petri net. Therefore, instead of using generic terms such as start node, goal node, and successor nodes, the L1 algorithm calls on the initial marking, the final marking, and the successor markings. Nevertheless, the search strategy of the L1 algorithm remains similar to that of the A*. The L1 algorithm attempts to expand the nodes of the reachability graph from the initial marking until the generated portion of the partially developed graph reaches the final marking. In step 4 of the algorithm one finds that once the algorithm discovers that it has reached the final marking, it immediately constructs the solution path by tracing back the pointers that lead from the final marking (node) to the initial marking (node).

Since each pointer (arc) in the reachability graph is labeled with a transition, the solution path is provided in the form of a sequence of transitions. This transition sequence, which is obtained from the L1 algorithm, provides the firing order which should be started from the initial marking to reach the final marking. The transitions offer the order in which certain events should be initiated. Thus the schedule is achieved in the form of a sequence of events (Lee and DiCesare 1992). Therefore, the schedule is event driven rather than time driven.

From the description of the algorithm details one finds that the L1 algorithm utilizes the heuristic evaluation function $f$ to guide and direct its search through the reachability graph. Since, the algorithm specifically deals with Petri nets, $f$ is a function of the marking $M$. The following equation is used to calculate the value of $f(M)$:

$$f(M) = g(M) + h(m) ; \text{ where}$$

- $f(M)$ = The estimate of the cost of an optimal path which goes through marking $M$
- $g(M)$ = The current lowest cost obtained from the initial marking to marking $M$
- $h(M)$ = The estimate of the cost of the optimal path from $M$ to the final marking

(Equation 4.1)
Since the objective of this research is to minimize the makespan, the term 'cost' signifies the factor of time. Thereby the function \( f(M) \) provides an estimate of the makespan in the case that the state (marking) \( M \) is encountered during the application of the optimal schedule. Thus the additive evaluation function provides an estimate measure of the potential of each marking (node) that is generated during the search of the reachability graph. It is apparent that this evaluation function depends on two sources of information. The first source provides the information gathered by the search up to that point, while the second source relies on the problem domain and the unexplored portion of the graph. In other words, equation 4.1 states that the cost (makespan) of a path heading from the initial to the final marking (i.e. schedule) and passing through \( M \), is divided into two parts. The first part is equal to the minimum time (cost) required to reach the marking \( M \) from the initial marking (i.e. the value of \( g(M) \)). The second part is equal to the time required to reach the final marking from the current marking \( M \) (i.e. the value of \( h(M) \)). It is obvious that \( g(M) \) is a definite function which requires no approximations, whereas the function \( h(M) \) is a heuristic function due to the fact that one attempts to guess the amount of time left for the schedule to be completed.

Hence the cost \( c(n,n') \) that is associated with the arc connecting node \( n \) to its child node \( n' \) in the reachability graph, corresponds to the time delay associated with the transformation of node \( n \) (marking \( M \)) to node \( n' \) (marking \( M' \)). Assuming that the firing of transition \( t \) converts \( M \) to \( M' \). If all the input places of transition \( t \) are immediate places, then the firing of transition \( t \) does not consume any time. Thus the cost of the arc connecting \( M \) to \( M' \) will be equal to zero in this case. Otherwise, the cost of the arc will be calculated using the following equation:

\[
c(n,n') = \max (r_{t1}, r_{t2}, \ldots, r_{tk}); \quad \text{where} \quad (\text{Equation 4.2})
\]

\( r_k = \) The remaining time delay of the input place \( p_i \) of transition \( t_i \) for \( i = 1, 2, \ldots, k \)
The reason for using the remaining time of the input places is to take into account the feature of concurrency (Lee and DiCesare 1994). After considering the cost of the arcs one can calculate the value of the g function using the following equation:

\[ g(n') = g(n) + c(n,n') \]  
(Equation 4.3)

Obviously, the symbols \( n \) and \( n' \) in Equation 4.3 can be replaced by \( M \) and \( M' \), respectively. In addition it is assumed that \( g(M_0) \) is equal to zero, since the time is set to zero at the initial marking (i.e. initial state of the FMS).

The heuristic function \( h(M) \) is calculated using the following equation:

\[ h(M) = -w \cdot \text{dep}(M); \text{ where} \]  
(Equation 4.4)

\( w \) = weighting factor  
\( \text{dep}(M) \) = depth of the marking \( M \)

The heuristic function is supposed to give a indication of the additional amount of time that is required for the FMS to reach the final state from the current state (marking \( M \)). In this research the heuristic function relies on the fact that the objective of the scheduling problem is to minimize the makespan. Thus it is assumed that the deeper the marking is in the reachability graph the closer it is to the final state. The depth of any marking \( M \) is equal to the number of transition firings that have taken place in order for the marking \( M_0 \) (initial state) to reach the marking \( M \) (current state). According to the definition of the heuristic measure, the deeper the marking the smaller the period of time required for it to reach the final state. In other words, the bigger the value of \( \text{dep}(M) \) the smaller the value of \( h \). This inverse relationship is expressed through the negative sign. In addition the weighting factor \( w \) reflects the emphasis placed on this heuristic guess (assumption).
The L1 algorithm is an admissible algorithm. In other words the L1 algorithm will always find an optimal path (i.e. optimal schedule) if the function \( h(M) \) satisfies the following condition:

\[
\text{h}(M) \leq h^*(M) \forall M; \quad \text{where} \\
\text{h}^*(M) = \text{The cost of the optimal path leading from M to the final marking}
\]  

(Equation 4.5)

However, the heuristic function used in this research does not fulfill the admissibility condition. Therefore, the scheduling algorithm employed does not guarantee that an optimal solution (i.e. optimal schedule) will be reached. Thus, a sensitivity analysis experiment will be conducted on the algorithm in order to reach the best possible compromise between the quality of the solution and the time required to generate the solution.

In Fig. 4.3 the input required for the algorithm to start operating is set to be the initial and final marking in addition to a description of the Petri net's structure which in this case is the A matrix. After the input is provided to the algorithm, it immediately forms the OPEN and the CLOSED lists. The list OPEN maintains all the markings that have been generated but not yet explored, whereas the list CLOSED maintains all the markings that have been expanded. It is evident from the description of the algorithm that both lists are updated after each iteration (i.e. the main loop of the algorithm).

Step 2 informs the algorithm to stop if all the generated nodes are exhausted before reaching the final marking. Thus there exists no path between the given initial and final markings. Steps 3 to 6 are concerned with the selection of a certain marking for expansion and the generation of all its successors. Obviously the heuristic evaluative function \( f(M) \) plays a pivotal role in this selection process (i.e. it guides the search). The result of these steps is that the expanded marking is removed from the OPEN list and placed on the CLOSED list, whereas the generated successors are added to the OPEN list. If in the execution of step 5 it was found that there are no
enabled transitions, then the following steps will have no effect and the algorithm will return to step 2 to select a different node. Thus in the case of the presence of a deadlock (i.e. a marking with no enabled transitions) the algorithm simply shifts its search to a different path. The effect of such an iteration is the removal of the corresponding marking from the OPEN list and its placement on the CLOSED list.

The aim of step 7 is to ensure that one of the basic principles of the search algorithm is maintained. This principle states that if two paths lead to the same node (marking), the one with the higher value of f is discarded while the other is kept for further expansion. Thus the algorithm checks if each newly generated marking was previously discovered (revealed) by the search. If the newly generated marking M already exists in the OPEN list, a comparison is made on the basis of the value of f(M). The marking with the lower value of f(M) is discarded while the other is kept in OPEN. If the newly generated marking M is found on the CLOSED list, the same comparison is made. In this case if the old marking is better the newly generated marking is discarded and no changes take place on both lists. However, if the new marking is better, the old marking is discarded and the new one is placed on the OPEN list. The reason for reinstalling the marking M in OPEN is to regenerate its descendants, if required, and to redirect their pointers along the path with the minimum cost.

Finally step 4 ensures that when the final marking is reached the algorithm stops and creates the path leading from the initial marking to the final marking. The cost or the makespan of the resultant schedule is equal to the summation of the cost of the arcs out of which the path is composed.

4.3.2 The Limited-Expansion A algorithm

The Limited-Expansion A algorithm can be regarded as a modified version of the A* algorithm. Since the A* is a specialized best-first search algorithm, it attempts to keep a record of all the generated nodes (markings). Obviously this poses the problem
of having to allocate a huge amount of memory for storing all these generated markings. The modification introduced by the Limited-Expansion A algorithm aims at reducing the memory requirement and the need for computer resources. Thus the algorithm assigns a maximum capacity, \( b \), for the number of markings that can be put in the OPEN list. If the number of markings in the OPEN list exceeds the value of \( b \), a pruning procedure takes place. In this procedure the markings with the highest values of \( f(M) \) (i.e. the worst markings) are deleted from the list until the number of markings in the OPEN list reaches the value of \( b \). Thus the Limited-Expansion A algorithm ensures that the maximum number of markings in the OPEN list never surpasses \( b \).

Thus the modification incurred by the Limited-Expansion A algorithm will cause a change in the stages that describe the L1 algorithm. However, the effect will be simply having to add two more steps to the algorithm description. The first 7 steps of the two algorithms are identical. The following steps of the Limited-Expansion A algorithm are given as follows:

8. If there are more than \( b \) markings on OPEN, truncate the marking \( M \) with the highest value of \( f(M) \) from the list OPEN
9. Go to step 8
10. Go to step 2

Obviously, this algorithm does not guarantee reaching an optimal solution even if an admissible heuristic function was used. Therefore, the Limited-Expansion A algorithm attempts to reach a compromise between the quality of the generated schedule (i.e. optimality) and the memory requirements. Accordingly this algorithm can significantly reduce the search time if a relatively small value of \( b \) is assigned to the OPEN list.

Finally, this research demonstrates that both algorithms can be applied on TPPN models and CPN models. The same search strategy is implemented and the algorithm
steps are not affected by the type of Petri net model that is used. However, the differences lie in the technicalities of the representation and the means of performing certain steps. In the case of TPPN models both algorithms can be applied exactly as has been described.

However, in the case of CPNs different methods have to be used. Obviously the representation of a marking M in a CPN differs from that of a TPPN. Therefore the code representing each node in the reachability graph of a CPN differs from that used in the case of a TPPN model. In addition the arcs connecting the nodes (markings) of the reachability graph are labeled with a color set in addition to the transition number. This is due to the fact that the firing of a transition in a CPN takes place with respect to a certain color set. Accordingly, the step of finding all the enabled transitions in the marking chosen for expansion, is performed in a different manner in the case of CPN. Finally, the schedule is provided in the form of a sequence of two-tuples that are composed of the transition that was fired and the color set with respect to which it fired.
Chapter 5
THE SENSITIVITY ANALYSIS

It is clear that the scheduling problem considered in this research is affected by several parameters. Any variation in one or more of these parameters results in a change in the final schedule that is reached by the search algorithm. Thus, the main goal of this chapter is to study the impact of changing the input parameters (data) of the problem on the final solution. Obviously, there are numerous parameters that can affect the final solution, however the sensitivity analysis that is performed in this chapter only considers the main parameters.

In addition, this chapter aims at illustrating the two major characteristics that were discovered through the application of the proposed scheduling technique. The first concept that was revealed by this research is that having extra places and transitions in the Petri net model affects the search algorithm. The second characteristic that will be discussed is the ordering of equal-cost states in the OPEN list of the heuristic search algorithm. These two characteristics offer a new and more profound understanding of the proposed scheduling technique. In addition the second characteristic is considered to be an improvement on the scheduling technique proposed by Lee and DiCesare (1992), since it causes favorable changes in the scheduling algorithm (however in some and not all cases).

Thus, this chapter offers the sensitivity analysis that has been conducted on several parameters. Along with that, a description of the proposed changes that result in an improvement in the efficiency and effectiveness of the scheduling approach, is provided. Two main case studies are used to assist in proving and conveying the findings of this research.
5.1 The Parameters

The problem specification stage is the starting point of any scheduling problem. In order to provide a full and precise description of the scheduling problem that is to be considered in this research, several properties have to be defined. The question/answer that has been proposed helps in accomplishing and completing the task of fully specifying the scheduling problem. Thus a process plan table of the various jobs (with the corresponding lot-sizes) that are involved, along with a description of the FMS configuration, offer a sufficient description of the problem on hand. However, for the sensitivity analysis to be conducted, some additional information has to be included. This information should provide the values of w (the weighting factor) and b (the maximum limit of the OPEN list), along with a description of the ordering strategy that will be used in the OPEN list. Evidently, this additional information is directly related to the heuristic search algorithm.

Any change in this input data would affect the final schedule that is reached. However, in this chapter one concentrates on the effect of changing the parameters that are related to the search algorithm. Changes regarding the other factors render a totally different problem. Nevertheless, the effect of changing the other variables is considered with respect to the size of the Petri net model and the resulting schedule.

Finally, one needs to define the output variables that will be used for comparison in this sensitivity analysis. The results and performance of the scheduling algorithm will be compared through three factors. These factors are the number of iterations performed, the schedule generation time, and the total schedule time (makespan). The number of iterations correspond to the number of times that the search algorithm attempts to generate new nodes (performs its main loop). The schedule generation time is directly related to the number of iterations and provides the amount of time that was required by the search algorithm to reach the final marking from the given initial marking (i.e. generate the schedule). The makespan is the resultant schedule's total
completion time (cost of the path from the initial to the final marking). In addition, the depth of the final marking will also be provided to indicate the number of transitions that have been fired to form the schedule (i.e. the length of the path). For the same scheduling problem, this value is constant, and therefore it will not be used during the comparison. However, it provides a useful and simple indication of the size of the problem that is being considered.

Two main case studies will be discussed and analyzed in this research. The first case study will be used to present the improvements offered by this research on the given scheduling approach, in addition to the important related characteristics that have been revealed. The second case study will be mainly used to indicate the effect of changing the problem input data. In both cases the effect of changing the parameters, w and b, will be studied.

5.2 Case Study 1

The first case study is taken from the work of Lee and DiCesare (1994a). Table 5.1 provides the process plan of the required job parts along with the required lot-size of each part. It can be seen that each part requires four processes and that the lot size is set to be 10 for each product type. In addition the FMS under consideration consists of 3 machines, M1, M2 and M3.

