Fall 10-31-1996

Scheduling and discrete event control of flexible manufacturing systems based on Petri nets

Huanxin Henry Xiong
New Jersey Institute of Technology

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

ABSTRACT

SCHEDULING AND DISCRETE EVENT CONTROL OF FLEXIBLE MANUFACTURING SYSTEMS BASED ON PETRI NETS

by

Huanxin Henry Xiong

A flexible manufacturing system (FMS) is a computerized production system that can simultaneously manufacture multiple types of products using various resources such as robots and multi-purpose machines. The central problems associated with design of flexible manufacturing systems are related to process planning, scheduling, coordination control, and monitoring. Many methods exist for scheduling and control of flexible manufacturing systems, although very few methods have addressed the complexity of whole FMS operations. This thesis presents a Petri net based method for deadlock-free scheduling and discrete event control of flexible manufacturing systems. A significant advantage of Petri net based methods is their powerful modeling capability. Petri nets can explicitly and concisely model the concurrent and asynchronous activities, multi-layer resource sharing, routing flexibility, limited buffers and precedence constraints in FMSs. Petri nets can also provide an explicit way for considering deadlock situations in FMSs, and thus facilitate significantly the design of a deadlock-free scheduling and control system.

The contributions of this work are multifold. First, it develops a methodology for discrete event controller synthesis for flexible manufacturing systems in a timed Petri net

framework. The resulting Petri nets have the desired qualitative properties of liveness, boundedness (safeness), and reversibility, which imply freedom from deadlock, no capacity overflow, and cyclic behavior, respectively. This precludes the costly mathematical analysis for these properties and reduces on-line computation overhead to avoid deadlocks. The performance and sensitivity of resulting Petri nets, thus corresponding control systems, are evaluated. Second, it introduces a hybrid heuristic search algorithm based on Petri nets for deadlock-free scheduling of flexible manufacturing systems. The issues such as deadlock, routing flexibility, multiple lot size, limited buffer size and material handling (loading/unloading) are explored. Third, it proposes a way to employ fuzzy dispatching rules in a Petri net framework for multi-criterion scheduling. Finally, it shows the effectiveness of the developed methods through several manufacturing system examples compared with benchmark dispatching rules, integer programming and Lagrangian relaxation approaches.

SCHEDULING AND DISCRETE EVENT CONTROL OF FLEXIBLE MANUFACTURING SYSTEMS BASED ON PETRI NETS

by
Huanxin Henry Xiong

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

Department of Electrical and Computer Engineering

October 1996

APPROVAL PAGE

SCHEDULING AND DISCRETE EVENT CONTROL OF FLEXIBLE MANUFACTURING SYSTEMS BASED ON PETRI NETS

Huanxin Henry Xiong

Dr. MengChu Zhou, Dissertation Advisor — Date
Associate Professor of Electrical and Computer Engineering, NJIT

Dr. Reggie J. Caudill, Dissertation Co-Advisor — Date
Professor of Industrial and Manufacturing Engineering, NJIT

Dr. John Carpinelli, Committee Member — Date
Associate Professor of Electrical and Computer Engineering,
Computer and Information Science, NJIT

Dr. Nirwan Ansari, Committee Member — Date
Associate Professor of Electrical and Computer Engineering, NJIT

Dr. Edwin Hou, Committee Member — Date
Associate Professor of Electrical and Computer Engineering, NJIT

BIOGRAPHICAL SKETCH

Author: Huanxin Henry Xiong

Degree: Doctor of Philosophy

Date: October 1996

Undergraduate and Graduate Education:

- Doctor of Philosophy in Electrical Engineering
  New Jersey Institute of Technology, Newark, NJ, 1996

- Master of Science in Systems Engineering
  Nanjing University of Science and Technology, Nanjing, P. R. China, 1988

- Bachelor of Science in Systems Engineering
  Nanjing University of Science and Technology, Nanjing, P. R. China, 1985

Major: Electrical Engineering

Presentations and Publications:

H. H. Xiong, M. C. Zhou and R. J. Caudill, "Design of optimal sequence controller for a flexible manufacturing system," to appear in *Proceedings of 1996 IEEE Int. Conf. on Systems, Man, and Cybernetics*, Beijing, China, Oct. 1996.