Table 5.1 The process plan for Case Study 1

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>M1 / M3</td>
<td>M1 / M2</td>
<td>M1 / M2 / M3</td>
<td>M2 / M3</td>
<td>M1 / M3</td>
</tr>
<tr>
<td>2</td>
<td>M2</td>
<td>M3</td>
<td>M2 / M3</td>
<td>M1 / M3</td>
<td>M2 / M3</td>
</tr>
<tr>
<td>3</td>
<td>M1 / M3</td>
<td>M1 / M2</td>
<td>M1 / M3</td>
<td>M2 / M3</td>
<td>M1 / M2</td>
</tr>
<tr>
<td>4</td>
<td>M1 / M2</td>
<td>M1 / M3</td>
<td>M1 / M2</td>
<td>M1 / M2 / M3</td>
<td>M1 / M2 / M3</td>
</tr>
</tbody>
</table>

138
Table 5.2 The FMS configuration for Case Study 1

<table>
<thead>
<tr>
<th>Components/Resources</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer Capacity</td>
<td>Infinity (The buffer constraint is neglected)</td>
</tr>
<tr>
<td>AGVs</td>
<td>None (The transportation constraint is not considered)</td>
</tr>
<tr>
<td>AGVS</td>
<td>Does not apply</td>
</tr>
<tr>
<td>AGVS Layout</td>
<td>Does not apply</td>
</tr>
<tr>
<td>Robots</td>
<td>Does not apply</td>
</tr>
<tr>
<td>Machines in Cells</td>
<td>None</td>
</tr>
</tbody>
</table>

It is apparent from Table 5.2 that the required schedule will only consider machines. Neither the buffer nor the transportation constraints are considered in this problem. Thus only machining operation times are given in Table 5.3.

Table 5.3 The operation times for Case Study 1

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time</th>
<th>Operation</th>
<th>Time</th>
<th>Operation</th>
<th>Time</th>
<th>Operation</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>O_{1,1,1}</td>
<td>7</td>
<td>O_{2,3,2}</td>
<td>14</td>
<td>O_{3,4,2}</td>
<td>3</td>
<td>O_{5,1,3}</td>
<td>15</td>
</tr>
<tr>
<td>O_{1,1,3}</td>
<td>4</td>
<td>O_{2,4,1}</td>
<td>8</td>
<td>O_{4,1,2}</td>
<td>9</td>
<td>O_{5,2,2}</td>
<td>7</td>
</tr>
<tr>
<td>O_{1,2,2}</td>
<td>3</td>
<td>O_{2,4,3}</td>
<td>4</td>
<td>O_{4,1,3}</td>
<td>5</td>
<td>O_{5,2,3}</td>
<td>14</td>
</tr>
<tr>
<td>O_{1,3,1}</td>
<td>3</td>
<td>O_{3,1,1}</td>
<td>10</td>
<td>O_{4,2,1}</td>
<td>6</td>
<td>O_{5,3,1}</td>
<td>5</td>
</tr>
<tr>
<td>O_{1,3,3}</td>
<td>6</td>
<td>O_{3,1,2}</td>
<td>15</td>
<td>O_{4,2,3}</td>
<td>2</td>
<td>O_{5,3,2}</td>
<td>8</td>
</tr>
<tr>
<td>O_{1,4,1}</td>
<td>2</td>
<td>O_{3,1,3}</td>
<td>8</td>
<td>O_{4,3,2}</td>
<td>7</td>
<td>O_{5,4,1}</td>
<td>4</td>
</tr>
<tr>
<td>O_{1,4,2}</td>
<td>4</td>
<td>O_{3,2,2}</td>
<td>2</td>
<td>O_{4,3,3}</td>
<td>12</td>
<td>O_{5,4,2}</td>
<td>6</td>
</tr>
<tr>
<td>O_{2,1,1}</td>
<td>8</td>
<td>O_{3,2,3}</td>
<td>6</td>
<td>O_{4,4,1}</td>
<td>9</td>
<td>O_{5,4,3}</td>
<td>8</td>
</tr>
<tr>
<td>O_{2,1,2}</td>
<td>12</td>
<td>O_{3,3,1}</td>
<td>2</td>
<td>O_{4,4,2}</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>O_{2,2,3}</td>
<td>4</td>
<td>O_{3,3,3}</td>
<td>4</td>
<td>O_{4,4,3}</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>O_{2,3,1}</td>
<td>7</td>
<td>O_{3,4,1}</td>
<td>6</td>
<td>O_{5,1,1}</td>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

139
The notation operation $O_{i,j,k}$ corresponds to the performance of the $i^{th}$ process of the $j^{th}$ product on the $k^{th}$ machine. In this particular case study there are 41 operations and their duration times are as given in Table 5.3 (Lee and DiCesare 1994a). The time unit of the operations given in Table 5.3 is assumed to be in minutes. Therefore the makespan (completion time) of the final schedule will be expressed in minutes. Thus the formulation of the case study problem is complete.

5.2.1 The Effect of Additional Places and Transitions

Since the transportation constraint is neglected in this case study, no transportation modules will be used. The Petri net model will be composed of several interconnected machine modules. In addition, in this case study, buffer capacities are assumed to be infinite, which results in the utilization of the modified machining module which is repeated in Fig. 5.1.

![Diagram](https://via.placeholder.com/150)

*Figure 5.1 The machining module with infinite buffer capacity*
The Petri net model that was constructed to represent the given scheduling problem using the machining module given in Fig. 5.1 was found to be composed of 151 places and 164 transitions. On the other hand, the number of places and transitions in the model constructed by Lee and DiCesare (1994a) was 69 and 82, respectively. This drastic reduction in the size of the model was due to the fact that instead of using the machining module in Fig. 5.1, Lee and DiCesare utilized a minimized version of it, where places \( p_1 \) and \( p_4 \) were discarded. In other words, the number of parts in the input and output buffer of any machine was completely neglected due to the fact that the buffer capacity is infinite and that number is of no significance\(^1\). Therefore places \( p_1 \) and \( p_4 \) in the machining module given in Fig. 5.1 are additional places and accordingly transitions \( t_1 \) and \( t_4 \) are additional transitions. Thus the reduced machining module is shown in Fig. 5.2.

Figure 5.2 The minimized machining module used by Lee and DiCesare

Obviously, both Petri net models given in Fig. 5.1 and 5.2 represent the same machine with the same constraints. However, the model given in Fig. 5.2 is more efficient in the sense that it utilizes a smaller number of transitions and places. Nevertheless, the proposed machining module (Fig. 5.1) was used to model the given scheduling problem and a comparison of the results obtained against those previously achieved was conducted to validate the modeling technique and the scheduling algorithm. The results and effect of using various values of \( w \) (the weighting factor) were calculated for the purpose of validation and sensitivity analysis. These results are

---

\(^1\) However, it should be noted that these places affect the logic of the model. Thus these places and their corresponding transitions are not redundant, yet their removal was necessary to validate the research results.
given in Table 5.4. It should be noted that these results were obtained using the L1 search algorithm, which means that there was no maximum limit assigned to the size of the OPEN list.

Table 5.4 The results for Case Study 1 using the machining module in Figure 5.1

<table>
<thead>
<tr>
<th>The weighting factor (w)</th>
<th>Total number of iterations</th>
<th>Depth of final marking</th>
<th>Generation time (sec)</th>
<th>Schedule makespan (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1121</td>
<td>800</td>
<td>112</td>
<td>634</td>
</tr>
<tr>
<td>5</td>
<td>877</td>
<td>800</td>
<td>99</td>
<td>778</td>
</tr>
<tr>
<td>10</td>
<td>838</td>
<td>800</td>
<td>108</td>
<td>941</td>
</tr>
<tr>
<td>20</td>
<td>800</td>
<td>800</td>
<td>99</td>
<td>986</td>
</tr>
<tr>
<td>50</td>
<td>800</td>
<td>800</td>
<td>99</td>
<td>986</td>
</tr>
</tbody>
</table>

Obviously, one expected the number of iterations and the depth of the final marking achieved to be greater than those obtained by Lee and DiCesare due to the fact that a larger model was used in this research to represent the same system. However, one expected to reach the same final result (i.e. the schedule makespan to be achieved) since the same FMS and scheduling problem was being considered using the same search algorithm.

Surprisingly, the scheduling makespan time obtained in Table 5.4 was completely different from those tabulated in the work of Lee and DiCesare. The results obtained in Table 5.4 were by far, much worse than their predecessors. The previously calculated values were in the range of the 420s and 430s. Thus the results obtained in this trial were either 1.5 times greater or even more than double the value.
Therefore, in order to ensure that the developed heuristic search algorithm was functioning properly and in order to validate the whole scheduling technique, one had to resort to the implementation of the reduced machining module given in Fig. 5.2. Thus one was certain that the same Petri net model was constructed, and accordingly the results should reflect the authenticity of the developed heuristic search algorithm. The corresponding results, given in Table 5.5, were found to be identical to those obtained by Lee and DiCesare.

Table 5.5 The results for Case Study 1 using the machining module in Figure 5.2

<table>
<thead>
<tr>
<th>The weighting factor (w)</th>
<th>Total number of iterations</th>
<th>Depth of final marking</th>
<th>Generation time (sec)</th>
<th>Schedule makespan (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>757</td>
<td>400</td>
<td>34</td>
<td>426</td>
</tr>
<tr>
<td>5</td>
<td>411</td>
<td>400</td>
<td>12</td>
<td>447</td>
</tr>
<tr>
<td>10</td>
<td>400</td>
<td>400</td>
<td>16</td>
<td>438</td>
</tr>
<tr>
<td>20</td>
<td>400</td>
<td>400</td>
<td>16</td>
<td>438</td>
</tr>
<tr>
<td>50</td>
<td>400</td>
<td>400</td>
<td>16</td>
<td>438</td>
</tr>
</tbody>
</table>

The only difference between the results obtained in Table 5.5 and those obtained by Lee and DiCesare lie in the value of the generation time. They stated that in order for their algorithm to reach a final solution, it took an amount of time ranging between 5 and 10 minutes. In Table 5.5 the maximum required generation time was 34 seconds, and even in the case of the larger model in Table 5.4 the maximum generation time was under 2 minutes. The main reason for this significant reduction in the schedule generation time is the continuous advancements that are being achieved in the microprocessor technological field. Thus, one has to mention that the results calculated in this research were obtained using a 486 DX-100 PC which had 16
megabytes of RAM (unless said otherwise). Obviously, if a more powerful computer with additional resources is used, then the generation time would be even smaller.

Thus, one has managed to validate the heuristic algorithm and the scheduling technique as a whole. In addition, one has discovered that the use of additional places/transition in the Petri net model leads to faulty results and probably bad schedules which are far from the optimum.

The reason for this effect of the additional places and transitions is that the heuristic function $h(M)$ relies on the depth of the marking. Thus depending on the depth of a corresponding marking the heuristic search function $f(M)$ decides if it should or should not pursue the path on which this marking is located. Thus through the addition of extra places and transitions in the Petri net model, one actually affects the decisions taken by the heuristic function. The addition of extra places and transitions may not affect the Petri net model and its representation reliability. However, these additions misguide the heuristic search algorithm during its quest of seeking for a minimum cost path leading from the initial marking to the final marking.

For instance, in the case of the two Petri net models that have been constructed, consider the situation where two identical states of the system are represented by both models. In the case of the large Petri net the depth of the marking (state) will be twice as big as the depth of the same state that is represented by the small Petri net model. The reason for this is that the machining module used in the former is composed of four transitions whereas the building block used in the latter contains only two transitions. Thus in order to reach the same state, the number of transitions to be fired (i.e. the depth) in the large Petri net model has to be the double of the equivalent number fired in the reduced Petri net model. Obviously, the increase in the value of the depth (i.e. $h(M)$) helps in obscuring the impact of the function $g(M)$, and accordingly affects the directions and the paths which the scheduling algorithm decides to follow during its search of the reachability graph.
Apart from affecting the decisions taken by the heuristic search function, the addition of the extra places and transitions in this particular Petri net model also obliterates the effect of operations that take the value of either 3 or 4 time units (minutes). Thus if the heuristic search comes along machining operations which are less than 4 time units, then it will definitely continue along the same path. The reason for this is that the addition of the extra transitions reduces the cost of the path by 4 between every two consecutive operations. It should be noted that the values 3 and 4 occur in the case that \(w\) is equal to 1. In general this limiting value is equal to the product of 4 and \(w\).

Thus it is preferable to remove any extra places or transitions. These additions enlarge the Petri net model which obviously increases the schedule generation time (i.e. search time). Moreover, these additions could have a serious effect on the quality of the schedule that is generated as was the case in this problem.

5.2.2 The Effect of the Ordering Strategy

In the description of the general A* algorithm, it is stated that the node chosen for expansion should be the best node on the OPEN list (i.e. the node with the minimum value of \(f(M)\) in our case). However, if there is a tie where more than one node has the minimum cost, then the selection process should be conducted arbitrarily. Therefore the general A* algorithm does not propose any ordering strategy for equal-cost nodes (i.e. markings).

However, in the application of the L1 and the Limited-Expansion A algorithms an ordering strategy is usually specified. This is due to the fact that the OPEN list is sorted and arranged after each insertion of a new node. Although it was not stated explicitly in the work of Lee and DiCesare, it is apparent that they used the Insert First strategy (IF). This proclamation is based on the fact that the results obtained in Table 5.5, which are
identical to those of Lee and DiCesare, were achieved using the Insert First ordering strategy. Clearly, the IF strategy deals with situations where a certain marking which is to be inserted in the OPEN list is found to have the same cost of a marking which is already located in the list. In the IF strategy, the new marking is placed ahead of the old marking (i.e. it comes first in the list ordering).

In this research, a different ordering strategy was used. This strategy was called the Insert Last strategy (IL). In contrast to the IF strategy the IL strategy placed an equal-cost marking after instead of before the existent markings in the OPEN list. Thus for any equal-cost markings, the IL strategy ensured that the old ones came first in their relative list order.

The results given in Tables 5.4 and 5.5 were obtained whilst the heuristic search algorithm was utilizing the IF strategy. The results that were obtained after applying the IL ordering strategy on the same two models, are given in Tables 5.6 and 5.7. A comparison of the results obtained through the application of both ordering strategies is given in Tables 5.8 and 5.9. It is obvious that the IL ordering strategy renders better results than the IF strategy, in both cases. The resulting reduction in the total completion time is also given in Tables 5.8 and 5.9.

Table 5.6 The results of using the IL ordering strategy in the case of the large Petri net model

<table>
<thead>
<tr>
<th>The weighting factor (w)</th>
<th>Total number of iterations</th>
<th>Depth of final marking</th>
<th>Generation time (sec)</th>
<th>Schedule makespan (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1007</td>
<td>800</td>
<td>95</td>
<td>565</td>
</tr>
<tr>
<td>5</td>
<td>839</td>
<td>800</td>
<td>75</td>
<td>717</td>
</tr>
<tr>
<td>10</td>
<td>801</td>
<td>800</td>
<td>73</td>
<td>870</td>
</tr>
<tr>
<td>20</td>
<td>800</td>
<td>800</td>
<td>73</td>
<td>870</td>
</tr>
<tr>
<td>50</td>
<td>800</td>
<td>800</td>
<td>73</td>
<td>870</td>
</tr>
</tbody>
</table>
Table 5.7 The results of using the IL ordering strategy in the case of the small Petri net model

<table>
<thead>
<tr>
<th>The weighting factor (w)</th>
<th>Total number of iterations</th>
<th>Depth of final marking</th>
<th>Generation time (sec)</th>
<th>Schedule makespan (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>779</td>
<td>400</td>
<td>62</td>
<td>401</td>
</tr>
<tr>
<td>5</td>
<td>423</td>
<td>400</td>
<td>22</td>
<td>398</td>
</tr>
<tr>
<td>10</td>
<td>400</td>
<td>400</td>
<td>20</td>
<td>400</td>
</tr>
<tr>
<td>20</td>
<td>400</td>
<td>400</td>
<td>20</td>
<td>400</td>
</tr>
<tr>
<td>50</td>
<td>400</td>
<td>400</td>
<td>20</td>
<td>400</td>
</tr>
</tbody>
</table>

Table 5.8 The comparison between the IF and IL ordering strategies in the case of the large Petri net model

<table>
<thead>
<tr>
<th>The weighting factor (w)</th>
<th>Insert First ordering strategy</th>
<th>Insert Last ordering strategy</th>
<th>Reduction (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Generation time (sec)</td>
<td>Schedule makespan (min)</td>
<td>Generation time (sec)</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------------------</td>
<td>---------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>2</td>
<td>112</td>
<td>634</td>
<td>95</td>
</tr>
<tr>
<td>5</td>
<td>99</td>
<td>778</td>
<td>75</td>
</tr>
<tr>
<td>10</td>
<td>108</td>
<td>941</td>
<td>73</td>
</tr>
<tr>
<td>20</td>
<td>99</td>
<td>986</td>
<td>73</td>
</tr>
<tr>
<td>50</td>
<td>99</td>
<td>986</td>
<td>73</td>
</tr>
</tbody>
</table>
Table 5.9 The comparison between the IF and IL ordering strategies in the case of the small Petri net model

<table>
<thead>
<tr>
<th>The weighting factor (w)</th>
<th>Insert First ordering strategy</th>
<th>Insert Last ordering strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Generation time (sec)</td>
<td>Schedule makespan (min)</td>
</tr>
<tr>
<td>2</td>
<td>34</td>
<td>426</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>447</td>
</tr>
<tr>
<td>10</td>
<td>16</td>
<td>438</td>
</tr>
<tr>
<td>20</td>
<td>16</td>
<td>438</td>
</tr>
<tr>
<td>50</td>
<td>16</td>
<td>438</td>
</tr>
</tbody>
</table>

Table 5.8 indicates that the Insert Last ordering strategy is unquestionably a better option than the Insert First strategy, in the case of the Petri net model that has additional places and transitions (i.e. places representing the input and output buffer occupied slots). The proof of this statement is that the IL strategy, as shown in Table 5.8, reaches a better schedule in terms of the makespan in a shorter amount of time (smaller generation time). For example, in the case that w is equal to 2 the IL strategy reaches a schedule which is shorter by 69 minutes (a 10.88% reduction) than the IF strategy, in a period of time which is shorter by 17 seconds (a 15.18% reduction). However, it should be noted that the IL ordering strategy did not manage to offset the effect of the additional places and transitions, since the best schedule offered using the IL strategy has a makespan of 565 minutes which is still greater than any schedule generated from the reduced Petri net model.

On the other hand, Table 5.9 indicates that in the case of the reduced Petri net model the IL strategy succeeded in reaching a better solution, and that the heuristic search algorithm took more time to reach the final schedule. However, this increase in
the generation time is outweighed by the magnitude of the reduction that is incurred in the makespan of the final schedule. For example, the worst case occurs when \( w \) is equal to 2. The term "worst case" is used to point out that the increase obtained in the generation time had the highest value encountered, and that the decrease in the total schedule time was the least. Nevertheless, it is obvious that decreasing the schedule completion time by 25 minutes (a 5.87% reduction) totally nullifies the effect of having to spend 28 more seconds to reach this better result. For the other values of \( w \) given in Table 5.9, one finds that the increase in the generation time does not exceed 10 seconds whereas the corresponding reduction in the makespan is at least equal to 38 minutes. Thus one can declare that the use of the Insert Last ordering strategy in this case offers an improvement on the algorithm used by Lee and DiCesare, since better results are reached without expending any significant additional search time.

After analyzing the effect of the ordering strategies, one needs to discuss the reasons for which the replacement of the IF strategy with the IL strategy led to the achievement of better results. First, one has to assert the fact that each ordering strategy causes the heuristic search function to follow a different path (route) through the reachability graph. This fact will certainly occur if the equal-cost markings are at the top of the OPEN list. In other words if more than one marking carries the minimum cost in the OPEN list, then the use of a different ordering strategy will lead to the selection of a different marking. Accordingly, the heuristic search algorithm will follow a different path leading from the initial to the final marking. It should also be noted that in the given Petri net modeling technique of the FMS scheduling problem the situation where more than one marking carries the minimum cost in the OPEN list, will always occur. Thus the effect of using a different ordering strategy will always take place.

The difference that takes place in the node (marking) selection process can be easily traced. In the case that the generated marking was found to be equal in cost to the best marking(s) in the OPEN list, the IF strategy places the marking at the top of the list (i.e. it becomes number 1). On the other hand the IL strategy would place the same
marking after the marking(s) that is (are) equal to its cost (e.g. if there is 5 equal-cost markings at the top of the OPEN list then the newly generated marking will be number 6). Thus, the IF ordering strategy ensures that out of the minimum equal-cost markings the most recently generated marking will be the one chosen for expansion. On the other hand the IL ordering strategy gives priority to the previously generated markings, and ensures that the old markings are selected first for expansion. However, one cannot guarantee whether any particular ordering strategy will definitely lead to a better result. In this particular case study the IL ordering strategy was proven to be better, yet this is by no means a rule. Therefore, one suggests that when this scheduling approach is applied, both ordering strategies should be tested, and the one offering a better result should be used for that particular case.

5.2.3 The Effect of Varying the Value of $w$

The weighting factor, $w$, plays a critical role in this scheduling approach. The value of $w$ directly affects the function $h(M)$ and accordingly the heuristic search function $f(M)$. The relative weight associated with the depth of any particular marking depends on the value of $w$ which is chosen. Accordingly, this has a direct impact on the decisions taken by the search algorithm.

In general the value of $w$ should be tuned according to each specific problem. The impact of the weighting factor, $w$, depends on the average value of the operation times that are involved in the scheduling problem under consideration. Therefore, a relatively small value of $w$ would result in an extensive and rather exhaustive search, whereas a large value would direct the heuristic search function along a certain path without considering any alternatives. The reason for this is that a large value of $w$ (e.g. 50) would totally cancel the effect of the $g(M)$ function, given that no operation takes more than 50 time units (which is usually the case in an FMS). Thus the heuristic search function will be only be guided by the depth of the markings (i.e. $h(M)$) that it generates. In other words the larger the value of $w$, the greater the effect and weight of
the depth and the \( h(M) \) function in determining the direction of the heuristic search. In the case that the value of \( w \) is relatively high the ordering strategy plays an important role at the early stages of the search in determining the schedule path. Thus in general as the value of \( w \) is increased the expected number of iterations should decrease, and the quality of the final schedule would most probably decrease due to the fact that the heuristic search algorithm neglects any alternative routes and follow only one path through the reachability graph.

In Table 5.4 the effect of changing the value of \( w \) is clear. As the value of \( w \) is increased the number of iterations decreases and the quality of the schedule deteriorates. It should be noted that setting the value of \( w \) to 1 was not included due to the fact that the program did not reach a solution due to memory thrashing. It can be seen that for the values of \( w \) of 20 and 50 the number of iterations was equal to its lower bound which is the depth of the final schedule. In other words, during these two trials the heuristic search algorithm did not attempt to explore any other alternative routes. This is due to the fact that the values of 20 and 50 managed to completely cancel the effect of the \( g(M) \) function. Thus any value of \( w \) greater than or equal to 20, when applied along with the L1 search algorithm on this specific problem would render exactly the same results, since the same path is being followed. In addition Table 5.6 indicates that the same conclusions can be drawn with respect to the effect of changing the \( w \) parameter for this particular problem in the case that the Insert Last ordering strategy is applied instead of the Insert First.

The previous analysis was conducted on the Petri net model that included additional places and transitions. The other analysis study that was conducted on the reduced Petri net model revealed some exceptions to the general rule pertaining to the effect of changing \( w \). In Table 5.5 the relation between changing the value of \( w \) and the number of iterations remains unchanged. However, the effect of varying the value of \( w \) on the quality of the solution is slightly distorted. This distortion is apparent in the case where \( w \) is equal to 5. It is obvious that the makespan for this trial is greater than that of
its predecessor (i.e. \( w \) equal to 2), yet the same value is also greater than the makespan obtained for larger values of \( w \) (e.g. 10, 20, and 50). Moreover, the time required to perform the 411 iterations involved in this case is smaller than the generation time required to perform the 400 iterations for the following values of \( w \) (it should be noted that a similar paradox with respect to the generation time was observed in the large Petri net model). The only logical explanation for this exception is that when the value of \( w \) is 5, the algorithm is guided into a path which leads to a relatively long schedule. Along this path the iterations involved do not require the generation of as much markings as other iterations do, and hence the time required to reach the final solution is reduced. It should be noted that in this case, when \( w \) is equal to or greater than 10 the search algorithm is forced to move along a certain path.

In Table 5.7 the use of the IL ordering strategy causes several drastic changes. The increase of the value of \( w \) retains its effect of reducing the number of iterations. However, the relation between \( w \) and the makespan takes on a completely different form. The makespan for all trial values of \( w \) remains almost constant. Moreover the greatest makespan obtained is for the case where \( w \) is equal to 2. The reason for this exceptional behavior is that the path which the large values of \( w \) dictate on the search algorithm actually provides a relatively good schedule. However, one believes that this is an extremely rare case which is unlikely to occur in other problems.

Thus, one can summarize the results of the sensitivity analysis study that was conducted on the variation of the weighting factor. The value of \( w \) directly indicates the effect of the depth of the markings on the decisions taken by the heuristic search function. Relatively high values of \( w \) cancel the effect of the \( g(M) \) function and guide the search algorithm along one particular path. Accordingly, if one is looking for a quick solution for a scheduling problem, then a high value of \( w \) is recommended (e.g. 50). On the other hand lower values of \( w \) maintain a balanced relationship between the \( g(M) \) and the \( h(M) \) functions. Therefore, the search algorithm tries to explore alternative paths, and usually this leads to a better schedule. Due to the advances in the
CHAPTER 5

THE SENSITIVITY ANALYSIS

microprocessor and computer fields the additional time required to search for and generate better schedules is negligible. Thus one recommends the utilization of low values of \( w \), whilst keeping in consideration the magnitude of the operation times and the size of the underlying Petri net model.

5.2.4 The Effect of Varying the Value of \( b \)

Another important parameter in this scheduling approach is the maximum capacity of the OPEN list. This maximum limit, \( b \), helps in reducing the required computer resources and quickens the search task by eliminating "bad" or high-cost markings. Obviously, all this is done at the expense of the quality of the generated schedule. This section attempts to study the effect of varying the value of \( b \).

In the previous analysis all the problems were being solved by applying the L1 algorithm where there is no limit on the size of the OPEN list (i.e. \( b \) is infinite). When the value \( b \) is introduced the modified search algorithm is called the Limited-Expansion A algorithm (Sun, Cheng, and Fu 1994). In the case of the large Petri net model, and using \( w \) equal to 2, the effect of changing the value of \( b \) is illustrated in Tables 5.10 and 5.11. In the former the IF ordering strategy is utilized whereas in the latter IL strategy is applied.