H. H. Xiong, M. C. Zhou and R. J. Caudill, "A hybrid heuristic search algorithm for scheduling flexible manufacturing systems," in *Proceedings of 1996 IEEE Int. Conf. on Robotics and Automation*, Minneapolis, MN, pp. 2793-2797, Apr. 1996.

H. H. Xiong, M. C. Zhou and C. N. Manikopoulos, "Scheduling flexible manufacturing systems based on timed Petri nets and fuzzy dispatching rules," in *Proceedings of 1995 IEEE Symposium on Emerging Technologies and Factory Automation*, Paris, France, pp. 309-315, Oct. 1995.

M. C. Zhou and H. H. Xiong, "Techniques in Petri net modeling and scheduling in manufacturing systems," *Gordon and Breach International Series in Engineering, Technology and Applied Science, Volume on Computer Aided and Integrated Manufacturing Systems Techniques and Applications*, to appear in 1997.

M. C. Zhou, H. Chiu and H. H. Xiong, "Petri net scheduling of FMS using branch and bound method," *Proc. of 1995 IEEE Int. Conf. on Industrial Electronics, Control, and Instrumentation*, Orlando, FL, pp. 211-216, Nov. 1995.

M. C. Zhou, H. H. Xiong and C. N. Manikopoulos, "Performance models for communication networks in manufacturing environment," in *Proceedings of the Fourth Int. Conf. on Computer Integrated Manufacturing and Automation Technology*, Troy, NY, pp. 417-422, Oct. 1994.

H. H. Xiong, M. C. Zhou and C. N. Manikopoulos, "Modeling and performance analysis of medical service systems using Petri nets," in *Proceedings of 1994 IEEE Int. Conf. on Systems, Man, and Cybernetics*, San Antonio, TX, pp. 2339-2342, Oct. 1994.

H. H. Xiong and M. C. Zhou, "Computer communication networks for automated manufacturing," in *Proceedings of 1994 Int. Conf. on Electronics and Information Technology*, Beijing, China, pp. 178-183, Aug. 1994.

This dissertation is dedicated to
my parents

ACKNOWLEDGMENT

The author expresses his deep sense of gratitude to his adviser, Dr. MengChu Zhou, for his invaluable guidance, friendship, and moral support throughout this research. Dr. Zhou's untiring help is sincerely appreciated.

Special thanks to his co-advisor Dr. Reggie Caudill for his encouragement, mentorship, and suggestions. The author is grateful to Professors John Carpinelli, Nirwan Ansari, and Edwin Hou for their suggestions and constructive comments. Their service as committee members is appreciated. The author would also like to express his appreciation to Dr. C. N. Manikopoulos for his help and encouragement.

The author's appreciation and thanks go to his fellow graduate students at Laboratory for Discrete Event Systems, Electrical and Computer Engineering Department. They are Dr. Venkatesh Kurapati, Mr. Hua-Sheng Chiu and Mr. Xin Ren.

The author's appreciation reaches out to his professor Zitong Huang at Nanjing University of Science and Technology (NUST), Nanjing, P. R. China for his support and encouragement in the period of his study and work at NUST.

Finally, the author wishes to express his sincere gratitude to his parents, wife, brothers and lovely son for their moral support, understanding and love.

This research was supported by the New Jersey Commission on Science and Technology via the Center for Manufacturing Systems at NJIT and the National Science Foundation under Grant No. DMI-9410386.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES
(Continued)

LIST OF FIGURES
(Continued)

CHAPTER 1

INTRODUCTION

In this chapter, the background, motivation and objectives of this work are stated. The organization of this dissertation is outlined.