It is evident from Table 5.10 that as the value of \( b \) is reduced the number of iterations is decreased and in return the achieved schedule completion time is increased. Several advantages of using the \( b \) parameter can be derived from Table 5.10. First it is obvious that the use of a maximum capacity equal to 50 and 20 leads to exactly the same schedule offered by the L1 algorithm. However, the generation time is reduced by approximately 58 seconds (a 51.79% reduction) when the maximum limit is used. Similarly, the use of a maximum capacity equal to 1 is equivalent to using a very high value of \( w \) such as 20 or 50. However, the generation time is reduced from 99 to 40 seconds which represents a reduction of 59.60%.
### Table 5.10 The effect of varying b in the case of IF ordering

<table>
<thead>
<tr>
<th>The maximum limit (b)</th>
<th>Total number of iterations</th>
<th>Depth of final marking</th>
<th>Generation time (sec)</th>
<th>Schedule makespan (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infinite</td>
<td>1121</td>
<td>800</td>
<td>112</td>
<td>634</td>
</tr>
<tr>
<td>50</td>
<td>1121</td>
<td>800</td>
<td>54</td>
<td>634</td>
</tr>
<tr>
<td>20</td>
<td>1121</td>
<td>800</td>
<td>53</td>
<td>634</td>
</tr>
<tr>
<td>10</td>
<td>1121</td>
<td>800</td>
<td>53</td>
<td>634</td>
</tr>
<tr>
<td>5</td>
<td>1069</td>
<td>800</td>
<td>42</td>
<td>635</td>
</tr>
<tr>
<td>3</td>
<td>964</td>
<td>800</td>
<td>40</td>
<td>698</td>
</tr>
<tr>
<td>2</td>
<td>947</td>
<td>800</td>
<td>40</td>
<td>712</td>
</tr>
<tr>
<td>1</td>
<td>800</td>
<td>800</td>
<td>40</td>
<td>986</td>
</tr>
</tbody>
</table>

### Table 5.11 The effect of varying b in the case of IL ordering

<table>
<thead>
<tr>
<th>The maximum limit (b)</th>
<th>Total number of iterations</th>
<th>Depth of final marking</th>
<th>Generation time (sec)</th>
<th>Schedule makespan (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infinite</td>
<td>1007</td>
<td>800</td>
<td>95</td>
<td>565</td>
</tr>
<tr>
<td>50</td>
<td>1007</td>
<td>800</td>
<td>49</td>
<td>565</td>
</tr>
<tr>
<td>20</td>
<td>1007</td>
<td>800</td>
<td>49</td>
<td>565</td>
</tr>
<tr>
<td>10</td>
<td>957</td>
<td>800</td>
<td>42</td>
<td>538</td>
</tr>
<tr>
<td>5</td>
<td>989</td>
<td>800</td>
<td>47</td>
<td>542</td>
</tr>
<tr>
<td>3</td>
<td>951</td>
<td>800</td>
<td>40</td>
<td>563</td>
</tr>
<tr>
<td>2</td>
<td>926</td>
<td>800</td>
<td>40</td>
<td>594</td>
</tr>
<tr>
<td>1</td>
<td>800</td>
<td>800</td>
<td>37</td>
<td>870</td>
</tr>
</tbody>
</table>
The reason for these reductions in the generation time is that the use of a maximum limit, b, helps in restraining the OPEN list from growing. In turn this reduces the complexity of the sorting and ordering operation that takes place whenever, a new marking is placed on the OPEN list. Thus the time required to perform a specific iteration is reduced.

On the other hand Table 5.11 offers different results with several exceptions. In the case of the iterations one finds that they follow the decreasing trend except for the trial where b is equal to 5. On the other hand the effect of decreasing b on the makespan is very unclear. However, if one neglects the first three readings, then the expected increasing trend will be restored. Similar to the previous table it is obvious that the use of a maximum capacity equal to 50 and 20 leads to exactly the same schedule offered by the L1 algorithm. However, the generation time is reduced by approximately 46 seconds (a 48.42% reduction) when the maximum limit is used. Accordingly, the use of a maximum capacity equal to 1 is equivalent to using a very high value of w such as 20 or 50. However, the generation time is reduced from 73 to 37 seconds which represents a reduction equal to 49.32%.

One recalls that in the previous section no results were reached when the weighting factor was equated to 1. The reason behind this was accredited to the limited memory resources. However, with the use of the maximum capacity parameter, b, one can achieve results. The following two trials presented in Table 5.12 were conducted on the reduced Petri net model. The ordering strategy used was the IL strategy which was proven to provide better results. It can be seen that both trials offer better schedules than those that have been presented earlier. However, it is obvious from the number of iterations that have been conducted that the use of w equal to 1 results in an exhaustive search of the Petri net reachability graph. Moreover, the increase of b from 5 to 10 resulted in a drastic (exponential) increase in the number of iterations and the required generation time. Actually the increase in the generation time was exactly 4 minutes and only resulted in the decrease of the total schedule by 8 minutes.
Table 5.12 The results of using $b$ with $w=1$ in the case of the reduced Petri net

<table>
<thead>
<tr>
<th>The maximum limit ($b$)</th>
<th>Total number of iterations</th>
<th>Depth of final marking</th>
<th>Generation time (sec)</th>
<th>Schedule makespan (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1144</td>
<td>400</td>
<td>47</td>
<td>395</td>
</tr>
<tr>
<td>10</td>
<td>3359</td>
<td>400</td>
<td>287</td>
<td>387</td>
</tr>
</tbody>
</table>

For this reason one recommends the use of either low values of $w$ in the case of the L1 algorithm, or a combination of the weighting factor (preferably equal to one) and an appropriate value of the maximum OPEN list capacity parameter, $b$, in the Limited-Expansion A algorithm. However, one has to keep in mind the size of the Petri net model under consideration.

5.3 Case Study 2

The second case study is also taken from the work of Lee and DiCesare (1994a). The scheduling problem is defined using Tables 5.13 and 5.14. However it should be noted that the configuration of this FMS will be changed in order to reflect the effect of adding additional components and constraints to the system. It is obvious that the scheduling problem under consideration is very small and simple since it is composed of three machines and two jobs.

Table 5.13 The process plan for Case Study 2

<table>
<thead>
<tr>
<th>Process No.</th>
<th>Job / Product Type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>J1 (1)</td>
</tr>
<tr>
<td>1</td>
<td>M1 / M2</td>
</tr>
<tr>
<td>2</td>
<td>M2 / M3</td>
</tr>
<tr>
<td></td>
<td>J2 (1)</td>
</tr>
<tr>
<td></td>
<td>M1 / M3</td>
</tr>
<tr>
<td></td>
<td>M1 / M2 / M3</td>
</tr>
</tbody>
</table>
Table 5.14 The FMS configuration for Case Study 2

<table>
<thead>
<tr>
<th>Components/Resources</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer Capacity</td>
<td>Infinity (The buffer constraint is neglected)</td>
</tr>
<tr>
<td>AGVs</td>
<td>None (The transportation constraint is not considered)</td>
</tr>
<tr>
<td>AGVS</td>
<td>Does not apply</td>
</tr>
<tr>
<td>AGVS Layout</td>
<td>Does not apply</td>
</tr>
<tr>
<td>Robots</td>
<td>Does not apply</td>
</tr>
<tr>
<td>Machines in Cells</td>
<td>None</td>
</tr>
</tbody>
</table>

The corresponding machining operation times are given in Table 5.15. The first scheduling problem is called Case Study 2.

Table 5.15 The operation times for Case Study 2

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>O_{1,1,1}</td>
<td>3</td>
</tr>
<tr>
<td>O_{1,1,2}</td>
<td>4</td>
</tr>
<tr>
<td>O_{1,2,2}</td>
<td>3</td>
</tr>
<tr>
<td>O_{1,2,3}</td>
<td>2</td>
</tr>
<tr>
<td>O_{2,1,1}</td>
<td>4</td>
</tr>
<tr>
<td>O_{2,1,2}</td>
<td>2</td>
</tr>
<tr>
<td>O_{2,2,1}</td>
<td>3</td>
</tr>
<tr>
<td>O_{2,2,2}</td>
<td>4</td>
</tr>
<tr>
<td>O_{2,2,3}</td>
<td>4</td>
</tr>
</tbody>
</table>
CHAPTER 5

THE SENSITIVITY ANALYSIS

Using the proposed machining module, the Petri net model representing this system is found to be composed of 36 places and 36 transitions. Several trial runs were conducted with different values of the involved parameters. The results are given in the following tables.

Table 5.16 The results for Case Study 2 using different values of $w$ and IF

<table>
<thead>
<tr>
<th>The weighting factor (w)</th>
<th>Total number of iterations</th>
<th>Depth of final marking</th>
<th>Generation time (sec)</th>
<th>Schedule makespan (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>17</td>
<td>16</td>
<td>Less than 1</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>16</td>
<td>16</td>
<td>Less than 1</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td>16</td>
<td>Less than 1</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>16</td>
<td>Less than 1</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>16</td>
<td>16</td>
<td>Less than 1</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 5.17 The results for Case Study 2 using different values of $w$ and IL

<table>
<thead>
<tr>
<th>The weighting factor (w)</th>
<th>Total number of iterations</th>
<th>Depth of final marking</th>
<th>Generation time (sec)</th>
<th>Schedule makespan (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>19</td>
<td>16</td>
<td>Less than 1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>17</td>
<td>16</td>
<td>Less than 1</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>17</td>
<td>16</td>
<td>Less than 1</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>16</td>
<td>Less than 1</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>16</td>
<td>16</td>
<td>Less than 1</td>
<td>10</td>
</tr>
</tbody>
</table>
Table 5.18 The results for Case Study 2 using different values of $b$

<table>
<thead>
<tr>
<th>The maximum limit (b)</th>
<th>Total number of iterations</th>
<th>Depth of final marking</th>
<th>Generation time (sec)</th>
<th>Schedule makespan (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infinite</td>
<td>17</td>
<td>16</td>
<td>Less than 1</td>
<td>7</td>
</tr>
<tr>
<td>10</td>
<td>17</td>
<td>16</td>
<td>Less than 1</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>17</td>
<td>16</td>
<td>Less than 1</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>17</td>
<td>16</td>
<td>Less than 1</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>16</td>
<td>16</td>
<td>Less than 1</td>
<td>8</td>
</tr>
</tbody>
</table>

5.3.1 The Effect of Adding a Distributed AGVS and Buffers

The scheduling problem considered in Case Study 2 is extended to include AGVs and buffers. The process plan remained unchanged, however the lot sizes of jobs 1 and 2 are increased to 5 each. The FMS configuration is given in Table 5.19.

Table 5.19 The FMS configuration for Case Study 2.1

<table>
<thead>
<tr>
<th>Components/Resources</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer Capacity</td>
<td>2</td>
</tr>
<tr>
<td>AGVs</td>
<td>Included</td>
</tr>
<tr>
<td>AGVS</td>
<td>Distributed AGVS with 2 AGVs in the input area</td>
</tr>
<tr>
<td>AGVS Layout</td>
<td>Layout type 1</td>
</tr>
<tr>
<td>Robots</td>
<td>Not included</td>
</tr>
<tr>
<td>Machines in Cells</td>
<td>None</td>
</tr>
</tbody>
</table>
CHAPTER 5

THE SENSITIVITY ANALYSIS

All transportation times are assumed to be equal to 1. The additions made to the FMS caused an increase in the model size. The new Petri net model is composed of 82 places and 75 transitions. The effect of varying the value of \( w \) is given in Table 5.20.

Table 5.20 The results for Case Study 2.1 using different values of \( w \)

<table>
<thead>
<tr>
<th>The weighting factor (( w ))</th>
<th>Total number of iterations</th>
<th>Depth of final marking</th>
<th>Generation time (sec)</th>
<th>Schedule makespan (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>138</td>
<td>130</td>
<td>2</td>
<td>38</td>
</tr>
<tr>
<td>2</td>
<td>130</td>
<td>130</td>
<td>2</td>
<td>42</td>
</tr>
<tr>
<td>10</td>
<td>130</td>
<td>130</td>
<td>2</td>
<td>42</td>
</tr>
</tbody>
</table>

For \( b = \infty \) and the IF ordering strategy

<table>
<thead>
<tr>
<th>The weighting factor (( w ))</th>
<th>Total number of iterations</th>
<th>Depth of final marking</th>
<th>Generation time (sec)</th>
<th>Schedule makespan (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>148</td>
<td>130</td>
<td>2</td>
<td>39</td>
</tr>
<tr>
<td>2</td>
<td>132</td>
<td>130</td>
<td>2</td>
<td>45</td>
</tr>
<tr>
<td>10</td>
<td>130</td>
<td>130</td>
<td>2</td>
<td>52</td>
</tr>
</tbody>
</table>

For \( b = \infty \) and the IL ordering strategy

It should be noted that in this particular problem the IF ordering strategy renders better results that the IL strategy, as opposed to the problem considered in Case Study 1. In addition, it was observed that varying the value of \( b \) in this case offers no significant changes.

5.3.2 The Effect of Adding a Centralized AGVS and Buffers

Instead of adding a distributed AGVS to the FMS configuration considered in Case Study 2, one decides to use a centralized AGVS. The FMS new configuration is given in Table 5.21. All transportation times are assumed to be equal to one.
Table 5.21 The FMS configuration for Case Study 2.2

<table>
<thead>
<tr>
<th>Components/Resources</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffer Capacity</td>
<td>2</td>
</tr>
<tr>
<td>AGVs</td>
<td>Included</td>
</tr>
<tr>
<td>AGVS</td>
<td>Centralized AGVS with 5 AGVs in the input area</td>
</tr>
<tr>
<td>AGVS Layout</td>
<td>Layout type 1</td>
</tr>
<tr>
<td>Robots</td>
<td>Not included</td>
</tr>
<tr>
<td>Machines in Cells</td>
<td>None</td>
</tr>
</tbody>
</table>

The Petri net model representing this system is composed of 74 places and 34 transitions. The effect of varying the value of \( w \) is given in Table 5.22.

Table 5.22 The results for Case Study 2.2 using different values of \( w \)

<table>
<thead>
<tr>
<th>The weighting factor (( w ))</th>
<th>Total number of iterations</th>
<th>Depth of final marking</th>
<th>Generation time (sec)</th>
<th>Schedule makespan (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>105</td>
<td>100</td>
<td>1</td>
<td>34</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>100</td>
<td>2</td>
<td>38</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>100</td>
<td>2</td>
<td>38</td>
</tr>
</tbody>
</table>

For \( b = \infty \) and the IL ordering strategy

<table>
<thead>
<tr>
<th>The weighting factor (( w ))</th>
<th>Total number of iterations</th>
<th>Depth of final marking</th>
<th>Generation time (sec)</th>
<th>Schedule makespan (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>114</td>
<td>100</td>
<td>1</td>
<td>33</td>
</tr>
<tr>
<td>2</td>
<td>101</td>
<td>100</td>
<td>1</td>
<td>41</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>100</td>
<td>1</td>
<td>44</td>
</tr>
</tbody>
</table>
CHAPTER 5

THE SENSITIVITY ANALYSIS

It is obvious that in this case using the centralized AGVS is more beneficial on terms of minimizing the schedule makespan than the distributed AGVS. Thus the developed scheduling technique could also be used as assisting tool in the design evaluation stage of an FMS.

5.4 The Use of Colored Petri Nets

In this research it is proposed to use colored Petri nets instead of timed-place Petri nets, during the modeling stage. The scheduling problem given in Case Study 2 is used as the problem to be modeled using both techniques. The heuristic search algorithm was run on a Pentium-133 PC which had 32 megabytes of RAM and the platform used was LINUX. It was found that both scheduling techniques reached the same solution using the same number of iterations. Obviously, this was expected since the same heuristic search algorithm was utilized. However, the difference between the two scheduling approaches was found to be in the generation time. The different values of \( w \) that were used along with the corresponding generation times for both Petri net models are given in Table 5.23.