1.1 Background and Motivation

A flexible manufacturing system (FMS) is a computerized production system that can simultaneously manufacture multiple types of products using various resources such as robots and multi-purpose machines. An FMS consists of a set of computer numerically controlled machine tools and supporting workstations connected by an automated material handling system. It can be controlled by either a central computer or distributed computers. In the latter case, one main computer serves as a supervisory one to synchronize and coordinate the other computers, forming a hierarchical computer control architecture (Zhou, DiCesare and Rudolph 1992). The key elements of an FMS include (1) automatically programmable machines, (2) automated tool delivery and change, (3) automated material handling for transferring parts between machines and for loading/unloading parts at machines, and (4) coordinated control (Askin and Standridge 1993).

In an FMS, many part types can be simultaneously loaded onto the system because machines have tooling and processing information to work on multiple types of products. While the flexibility in FMS offers an opportunity to meet customer demand for product

variety in a timely fashion and at low cost, its design and operation impose many challenging problems on planning, scheduling, monitoring and control of manufacturing systems (Chaar *et al.* 1993).

The central problems associated with design of flexible manufacturing systems are related to process planning, scheduling, coordination control, and monitoring.

Given a set of production requirements and a physical system configuration, scheduling deals with the allocation of shared resources over time for manufacturing products such that all the production constraints are satisfied, production cost is minimized and productivity is maximized. The control decisions deal with the coordination and execution of part flow and processing. The controller must be capable of keeping track of system states such as the location of all parts, and the operational status of each resource. Based on the current state and production plan, the controller supervises all the individual system components.

Production scheduling problems are known to be very complex and are NP-hard for general cases. Compared with a classical job shop system, the main characteristics of an FMS include multi-layer resource-sharing, deadlock and routing flexibility. A flexible manufacturing system consists of different kind of resources such as machines, robots, transporters and buffers. The job processes share all machines and machines share transportation systems, robots, tools and so on. The complex interaction of the multiple resources and concurrent flow of multiple jobs in an FMS can lead to a deadlock situation in which any part flow is inhibited. The occurrence of a deadlock can cripple the entire system. This requires an explicit consideration of deadlock conditions in the scheduling

and control methods to prevent from or avoid the deadlock states in FMSs. Machine routings specify the machines that are required for each operation of a given job. In an FMS, a job may have alternative routings. The routing flexibility results in benefits to the system such as increasing the throughput and handling the machine breakdown situations, while it increases the complexity of scheduling and control of FMSs. Other factors of FMS operations include multi-criteria optimization objective and stochastic working environment due to processing time variations, machine breakdowns and demand changes.

Many researchers are constantly seeking advanced and unifying methodologies for modeling, performance evaluation, scheduling and control of flexible manufacturing systems. A review about these methodologies is presented in Chapter 2. One methodology resulting from this effort is based on Petri nets and related graphical and mathematical tools. This dissertation is dedicated to the investigation of Petri net based method for deadlock-free scheduling and discrete event control of flexible manufacturing systems. The motivation for the present work is described below:

- The concepts of liveness, boundedness and reversibility of Petri nets are central to the function of a coordinating discrete-event controller. If a system is live, then all events associated with that system can eventually occur. The liveness implies deadlock-freeness. Boundedness or safeness guarantees a stable discrete manufacturing process or no capacity overflow. Reversibility ensures a cyclic manufacturing system with the ability to initialize from any reachable state and has implications for error recovery in the manufacturing context. In the literature, there are actually three kinds of approaches for avoiding deadlocks in an FMS. The first