<table>
<thead>
<tr>
<th>The weighting factor ((w))</th>
<th>The Generation time for the TPPN Petri net model (seconds)</th>
<th>The Generation time for the CPN Petri net model (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.074</td>
<td>0.00339</td>
</tr>
<tr>
<td>2</td>
<td>0.061</td>
<td>0.00276</td>
</tr>
<tr>
<td>3</td>
<td>0.0617</td>
<td>0.0027</td>
</tr>
<tr>
<td>5</td>
<td>0.0617</td>
<td>0.0027</td>
</tr>
<tr>
<td>10</td>
<td>0.0617</td>
<td>0.0027</td>
</tr>
<tr>
<td>20</td>
<td>0.0617</td>
<td>0.0027</td>
</tr>
<tr>
<td>50</td>
<td>0.0617</td>
<td>0.0027</td>
</tr>
</tbody>
</table>

Table 5.23 The comparison between TPPN and CPN
The results obtained in Table 5.22 are very promising. For all the experimented values of $w$ the scheduling technique employing the CPN model was found to be approximately 22 times faster than the approach utilizing TPPN. Nevertheless, one has to perform this comparison on larger models to confirm the fact that CPN models assist in reaching a final schedule in a shorter period of time. Another factor which was discovered through the implementation of the CPN model is the memory allocation. The problem of memory allocation is apparent in the construction of the CPN model since a lot of information has to be saved for each token (i.e. part) involved in the scheduling problem. This continuous process of information storage and retrieval obviously affects the efficiency of the model. Therefore, the number of tokens involved in the Petri net model will definitely affect the effectiveness of the CPN scheduling approach. However, one has to admit that the results obtained from Table 5.22 indicate that there could be some potential and benefits to be gained from the usage of CPN as an alternative modeling tool in the proposed scheduling approach.
Chapter 6
CONCLUSIONS AND RECOMMENDATIONS

Several important conclusions have been drawn from this research and the corresponding results that were calculated. In addition numerous recommendations are made to direct the future research in this field.

6.1 The Conclusions

1. One of the main conclusions reached by this research is that using different ordering strategies of equal-cost markings in the OPEN list, will definitely lead to different solutions. A certain scheduling problem that was previously treated and solved using a specific ordering strategy was used to demonstrate and prove this hypothesis. After using the other ordering strategy one found that better schedules were obtained, and in some cases these schedules were even reached in a shorter period of time. Thus the heuristic search algorithm employed in the proposed scheduling approach should be extended to include both the Insert First (IF) ordering strategy and the Insert Last (IL) ordering strategy.

2. Another important conclusion that was reached through this research is that additional places and transitions (whether required for specific purposes or redundant) have a severely negative impact on the effectiveness of the heuristic search algorithm. This is due to the fact that the very essence of the heuristic function relies on the depth of the marking in the reachability graph. Therefore, adding extra unnecessary places and transitions obviously misguides the heuristic search algorithm. In addition these redundancies affect the role and relative weight of the amount of time that has been spent in order to reach any particular marking (i.e. g(M)). Thus, it has been proven that using redundancies in the Petri net model affects the heuristic search algorithm. Although these additional places and transitions do not affect the truth or the accuracy of the modeling representation, yet they do mislead the search algorithm and cause it reach low quality solutions (i.e. 164
schedules which are far from the optimum). Thus, any additional places or transitions should be removed in order to enable the heuristic search algorithm to reach a reasonable solution in a reasonable amount of time.

3. The value of the weighting factor \( w \) plays a pivotal role in the direction of the heuristic search function. Therefore one should try to choose the best value of \( w \) which provides a reasonable compromise between the required generation time and the quality of the solution (i.e. the makespan of the schedule). Due to the advances in the computing power of microprocessors, it is more feasible now to choose small values of \( w \) which provide a favorable solution.

4. The maximum capacity of the OPEN list, \( b \), is a very important parameter in determining the direction of the search and more importantly in limiting the required memory and computer resources. It was found that setting the value of \( b \) to 1 is identical to running the L1 algorithm with a very high value of \( w \). Moreover, applying the former helps the heuristic search in reaching the same solution in a shorter period of time. Thus if one wants to get a quick solution, regardless of the quality of the final schedule, then it is recommended to select the value of \( b \) as 1 in the Limited-Expansion A algorithm, instead of running the L1 algorithm with a high value of \( w \).

5. The combination of the two parameters \( b \) and \( w \), can help the heuristic search algorithm in reaching extremely favorable solutions. This is due to the fact that in some cases (e.g. large Petri net models) where low values of \( w \) (e.g. \( w = 1 \)) are utilized, the L1 algorithm usually causes memory thrashing due to the fact that the OPEN list increases drastically. However, if one adds an appropriate value of the parameter \( b \), this problem is solved and the search algorithm manages to get an acceptable schedule. Obviously, the higher the value of \( b \), the better the result that is reached by the search algorithm. However, this value is constrained by the available computer and memory resources.

6. The proposed scheduling algorithm can be used as an assisting tool in the design evaluation stage of an FMS. Different FMS configurations with the expected batch and job requirements (based on averages) can be considered as different
scheduling problems. After constructing the necessary Petri net models and applying the proposed heuristic scheduling approach, the FMS configuration which offers the best schedule (i.e. minimum makespan) should be selected. However, this selection will be based on the criterion of maximizing the production rate (i.e. minimizing the makespan). If other criteria are also considered, then the available scheduling approach will offer no direct assistance.

7. Finally, the preliminary results achieved from the application of CPN models in conjunction with the proposed heuristic search algorithm, can be considered to be very promising. Therefore this new field should be profoundly explored since it could offer the advantage of generating schedules in a shorter period of time.

8. The proposed machining module for the case where buffer capacities are neglected, contains additional places and transitions. Therefore, this particular module should be modified by removing the extra places and transitions, in order to prevent it from hindering or misleading the heuristic search algorithm. However, this modification should only take place in the case that the transportation constraint is not considered, since these places have an impact on the logic of the model in the case of scheduling problems where AGVs are utilized.

9. The combination of Petri net modeling and heuristic graph-searching, offers a very powerful scheduling tool for FMSs. The Petri net model captures several important characteristics of FMSs such as shared resources, routing flexibility, concurrency, precedence constraints and mutual exclusions. In addition the integrated scheduling of AGVs and jobs is performed easily without having to add any more rules.

6.2 The Recommendations

Several recommendations are offered with regards to the future research that could be conducted in this field :-

166
1. In this research several assumptions were made to relax the Petri net model. Thus several constraints were neglected such as the tool-competition constraint, and the set-up times. In addition several constraints were imposed on the system, such as only considering machining and transportation operations and neglecting the assembly processes. In addition several fixed layout settings were imposed on the FMS when AGVs were considered. All these assumptions limit the flexibility of the model and should be included in the future research.

2. Another important recommendation is the inclusion of probabilistic functions to allow for variations in the machining and transportation operations. In this research all the involved operations were assumed to be constant. Obviously this is not a true reflection of real-life manufacturing systems.

3. Probabilities should be associated with the machine status. In other words a machine could be either functioning or down. Thus machine failure could be included in the Petri net model. Obviously variable machine repair time should also be considered.

4. In this research the scheduling criterion was minimizing the makespan. Other heuristic functions should be experimented and they should be able to account for other scheduling criteria (such as meeting due-dates). In addition research could be targeted at developing a heuristic function which satisfies the admissibility condition, which would ensure that the optimal schedule is reached.

5. The substitution of the TPPN model with the CPN model should be investigated. The use of a CPN model with the heuristic search algorithm seems to be a promising field which requires extensive research.


168


APPENDIX A: A BRIEF INTRODUCTION TO PETRI NETS

An Introduction to Petri Nets

The concept of Petri nets was proposed by Carl Adam Petri in his doctoral presentation in Germany in 1962. C.A. Petri used his new graphical/mathematical tool for modeling assistance in the field of automata. However, over the past 35 years Petri nets have grown drastically in their application domain. This fact is elaborated in the following section, Appendix 2. Moreover, this study as a whole could be used to illustrate the diversity of Petri net applications since it deals with both the modeling and scheduling of flexible manufacturing systems.

In this appendix some basic concepts are introduced which constitute the fundamentals on which Petri nets are constructed and developed. Thus, this appendix is a simple and brief introduction to the concept of Petri nets. It is believed that examining this appendix is essential for easily understanding this study, since any further reading in the main body of this work will be fruitless if one lacks prior knowledge of the field of Petri nets. This appendix offers a definition of Petri nets in addition to the graphical and mathematical representation used to describe these nets. This is followed by a description of how a Petri net evolves (i.e. changes its marking) and how this process is depicted graphically and mathematically. Finally, a very brief reference to the different types of Petri nets is given.

A Definition of Petri Nets

A Petri net can be described verbally as a weighted-bipartite directed graph. Petri nets are composed of two types of objects which are namely, places and transitions; therefore the term "bipartite" was used. In addition "directed" arcs are used to link places to transitions and transitions to places. These arcs can carry a certain weight, and this is the reason for adding the prefix "weighted" to the previous definition of Petri nets. Pictorially, places are depicted as circles whereas transitions are
represented as bars or boxes. In the terminology of Petri nets a place is considered as an input place to a transition if a directed arc connects this place to the transition. Similarly, a place is considered as an output place of a transition if a directed arc connects this transition to that place. Figure A.1 provides two identical examples of a Petri net structure. It can be seen from Fig. A.1(a) and A.1(b) that parallel arcs connecting places to transitions or transitions to places may be represented by a single directed arc carrying a weight equal to the corresponding number of parallel arcs. This feature is simply used to increase the compactness of the graphical representation of the model. Arcs carrying no weight label are assumed to have unity weight. Moreover, applying the Petri net terminology mentioned earlier, one can see that places \( p_1 \) and \( p_2 \) are input places to transition \( t_1 \), whereas places \( p_3 \) and \( p_4 \) are output places of \( t_1 \). A final remark to be made about Fig. A.1 is that it depicts the two forms by which a transition could be drawn; namely, in part (a) the transition was represented as a box whereas in part (b) it was represented as a bar.

![Figure A.1 Petri net modeling compactness](image)

The description provided above has only considered the static portion of a Petri net, or as stated by Silva (DiCesare et al. 1993) the "net structure". The net structure, as such, is of no value since it cannot be used to model the state nor the evolution of a discrete event dynamic system (DEDS), which is in our case a flexible manufacturing system. Thus, one needs to take into account the section of a Petri net which helps in...
the dynamic modeling and analysis of the system under consideration. Therefore in order to study the dynamic behavior of the modeled system, in terms of its different states and their changes, a marking is integrated with the Petri net structure. A marking of a Petri net defines the present state of the modeled system under consideration. This is accomplished by assigning a non-negative number of tokens to each place in the Petri net. Thus any place can potentially hold either none or a positive number of tokens, which are pictured as small black dots. Depending on what a place is supposed to represent the presence or absence of tokens indicates a certain fact. For example, if a place represented a certain condition, then the presence or absence of tokens signifies whether this condition is true or false, respectively. Another example is when a place represents the availability of a resource (e.g. machines). The presence of k-tokens in this place means that k resources (machines) are available, and accordingly the absence of tokens states that no resources (machines) are currently available. Thus one can regard an initial marking of a Petri net as a specification of the initial distribution of tokens in the corresponding places, or in other words, the initial state of the system. An example of a marked Petri net is given below in Fig. A.2.

![Figure A.2 A marked Petri net](image-url)

Up to this point, the definitions provided for Petri nets and their marking can be classified as informal or verbal. However, one believes that starting with an informal explanation of the concept on hand, would make the mathematical definition and
notation to be used later on a lot easier to visualize and conceive. Thus, after completing the informal description during the previous stage one can formally define Petri nets as follows. A Petri net is a five-tuple:

\[ \text{PN} = (P, T, I, O, M_0) \]  \hspace{1cm} \text{(Equation App1.1)}

- \( P = \{ p_1, p_2, \ldots, p_m \} \) is a finite set of places
- \( T = \{ t_1, t_2, \ldots, t_m \} \) is a finite set of transitions where \( P \cup T \neq \emptyset \), and \( P \cap T = \emptyset \)
- \( I : (P \times T) \rightarrow \mathbb{N} \) is an input function that defines directed arcs from places to transitions, where \( \mathbb{N} \) is a set of non-negative integers
- \( O : (P \times T) \rightarrow \mathbb{N} \) is an output function that defines directed arcs from transitions to places
- \( M_0 : P \rightarrow \mathbb{N} \) is the initial marking

It should be noted that the initial marking or any subsequent marking of a Petri net, is represented by an \((m \times 1)\) column vector, \( M \). The letter \( m \) being the total number of places in the Petri net. In addition, the \( p^{th} \) element of \( M \), denoted as \( M(p) \) is a non-negative number which represents the number of tokens in the place \( p \). Another point to be clarified is that if \( I(p,t) = k \), then this indicates that there are \( k \) directed arcs connecting place \( p \) to transition \( t \). Similarly if \( O(p,t) = k \), then there exists \( k \) directed arcs heading from transition \( t \) to place \( p \). Accordingly if \( I(p,t) \) or \( O(p,t) = 0 \) then there are no arcs from \( p \) to \( t \) or from \( t \) to \( p \), respectively. The previous definitions are illustrated using the simple Petri net in Fig. A.3 along with Fig. A.4 that includes the corresponding input and output functions in a matrix form, in addition to the vector \( M \) defining the marking of the Petri net.

![Figure A.3 An initial marking of a Petri net](image-url)
At this point one can discuss the other form of representation of Petri nets. Up to this stage one has only been considering and portraying Petri nets in the graphical form. Obviously, the graphical representation offers Petri nets a great advantage and simplifies the process of understanding and communicating information through it. However, Petri nets can also be represented in matrix form, which will be very useful and important, as will be shown later on. The matrix which represents a Petri net is called an incidence matrix. This incidence matrix defines all possible interconnections between places and transitions in a Petri net. The incidence matrix of a Petri net is an integer, $n \times m$, matrix $A$, where $n$ is the number of transitions, and $m$ is the number of places. The entries of the incidence matrix are defined as follows:

$$a_{ij} = a_{ij}^+ - a_{ij}^-; \text{ where}$$

(Equation App1.2)

$$a_{ij}^+ = \#(p_i, t_i) \text{ is the number of arcs connecting transition } t_i \text{ to its output place } p_j$$

$$a_{ij}^- = \#(p_i, t_i) \text{ is the number of arcs connecting transition } t_i \text{ to its input place } p_j$$
The incidence matrix $A$ of the Petri net provided in Fig. A.3 is given below in Fig. A.5.

$$A = O - I =$$

$$\begin{bmatrix}
    t_1 & -1 & -3 & 2 & 1 & 0 & 0 \\
    t_2 & 0 & 0 & 0 & -1 & 1 & 0 \\
    t_3 & 0 & 0 & -1 & 0 & 0 & 2 \\
    t_4 & 0 & 0 & -2 & 0 & -1 & 1 \\
\end{bmatrix}$$

Figure A.5 The incidence matrix

It remains to be said that the previous definition of Petri nets is the one most commonly used in the literature concerning this topic. However, some introductory books and papers tend to consider Petri nets as a combination of a Petri net structure (i.e. $Z = \{P, T, I, O\}$) and an initial marking $M_0$. Thus a Petri net, PN, with an initial marking can be defined as a two tuple $(Z, M_0)$. Nevertheless, there is no contradiction between both definitions since both lead to the same end result.

The Rules of Petri Nets

In his award-winning tutorial paper on Petri nets, Murata (1989) states that there is only one rule which one needs to know regarding Petri net theory. He continues by claiming that although this rule appears very simple, yet its application in Petri nets is very critical and its implications are rather complex. This rule is namely, the rule of transition enabling and firing. Several other sources refer to this rule as the “token game” since this rule governs the flow of tokens through the Petri net structure. Thus one can think of this rule as the mechanism which provides the means of changing the distribution of tokens among the different places of the Petri net model. Obviously, this change reflects the occurrence of events or the execution of operations, and ultimately enables one to study the dynamic behavior of the modeled system.

However, in this study a minor modification will be introduced in the method of describing this rule. One tends to find the manner in which Zurawski and Zhou (1994) tackled this issue rather more logical and clear. In their tutorial paper on Petri nets,
they introduced the token game in the form of two rules which were the Enabling Rule and the Firing Rule. Through dividing this so-called “one-and-only” rule into two parts one can think of them as two different concepts, which actually is true. Thus, one needs to understand two principles which in fact constitute the rule that was stated by Murata. Thereby, we begin by defining the Enabling Rule. This rule states that a transition $t$ is said to be enabled if each input place $p$ of $t$ contains at least the number of tokens equal to the weight of the directed arc connecting $p$ to $t$. This condition could be denoted mathematically as follows:

\[ \text{Transition } t \text{ is enabled iff } M(p) \geq l(p, t) \forall p \in P \]  

(Equation App1.3)

The second rule which governs the flow of tokens is the Firing Rule. This rule states that an enabled transition $t$ may or may not fire depending on the additional interpretation (e.g. whether or not an event actually takes place). However, in the case that an enabled transition fires, this firing operation will result in the removal of tokens from each input place $p$ and the deposit of tokens in each output place $p$ equal to the weight of the directed arc connecting the transition $t$ to the specific place $p$. The Enabling and Firing rules are clearly illustrated in Fig. A.6. In Fig. A.6(a) the transition $t_1$ is enabled and in Fig. A.6(b) we can see the result of the firing of transition $t_1$.

![Diagram](image)

(a) Transition $t_1$, enabled

(b) Transition $t_1$, fires

Figure A.6 The Petri net Enabling and Firing Rules
The State Equation

After the graphical representation of the firing mechanism involved in Petri nets, one needs to consider the mathematical formulation of this phenomenon. This alternative approach will be based on matrix equations. The starting point from which to commence this study is the incidence matrix, A, which has been introduced and defined at an earlier stage. The incidence matrix is the basis of the representation of the dynamic behavior of Petri nets, in addition to their analytical analysis. However, the analytical analysis of Petri nets is beyond the scope of this introductory section, and therefore it will be limited to the utilization of the incidence matrix in representing the firing operation.

The entries $a_{ij}$ of the incidence matrix have been discussed earlier and were defined mathematically in Equation 2. However, upon studying these entries one can reach an interesting verbal definition. Each entry $a_{ij}$ actually represents the corresponding flow or change in the number of tokens in the place $p_i$ in the case that transition $t_j$ fires. This is evident from the fact that $a_{ij}$ indicates the number of tokens to be deposited in the output place $p_j$ when transition $t_j$ fires. Likewise, $a_{ij}$ specifies the number of tokens to be removed from place $p_i$ in the case that transition $t_j$ fires. Therefore, one can restate the Enabling rule in the following equation:

Transition $t_j$ is enabled iff $a_{ij} \leq M(p_j)$, $j=1,2,\ldots,m$; where (Equation App1.4)

$$M(p_j) = \text{The marking of place } p_j$$
$$m = \text{The total number of places}$$

Thereby, the incidence matrix can be used to depict the firing operation of a transition $t$. This mathematical representation takes the form of a matrix equation, which is known as the state equation. Thus, the state equation denotes the change in the marking (distribution of the tokens) of a Petri net caused by the firing of a transition $t$. As a result, the state equation depicts the dynamic behavior of the modeled system.
The state equation is defined as follows:

\[ M_k = M_{k-1} + A^T \cdot u_k \]  

(Equation App1.5)

\[ M_k \] = The k-th Petri net marking  
\[ A^T \] = Transpose of the incidence matrix  
\[ u_k \] = The k-th firing vector

\( M_k \) is an \( m \times 1 \) column vector representing a marking \( M_k \) that is reached after firing the enabled transition \( t_i \) in the previous marking \( M_{k-1} \). The k-th firing vector \( u_k \) is an \( n \times 1 \) column vector which has only one non-zero entry. This entry, which is a 1, is in the \( i \)-th place denoting that the transition \( t_i \) will fire in the k-th firing of the Petri net transition firing sequence that started from \( M_0 \) (e.g. \( t_1 \cdot t_4 \cdot t_5 \cdot t_2 \cdot t_3 \)). Thus, the state equation presents the resulting marking (new state of the system) given that a transition fires (an event takes place) in the existing marking (current state).

The Different Types of Petri Nets

After completing this brief introduction of the basic fundamentals of Petri nets in both the graphical and mathematical form, one needs to give a quick overview of the available classes of Petri nets along with an indication of the specific types of Petri nets that will be adopted and utilized in this study. Due to the wide diversity of the application-domain of the topic and the numerous modifications and variations that have been incurred on Petri nets to accommodate the different modeling needs, one can only mention some of the different types of Petri nets. In addition one can neglect the groups of Petri nets that are not relevant to this study. Therefore, this section will only introduce the types of Petri nets that are associated with this study.

Pure Petri nets is a term used to classify Petri nets that do not contain any self-loops. A self-loop occurs when a place \( p_i \) is both an input and an output place of a transition \( t_i \). The problem with a self-loop is that the incidence matrix cannot account for them (see them). This situation could be resolved by transforming the self-loop into a
normal loop, through the introduction of a dummy place and transition. An example for the elimination of a self-loop is given below. Figure A.7(a) depicts a self-loop between the pair \( p_i \) and \( t_i \), whereas Fig. A.7(b) portrays the Petri net after adding the dummy place \( p_d \) and the dummy transition \( t_d \).

(a) A self-loop

(b) Adding dummy place and transition

Figure A.7 Self-loop elimination

The class of ordinary Petri nets is used to categorize the nets which do not contain any directed arcs carrying a weight greater than one. Thus all ordinary Petri nets have directed arcs with the weight of one (i.e. unlabeled). However, ordinary Petri nets do not include any conception of time which could be a critical factor in the system that is being modeled. Therefore ordinary Petri nets are only used to study the logical structure of the modeled system but can play no role in exploring the system’s evolution with respect to time. Moreover, regarding the problems of performance evaluation or scheduling (which is the main topic of this study), time is an essential factor which has to be taken into consideration.

Therefore, time was incorporated into Petri nets. However, time has been introduced into Petri nets in several different ways; although these various ways are all grouped together under the title of timed Petri nets (TPN) or Petri nets with time-extensions. The two fundamental categories of TPNs are the deterministic timed Petri nets and the stochastic timed Petri nets. This study deals with the class of deterministic timed Petri nets, which in turn is divided into two main groups. Those two main groups are, namely, the timed place Petri nets (TPPN) and the timed transition Petri nets (TTPN). The difference between the two types is that in the former time delays are
associated with places whereas in the latter time is linked to the transitions. In this study, timed place Petri nets (TPPNs) will be used to model and schedule the flexible manufacturing systems under consideration.

However, there still remains the main problem that hinders the use of Petri net as modeling tool. This problem is that using ordinary Petri nets (with or without time extensions) to model complex systems is usually a tedious task, due to the large size of the resulting Petri net model. Moreover, the huge size of the final Petri net model increases the difficulty of analyzing and validating the model. For this purpose, high-level Petri nets (HLPN) were developed in the late 1970s and early 1980s. HLPNs help in constructing more compact Petri net models that incorporate the feature of information transmission. The two main types of HLPNs are the predicate/transition nets (PrT-nets) and the colored Petri nets (CPNs). In this study, CPNs will be used in addition to the TPPNs as the modeling tool. CPNs were originally introduced by Kurt Jensen in his Ph.D. thesis that was published in 1980. The main difference between CPNs and ordinary Petri nets is that in CPNs the tokens are identifiable and carry information (called colored tokens). In addition formal expression are attached to the arcs of the Petri net model (called inscriptions) in order to constrain and specify the flow of tokens through the transition firing process. The information attached to the tokens is usually altered and modified when a transition is fired. All these features enable CPNs to combine and group several similar subnets into a single net. Thus CPNs can be used to construct more compact and concise models than ordinary Petri nets. Nevertheless, this compactness of the final model is compensated for or counterbalanced by the more complex inscriptions attached to the arcs of the net.

Finally, one needs to stress on the fact that CPNs have the same modeling powers of ordinary Petri nets. As mentioned earlier, the main advantage of CPNs is that their formalism assists in constructing more compact Petri net models which in turn help in simplifying the analysis process.
APPENDIX B: A HISTORICAL OVERVIEW OF PETRI NETS

The historical development of Petri Nets and their applications

The Ph.D. dissertation which was delivered by Carl Adam Petri in 1962 to the faculty of Mathematics and Physics at the Technical University of Darmstadt, West Germany; along with his work that took place during his stay at the University of Bonn was the starting point for all the advances that were to occur in the field of Petri nets. Carl Petri’s work came under the hand of A.W. Holt, who later led the Information System Theory Project of Applied Data Research, Inc., in the United States. The early developments and applications of Petri nets are found in the reports associated with this project, and in the Record of the 1970 Project MAC Conference on Concurrent Systems and Parallel Computation (Murata 1989).

The baton for leading the research in Petri nets and their applications was then handed to the Computation Structure Group at MIT. From 1970 to 1975 this Group was the most active in studying Petri nets and it produced several papers and theses related to Petri nets and their applications.

Since the late 1970s, the Europeans have been very active in organizing workshops, advanced courses and publishing conference proceedings on Petri nets. In October 1979, about 135 researchers mostly from European countries assembled in Hamburg, West Germany, for a two-week advanced course on General Net Theory of Processes and Systems (Murata 1989). The proceedings of this meeting was published, and included all 17 lectures which were delivered during the course. The second Advanced Course was held in Bad Honnef, West Germany, in September 1986. The proceedings of this course included 34 articles including two articles by Carl Adam

---

1 The following historical overview of Petri nets is given in accordance with the paper of Murata (1989).
Petri in which he discussed his axioms of concurrency theory and his suggestions for future research. The third course took place in Dagstuhl, Germany, in October 1996. The material delivered in these Advanced Courses serve as a presentation of the latest progress in the field of Petri nets. The content of all three courses are published by Springer-Verlag under the title of "Lecture Notes in Computer Science".

This increased interest in the field of Petri nets led to the first "European Workshop on Application and Theory of Petri Nets", that was held in 1980 in Strasbourg, France. Since then, the workshops have been held annually at different locations in Europe. In the year of 1989 this assembly (workshop) had its title changed to the "International Conference on Application and Theory of Petri Nets". This was due to the fact that this annual meeting became the main conference associated with the field of Petri nets. People from industry and academic fields (mainly universities), ranging from 150 to 200, would annually create this forum to discuss the latest achievements in both the application and theory of Petri nets, in addition to presenting any newly developed Petri net tools. The sites at which this conference took place are listed below in Table B.1. It should be noted that in 1993 the conference was held, for the first time, outside Europe, in the United States. This could be viewed as a recognition of the world-wide interest in the subject. It remains to be said that the 18th International Conference, Petri Nets '97, is scheduled to take place in Toulouse, France, during the period of June 23 till June 27, 1997.

It should be noted that the distribution of the proceedings of these workshops is usually limited to the participants. However, selected papers from these workshops and other relevant research articles have been published by Springer-Verlag in their series of books under the title of "Advances in Petri Nets". In the 1987 volume of "Advances in Petri Nets" a complete bibliography of all the previous publications related to Petri nets was made. It listed 2074 entries dated from 1962 till early 1987. In the 1991 volume, a similar bibliography of publications related to Petri nets was issued and it cited 4099 entries dealing with Petri net theory and applications. Obviously, this is an evident
indication that research in this topic is continuously taking place since the figure has almost doubled. The increase from 1987 till 1991 manifests the fact that studying Petri nets has boomed in the past 10 years.

Table B.1 Conference sites from 1980 to 1996

<table>
<thead>
<tr>
<th>Year</th>
<th>City</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>1980</td>
<td>Strasbourg</td>
<td>France</td>
</tr>
<tr>
<td>1981</td>
<td>Bad Honnef</td>
<td>West Germany</td>
</tr>
<tr>
<td>1982</td>
<td>Varenna</td>
<td>Italy</td>
</tr>
<tr>
<td>1983</td>
<td>Toulouse</td>
<td>France</td>
</tr>
<tr>
<td>1984</td>
<td>Aarhus</td>
<td>Denmark</td>
</tr>
<tr>
<td>1985</td>
<td>Espoo</td>
<td>Finland</td>
</tr>
<tr>
<td>1986</td>
<td>Oxford</td>
<td>UK</td>
</tr>
<tr>
<td>1987</td>
<td>Zaragoza</td>
<td>Spain</td>
</tr>
<tr>
<td>1988</td>
<td>Venice</td>
<td>Italy</td>
</tr>
<tr>
<td>1989</td>
<td>Bonn</td>
<td>West Germany</td>
</tr>
<tr>
<td>1990</td>
<td>Paris</td>
<td>France</td>
</tr>
<tr>
<td>1991</td>
<td>Aarhus</td>
<td>Denmark</td>
</tr>
<tr>
<td>1992</td>
<td>Sheffield</td>
<td>UK</td>
</tr>
<tr>
<td>1993</td>
<td>Chicago</td>
<td>USA</td>
</tr>
<tr>
<td>1994</td>
<td>Zaragoza</td>
<td>Spain</td>
</tr>
<tr>
<td>1995</td>
<td>Torino</td>
<td>Italy</td>
</tr>
<tr>
<td>1996</td>
<td>Osaka</td>
<td>Japan</td>
</tr>
</tbody>
</table>
In July 1985, another series of international workshops was initiated. These series placed emphasis on timed and stochastic nets; and was held under the title of the "International Workshop on Petri Nets and Performance Models". The first international workshop on timed Petri nets was held in Torino, Italy in July 1985. The second was held in Madison, Wisconsin, in August 1987; and the third in Kyoto, Japan, in December 1989. The fourth international workshop took place in Melbourne, Australia, in 1991; the fifth in Toulouse, France, in 1993; and the sixth in North Carolina, USA, in 1995. The seventh workshop, PNPM '97 is to be held in Saint Malo, France, during the period of June 2-6, 1997. Thus one can see the global interest in the advancements of timed and stochastic Petri nets and their applications in the design and performance evaluation of systems.

After this brief overview of Petri nets and their historical progression, one moves to the areas of application of Petri nets. A quick glance at the numerous topics and fields covered by Petri nets, will give an immediate indication of their power and enormous versatility. It shall also provides us with the undoubted notion that Petri nets have grown drastically in the sense of application-domain and research, ever since their introduction and conception by Carl Adam Petri in 1962.

Over the past 35 years or so, Petri nets have been used for several purposes. Obviously the main areas were modeling and analysis. Petri nets were used to model and verify real-time safety critical systems such as air-traffic control systems, rail-traffic control systems, nuclear reactor control systems, etc. In addition, fault detection and in-process monitoring were modeled and analyzed. However one of the most successful areas of application of Petri nets was the modeling and analysis of communication protocols. The work in this area started in the early 1970s and recently several approaches have been proposed for constructing Petri net models of protocols from specifications written in relatively skill-free languages. Methods were also proposed for transforming SDL, Lotos, and Estelle based protocol specifications into Petri nets for performance and reliability analysis (Zurawski and Zhou 1994).
APPENDIX B

Petri nets have also been used in software development. The work in this field was directed towards modeling and analysis of software systems using Petri nets. The most mature advancements involved the use of colored Petri nets which have been proven to be a useful language for the design, specification, simulation, validation and implementation of large software systems. The design and analysis of Ada systems using Petri nets have also attracted a considerable amount of attention (Zurawski and Zhou 1994).

Another successful area of application of Petri nets is the performance evaluation of the modeled systems. Petri nets incorporating time in their definition, namely Timed Petri nets (TPN), have been used in this field. The time factor installed in the Petri nets can follow either deterministic and/or probabilistic functions. The performance evaluation of deterministic timed Petri nets and stochastic timed Petri nets can be conducted using two different methods, which are using either analytical techniques based on solving the underlying (semi-) Markov processes, or discrete-event simulation techniques. However, there are cases where the complexity of the model render analytical techniques infeasible and therefore one has to resort to the discrete-event simulation alternative which is both an expensive and a time consuming technique. Thus using Petri nets with probabilistic time functions allows one to measure several performance measures such as production rates and percentage utilization of machines in manufacturing system models.

Another area of application of Petri nets is modeling and analyzing communication networks. Modeling of Fiber Optics Local Area Networks such as Expressnet, Fastnet, D-Net, U-Net, and Token Ring is one example. Fieldbuses, such as FIP and ISA-SP50, have attracted lots of attention since 1992, due to the fact that they are very important networks for factory automation. Moreover, the interest is growing in the area of modeling and evaluating High Speed Networks which are crucial for the successful development of Multimedia Systems (Zurawski and Zhou 1994).
Another success story is the application of Petri nets in the modeling of sequence controllers. Programmable Logic Controllers (PLCs) are commonly used for the sequence control in automated systems. They are designed using ladder logic diagrams, which are known to be very difficult to debug and modify; on the other hand, Petri net based sequence controllers can be easily designed, modified, implemented and maintained. In the early 1980s, Hitachi Ltd. developed a Petri net based sequence controller which was successfully used in real-life applications to control parts assembly systems, and automatic warehouse load/unload systems. The main advantage of Petri nets over PLCs is the reduced development time. In addition, in the past few years, numerous approaches were proposed for the synthesis and implementation of Petri net based sequence controllers.

Another major area of application of Petri nets, which is directly related to this study, is manufacturing systems. Petri nets have been used extensively in the modeling and the analysis of manufacturing systems. Petri nets were used to represent simple production lines with buffers, machine shops, automotive production systems, flexible manufacturing systems, automated assembly lines, resource-sharing systems, and recently just-in-time and kanban manufacturing systems.

Accordingly, Petri nets were also used for the performance evaluation of production systems, involving simple production lines, job shops, robotic assembly cells, flexible manufacturing systems, etc. (Zurawski and Zhou 1994). As mentioned earlier if a model is complex then a state explosion problem arises, and the stochastic model is not amenable for mathematical analysis. Thus simulation is performed for the analysis of both qualitative and quantitative properties, in addition to the performance evaluation. Another field of application which is directly related to this study is the use of Petri nets with time extensions, combined with heuristic search techniques to model and schedule manufacturing systems.
Most of the applications which have been mentioned above are related to industrial applications, due to the fact that they are associated with one's field of interest and with the subject of this research. However, this is by no means a complete overview of Petri nets and their applications which are definitely not restricted to the industrial field. This is apparent when one realizes that Petri nets have been used to model and analyze distributed-software systems, distributed-database systems, concurrent and parallel programs, multiprocessor memory systems, data-flow computing systems, asynchronous circuits and structures, compiler and operating systems, office information systems, management information systems, formal languages, logic programs, legal systems, human factors, neural networks, digital filters, and decision models (Murata 1989).

Through this brief overview of the historical developments of Petri nets and their application domain, one hopes to have managed to convey the notion of how important and powerful Petri nets are as a modeling, analysis and performance evaluation tool. It is clear that there is an increasing interest in this mathematical/graphical tool; which can be verified by the numerous fields in which Petri nets have been applied, whilst keeping in consideration that the formalism of Petri nets is a relatively modern concept since it originated in 1962.
APPENDIX C : THE COMPUTER PROGRAM

The computer program that was developed for this research is divided into two interconnected modules. The first module of the program receives the user's input data and constructs the corresponding Petri net model. The other module of the program performs the heuristic search algorithm. Thus the first section of the program creates the A-matrix and the initial and final markings, whereas the second section takes these matrices as its input and applies the heuristic search algorithms to reach the final schedule.

This appendix includes the pseudocode for both sections of the computer program. In addition, a listing of the heuristic search program (computer-code) is provided. As for the modeling section of the program, the reader should refer to the author if he/she wishes to see the actual computer code. It should be noted that the computer program was developed with the aid of Microsoft Visual C++ (version 4.2).

The Pseudocode of the Modeling Module

loop until end of operations
{
    if job of current operation is new
    {
        link machines of the last process of previous job to the final place;
    }
    create place of new job and link it to a new transition;
    if process of current operation = 1
    {
        if machine of current operation is new
        {

191
create new machine and save info. about it;
link new machine to current transition;
update current transition and current place;
}
else
{
  get info. about machine;
  create a machine that was used before;
  link it to current transition;
  link the places saved in info. to the machine that was created;
  update current transition and current place;
}
}
if process of current operation > 1
{
  if process of current operation is new
  {
    link machines of last process to intermediate place;
  }
  link intermediate place to current transition;
  if machine of current operation is new
  {
    create new machine and save info. about it;
    link new machine to current transition;
    update current transition and current place;
  }
  else
  {
    get info. about machine;
    create a machine that was used before;
link it to current transition;
link the places saved in info. to the machine that was created;
update current transition and current place;
}
}
}

link machines of last process in last job to the final place of last job;

In the case that AGVs and robots are included in the model:

loop until end of operations
{
    if job of current operation is new
    {
        link each machine of the last process of previous job to the appropriate AGV;
        link all AGVs of the last process of previous job to the final place of previous job;
    }
    create place of new job and link it to a new transition;
    if process of current operation = 1
    {
        link input area to appropriate AGV;
        if machine of current operation is new
        {
            create new machine and save info. about it;
            link new machine to current transition;
            update current transition and current place;
        }
    }

    }
else
{
    get info about machine;
    create a machine that was used before;
    link it to current transition;
    link the places saved in info. to the machine that was created;
    update current transition and current place;
}
}

if process of current operation > 1
{
    link each machine of previous process to the appropriate AGV or Robot;
    if machine of current operation is new
    {
        create new machine and save info. about it;
        link new machine to AGVs of previous process;
        update current transition and current place;
    }
    else
    {
        get info about machine;
        create a machine that was used before;
        link it to AGVs of previous process;
        link the places saved in info. to the machine that was created;
        update current transition and current place;
    }
}

link each machine of the last process of last job to the appropriate AGV;
link all AGVs of the last process of last job to the final place of last job;
The Pseudocode of the Heuristic Search Module

The modified A* algorithm:

- construct the topology matrix A
- construct the initial and final states
- construct the topology module
- construct the search engine
- insert the initial state into open list
- loop while (final state not found)
  - get best state in open list
  - find all enabled transitions due to the marking in this state
  - for each enabled transition
    - get the new state resulting from firing this transition
    - insert this state into open list
      - if the state is already there then check for the cost
        - if the cost of the new is better then insert it and delete the old one
        - else delete the new one
      - else insert it in order according to its cost
  - insert this state into closed list
    - if the state is already there in closed list
      - if the cost of the new is better then insert it into open again
      - else delete the new one
  - end loop
colored PN:
- construct the topology matrix A
- construct the initial and final states
- construct the topology module
- construct the search engine
- insert the initial state into open list
- loop while (final state not found)
  - get best state in open list
  - find all enabled transitions due to the marking in this state
    - for each color
    - for each place
  - for each enabled transition
    - get the new state resulting from firing this transition
      - fire the transition with respect to an input place and a color
      - construct the new state
    - insert this state into open list
      - if the state is already there then check for the cost
        - if the cost of the new is better then insert it and delete the old one
        - else delete the new one
      - else insert it in order according to its cost
    - insert this state into closed list
      - if the state is already there in closed list
        - if the cost of the new is better then insert it into open again
        - else delete the new one
  - end loop
The Computer Code of the Heuristic Search Program

// cAstar.h

#ifndef _cAstar
#define _cAstar

#include "cOpenList.h"
#include "cCloseList.h"
#include "cTopology.h"
#include "cState.h"

class cAstar
{
private:
    cState *initial_state;
    cState *final_state;
    cOpenList *open_list;
    cCloseList *close_list;
    cTopology *topology;

public:
    cAstar(cState *initial, cState *final, cTopology *the_topology);
    ~cAstar();
    void set_initial_state(cState *state);
    void set_final_state(cState *state);
    void set_topology(cTopology *the_topology);
    cState *activate_search();
};
#endif

// cAstar.cpp

#include "cAstar.h"

cAstar::cAstar(cState *initial, cState *final, cTopology *the_topology)
{
    set_final_state(final);
    set_initial_state(initial);
    cout<<"Final state dump..\n";
    final->dump();
    cout<<"Initial state dump..\n";
    initial->dump();
```cpp
set_topology(the_topology);
close_list=new cCloseList();
open_list=new cOpenList(close_list);
}
cAstar::~cAstar()
{
}

void cAstar::set_topology(cTopology *the_topology)
{
    topology=the_topology;
}

void cAstar::set_initial_state(cState *state)
{
    initial_state=state;
}

void cAstar::set_final_state(cState *state)
{
    final_state=state;
}

cState *cAstar::activate_search()
{
    cState *state;
    char chr;
    int total_iteration_number, current_size;
    int enabled_transitions_number, i;
    int *enabled_transitions=new int[topology->get_places()];
    cout<<"Inserting initial state.\n";
    open_list->insert_state(initial_state);
    cout<<"Starting the main loop.\n";
    total_iteration_number=0;
    while (!(*final_state)==open_list->get_best_state() && !open_list->empty())
    {
        state=open_list->get_best_state();
        close_list->insert_state(state);
        open_list->delete_current();
        enabled_transitions_number=0;
        topology->get_enabled_transitions(state,
                                           &enabled_transitions_number, enabled_transitions);
        for (i=0;i<enabled_transitions_number;i++)
            open_list->insert_state(topology->
```

fire_transition(state, enabled_transitions[i]);
current_size=open_list->get_current_size();
if (current_size>MAX_DEPTH)
    for (i=0;i<current_size-MAX_DEPTH;i++)
        open_list->destroy_last();
    total_iteration_number++;
}
if (open_list->empty()) return(0);
cout<<"Total iterations number: "<<total_iteration_number<<"n";
cin>>chr;
return(open_list->get_best_state());

// cCloseList.h

#ifndef _CCLOSELIST
#define _CCLOSELIST

#include "cState.h"
#include "cNode.h"
#include "cList.h"

class cCloseList : public cList
{
private:
    int collisions;
public:
    cCloseList();
    ~cCloseList();
    void insert_state(cState *state);
    int get_collisions();
};
#endif

// cCloseList.cpp

#include "cCloseList.h"

cCloseList::cCloseList()
{
}

cCloseList::~cCloseList()
{
void cCloseList::insert_state(cState *state) 
{
    cNode *found_node;
    found_node=search_state(state);
    if (found_node)
    {
        cout<<"Fetal: Can not handle event.\n";
        while (1);
    }
    else insert_last(state);
}

int cCloseList::get_collisions()
{
    return(collisions);
}

// cList.h

#ifndef _cList
#define _cList

#include "cState.h"
#include "cNode.h"

class cList  
{
protected:
    cNode *first_node;
    cNode *last_node;
    int current_size;
    cNode *index;
    void insert_after(cState *state);
    void insert_first(cState *state);
    void insert_last(cState *state);

public:
    cList();
    ~cList();
    void delete_current();
    void destroy_last();
    int get_current_size();
    cNode *search_state(cState *state);
    int empty();
};
#endif

// cList.cpp

#include "cList.h"

cList::cList()
{
    first_node=0;
    last_node=0;
    current_size=0;
    index=0;
}

cList::~cList()
{
}

void cList::insert_after(cState *state)
{
    cNode *new_node;
    if (!first_node) insert_first(state);
    else if (index==last_node) insert_last(state);
    else
    {
        new_node=new cNode(index->get_next_node(), index, state);
        index->get_next_node()->set_previous_node(new_node);
        index->set_next_node(new_node);
        index=new_node;
        current_size++;
    }
}

void cList::insert_first(cState *state)
{
    cNode *new_node;
    if (!first_node)
    {
        first_node=new cNode(0, 0, state);
        last_node=first_node;
    }
    else
    {
        new_node=new cNode(first_node, 0, state);
        first_node->set_previous_node(new_node);
    }
first_node=new_node;
    }
    current_size++;
}

void cList::insert_last(cState *state)
{
cNode *new_node;
if (!first_node) insert_first(state);
else
{
    new_node = new cNode(0, last_node, state);
    last_node->set_next_node(new_node);
    last_node = new_node;
}
current_size++;
}

void cList::delete_current()
{
cNode *killed_node;
if (!first_node)
{
    cout<<"Open list is empty..\n";
    return;
}
else
{
    if (current_size==1)
    {
        delete index;
        last_node=0;
        first_node=0;
        index=0;
        current_size=0;
        return;
    }
    if (index==first_node)
    {
        first_node=index->get_next_node();
        first_node->set_previous_node(0);
        killed_node=index;
        index=first_node;
    }
    else if (index==last_node)
    {

last_node=index->get_previous_node();
last_node->set_next_node(0);
killed_node=index;
index=last_node;
}
else
{
index->get_previous_node() -> set_next_node(index->
get_next_node());
index->get_next_node() -> set_previous_node(index->
get_previous_node());
killed_node=index;
index=index->get_previous_node();
}
delete killed_node;
current_size--;
}

void cList::destroy_last()
{
index=last_node;
delete index->get_data_node();
delete_current();
}

int cList::get_current_size()
{
return(current_size);
}

cNode *cList::search_state(cState *state)
{
index=first_node;
while (index)
{
if ((*state)==index->get_data_node())
return(index);
else index=index->get_next_node();
}
return(0);}

int cList::empty()
{
if (current_size) return(0);
else return(1);
}

// cNode.h

#ifndef __cNode
#define __cNode

#include "cState.h"

class cNode
{
    private:
        cNode *next_node;
        cNode *previous_node;
        cState *node_data;
    public:
        cNode(cNode *next, cNode *previous, cState *data);
        cNode();
        ~cNode();
        void set_next_node(cNode *next);
        cNode *get_next_node();
        void set_previous_node(cNode *previous);
        cNode *get_previous_node();
        void set_data_node(cState *data);
        cState *get_data_node();
};
#endif

// cNode.cpp

#include "cNode.h"

cNode::cNode(cNode *next, cNode *previous, cState *data)
{
    set_next_node(next);
    set_previous_node(previous);
    set_data_node(data);
}

cNode::cNode()
{
}

cNode::~cNode()
```cpp
void cNode::set_next_node(cNode *next)
{
    next_node=next;
}

cNode *cNode::get_next_node()
{
    return(next_node);
}

void cNode::set_previous_node(cNode *previous)
{
    previous_node=previous;
}

cNode *cNode::get_previous_node()
{
    return(previous_node);
}

void cNode::set_data_node(cState *data)
{
    node_data=data;
}

cState *cNode::get_data_node()
{
    return(node_data);
}

// cOpenList.h
#endif _cOpenList
#define _cOpenList

#include "cState.h"
#include "cNode.h"
#include "cList.h"
#include "cCloseList.h"

class cOpenList : public cList
{
private:
```
int collisions;
void insert_in_order(cState *state);
cCloseList *close_list;

public:
cOpenList(cCloseList *closelist);
~cOpenList();
void insert_state(cState *state);
cState *get_best_state();
int get_collisions();
void dump();
);

#endif

// cOpenList.cpp

#include "cOpenList.h"
#include <stdlib.h>

cOpenList::cOpenList(cCloseList *closelist)
{
    close_list=closelist;
}

cOpenList::~cOpenList()
{
}

void cOpenList::insert_state(cState *state)
{
    cNode *found_node;
    found_node=search_state(state);
    if (found_node)
    {
        cout<<"Collision detected in open list. \n";
        collisions++;
        if (state->get_cost()>=found_node->get_data_node()->get_cost())
            delete state;
        else
        {
            delete_current();
            insert_in_order(state);
        }
    }
    return;
}
found_node=close_list->search_state(state);
if (found_node)
{
    cout<<"Collision detected in close_list..\n";
    collisions++;
    if (state->get_cost()>=found_node->get_data_node()->get_cost())
        delete state;
    else
    {
        close_list->delete_current();
        insert_in_order(state);
    }
    return;
}
insert_in_order(state);

void cOpenList::insert_in_order(cState *state)
{
    float cost;
    if (!first_node)
    {
        insert_first(state);
        return;
    }
    cost=state->get_cost();
    index=first_node;
    #ifdef INSERT_AFTER
    if (cost<first_node->get_data_node()->get_cost())
        insert_first(state);
    else if ((cost==first_node->get_data_node()->get_cost())
            && (get_current_size()==1)) insert_after(state);
    else if (cost>last_node->get_data_node()->get_cost())
        insert_last(state);
    else
        while (index->get_next_node())
        {
            if ((cost>index->get_data_node()->get_cost())
            &&
                (cost<=index->get_next_node()->
                get_data_node()->get_cost()))
            {
                insert_after(state);
                return;
            }
            index=index->get_next_node();
        }
    #endif
}
#else
    if (cost <= first_node->get_data_node()->get_cost())
        insert_first(state);
    else if (cost >= last_node->get_data_node()->get_cost())
        insert_last(state);
    else
        while (index->get_next_node())
        {
            if ((cost >= index->get_data_node()->get_cost()) &&
                (cost <= index->get_next_node()->
                 get_data_node()->get_cost()))
            {
                insert_after(state);
                return;
            }
        }
        else index = index->get_next_node();
#endif
}

cState *cOpenList::get_best_state()
{
    index = first_node;
    if (!index)
    {
        cout << "FATAL : cState *cOpenList::get_best_state() : Empty open list encountered\n";
        exit(1);
    }
    return(first_node->get_data_node());
}

int cOpenList::get_collisions()
{
    return(collisions);
}

void cOpenList::dump()
{
    index = first_node;
    while (index)
    {
        index->get_data_node()->dump();
        index = index->get_next_node();
    }
}
// cState.h

#ifndef _cState
#define _cState

#include <fstream.h>
#include "Parameters.h"

class cState
{
private:
    int *marking;
    float *remaining_time;
    cState *previous_state;
    int places_number;
    int transition_id;
    int state_id;
    float cost;

public:
    cState(int *mark, float *time, cState *state, int transition ,int places);
    cState();
    ~cState();
    cState *get_previous_state();
    void calculate_cost(float consumed);
    void set_places(int places);
    void set_previous_state(cState *state);
    void set_transition_id(int id);
    int get_transition_id();
    void set_marking(int *mark);
    int *get_marking();
    void set_time(float *time);
    float *get_time();
    void set_state_id(int id);
    int get_state_id();
    float get_cost();
    int operator==(cState *second_state);
    void dump();
};
#endif

int cState::get_state_id()
{
    return(state_id);
float cState::get_cost()
{
    return(cost);
}

int cState::operator==(cState *second_state)
{
    int i;
    for (i=0;i<places_number;i++)
        if (((get_marking())[i]) !=
            (second_state->get_marking())[i]) return(0);
    return(1);
}

void cState::dump()
{
    int i;
    for (i=0;i<places_number;i++)
        cout<<(get_marking())[i]<<(get_time())[i];
    cout<<\n"<<get_cost()<<" "<<get_transition_id()<<\n";?

// cTopology.h

ifndef _cTopology
define _cTopology

#include "cState.h"

class cTopology
{
private:
    int places_number;
    int transition_number;
    int *A;
    cState *initial_state;
public:
    cTopology(int *a, int places, int transitions, cState *state);
    cTopology();
    ~cTopology();
    void set_initial_state(cState *state);
    void set_topology(int *a, int places, int transitions);
    int *get_topology();
    int get_places();
int get_transitions();
void get_enabled_transitions(cState *state, int *counter,
                             int *transitions);
cState *fire_transition(cState *state, int id);

#endif

// cTopology.cpp

#include "cTopology.h"

// build the topology object
cTopology::cTopology(int *a, int places, int transitions, cState *state)
{
    set_topology(a, places, transitions);
    set_initial_state(state);
}

cTopology::cTopology()
{
}

cTopology::~cTopology()
{
}

void cTopology::set_initial_state(cState *state)
{
    initial_state=state;
}

void cTopology::set_topology(int *a, int places, int transitions)
{
    places_number=places;
    transition_number=transitions;
    A=a;
}

int *cTopology::get_topology()
{
    return(A);
}

int cTopology::get_places()
{
```c
return(places_number);

int cTopology::get_transitions()
{
    return(transition_number);
}

// find all the enabled transitions for a given state
void cTopology::get_enabled_transitions(cState *state, int *counter, int *transitions)
{
    int flag, i, j;
    char t;
    *counter=0;
    for (i=0; i<transition_number; i++)
    {
        flag=0;
        for (j=0; j<places_number; j++)
        {
            if ((A[i*places_number+j]<0) && !((state->get_marking())[j])) flag=1;
            if (flag) break;
            // okay this is an enabled transition
        }
        if (!flag) // store it and continue searching
        {
            transitions[*counter]=i;
            (*counter)++;
        }
    }
}

// fire a transition and return the new state
cState *cTopology::fire_transition(cState *state, int id)
{
    int j, *marking;
    float time, *remaining_time;
    // copy the given state into the new state
    time=0.0;
    cState *new_state;
    marking=new int[places_number];
    remaining_time=new float[places_number];
    for (j=0; j<places_number; j++)
    {
        marking[j]=(state->get_marking())[j];
        remaining_time[j]=(state->get_time())[j];
    }
    return new_state;
}
```
new_state=new cState(marking, remaining_time, state, id, places_number);
// adjust the copy to match the new state after firing
for (j=0; j<places_number; j++) // for each place
{
    if (A[id*places_number+j]<0) // if it is an input place
    {
        if (time<(new_state->get_time())[j]) // if it has less time
            time=(new_state->get_time())[j];
        (new_state->get_marking())[j]=--; // decrement input places
        if (!((new_state->get_marking())[j])) // update the time
        {
            (new_state->get_time())[j]=
            (initial_state->get_time())[j];
        }
    }
    else if (A[id*places_number+j]>0) // if it is an output place
        (new_state->get_marking())[j]++; // increment places
}
new_state->calculate_cost(time); // calculate the new cost
for (j=0; j<places_number; j++)
    if ((new_state->get_marking())[j])
        if ((new_state->get_time())[j]-time<0)
            (new_state->get_time())[j]=0;
    else
        (new_state->get_time())[j]=
        (new_state->get_time())[j]-time;
return new_state; // return the new state

// Parameters.h

#ifndef _PARAMETERS
#define _PARAMETERS

#define INSERT_AFTER    // the ordering strategy
#define TRANSITIONS 43   // total number of transitions
#define PLACES 52        // total number of places
#define W 1              // the weighting factor w
#define MAX_DEPTH 5000   // the maximum capacity of the OPEN LIST b

#endif

// main.cpp

#include <fstream.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include "cAStar.h"
#include "Parameters.h"

void main()
{
    time_t  start, finish;
    double  elapsed_time;
    int i,j;
    int *a;
    char kk;
    int initial_mark[PLACES];
    float initial_timing[PLACES];
    int final_mark[PLACES];
    float final_timing[PLACES];
    cState *initial_state;
    cState *final_state, *temp;
    cState *found_state;
    cTopology *topology;
    cAstar *astar;
    ifstream a_matrix_file("a-matrix.out",ios::in);
    ifstream initial_mark_file("initial.out",ios::in);
    ifstream final_mark_file("final.out",ios::in);
    a=new int[PLACES*TRANSITIONS];
    cout<<"Now reading input files.\n"
    for (i=0;i<TRANSITIONS;i++)
    {
        for (j=0;j<PLACES;j++)
        {
            a_matrix_file>>a[i*PLACES+j];
            cout<<a[i*PLACES+j]<<" ";
        }
    }
    for (i=0;i<PLACES;i++)
    {
        initial_mark_file>>initial_mark[i];
        cout<<initial_mark[i];
        initial_mark_file>>initial_timing[i];
        cout<<initial_timing[i];
        final_mark_file>>final_mark[i];
        cout<<final_mark[i];
        final_mark_file>>final_timing[i];
        cout<<final_timing[i];
    }
cout<<"nConstructing search engine..\n";
initial_state=new cState(initial_mark,initial_timing,0,0,PLACES);
final_state=new cState(final_mark,final_timing,0,0,PLACES);
topology=new cTopology(a,PLACES,TRANSITIONS,initial_state);
astar=new cAstar(initial_state,final_state,topology);
cout<<"Starting the search engine..\n";
time( &start );
found_state=astar->activate_search();
time( &finish );
if (found_state) cout<<"Final state was found.\n";
else
{
    cout<<"Failed\n";
    return;
}
i=0;
temp=found_state;
while (found_state)
{
    cout<<found_state->get_transition_id() << "\n" << found_state->get_cost()<<"\n";
    found_state=found_state->get_previous_state();
    i++;
}
cout<<"Total schedule time: "<<temp->get_cost()+(i-1)*W<<"\n";
cout<<"Depth found: "<<i-1<<"\n";
elapsed_time = diff_time( finish, start );
cout<<"Generation time in sec: "<<elapsed_time;
}