approach addresses deadlock detection and recovery (Viswanadham, Narahari and Johnson 1990, Wysk, Yang and Joshi 1994). In this approach, if a deadlock results, the system detects and resolves it. It is obvious that using such a method may be very costly since it may be expensive to detect and resolve deadlocks in a fully automated system. The second approach emphasizes on-line deadlock avoidance (Banaszak and Krogh 1990, Hsieh and Chang 1994, Xing, Hu and Chen 1995). Some deadlock avoidance policies are proposed to restrict requests for resources when they will potentially lead to circular wait conditions. For example, the easiest way is to allow only one job in the system at a time. Thus, the difficulty behind this approach is how to develop a less restrictive policy which not only avoids deadlocks but also allows the maximal use of resources. This kind of approach also results in on-line computation overhead. The third approach emphasizes on designing a controller which inherently guarantees the desirable properties of liveness, boundedness, and reversibility. Our present research falls into this category. Even though there are several studies in this aspect (Krogh and Beck 1986, Koh and DiCesare 1991, Zhou, DiCesare and Desrochers 1992), for the system with multi-layer resource-sharing and different product sets manufactured concurrently, modeling of a Petri net controller with desirable properties becomes extremely difficult based on these methods.

- Previous studies in scheduling and control of flexible manufacturing systems are reviewed in Chapter 2. Even though many methods exist for scheduling of flexible manufacturing systems, very few methods have addressed the complexity of whole

FMS operations. Typical assumptions are still confined to the classical job shop environments for most methods. Petri nets can explicitly and concisely model the concurrent and asynchronous activities, multi-layer resource sharing, part contact states (loading/unloading), routing flexibility, limited-size buffers and precedence constraints in flexible manufacturing systems. Petri nets can also provide an explicit way for considering deadlock situations in FMSs, and thus facilitate deadlock-free scheduling of flexible manufacturing systems. These modeling capabilities of Petri nets motivate us to investigate Petri net based methods for scheduling of flexible manufacturing systems. Lee and DiCesare (1994) presented a scheduling method using Petri nets and heuristic search. The proposed heuristic functions do not guarantee to satisfy the admissible condition (Pearl 1984). Moreover, no deadlock issues are discussed in their demonstrated examples because they always put an intermediate place which serves as the role of a buffer with unlimited capacity between two operations.

- Because of their simplicity, heuristic dispatching rules, such as SPT (Shortest Processing Time), EDD (Earliest Due Date), S/RO (Slack per Remaining Operation), and FCFS (First Come First Served) have been commonly used for scheduling in practice (Montazeri and Wassenhove 1990). Each of these dispatching rules aims at satisfying a single criterion. A rule that performs well when one measure is used may not do well for another measure (Blackstone *et al.* 1982). There is a need to develop some simple combined rules to obtain a compromise between the satisfaction of several criteria.

1.2 Objectives

The goal of this dissertation is to develop a Petri net based method for deadlock-free scheduling and discrete event control of flexible manufacturing systems. The specific objectives are:

1. To present a review on the current methodologies for scheduling and control of flexible manufacturing systems.
2. To present the definitions and properties of Petri nets. The conventional methods for Petri net modeling of manufacturing systems are given and illustrated through examples.
3. To develop a methodology for discrete event controller synthesis for flexible manufacturing systems in a timed Petri net framework. The method should guarantee that the resulting Petri nets have the desired qualitative properties of liveness, boundedness, and reversibility. The performances and sensitivities of resulting Petri net controllers are evaluated.
4. To develop a hybrid heuristic search algorithm based on Petri nets for deadlock-free scheduling of flexible manufacturing systems. The issues such as deadlock, routing flexibility, multiple lot size, limited buffer size and material handling (loading/unloading) are explored.
5. To propose a way to employ fuzzy dispatching rules in a Petri net framework for multi-criterion scheduling.

1.3 Organization

The next chapter presents a literature review of methodologies for scheduling and control of flexible manufacturing systems and suggests that the Petri net based methods have their potential to make major contributions to FMS operation. Chapter 3 contains the discussion of the fundamentals of Petri nets. The conventional methods for Petri net modeling of manufacturing systems are given and illustrated through examples.

In Chapter 4, a methodology for synthesis of Petri net based discrete event controller is presented. The bottom-up method is used to modeling the system. Once the modeling is done, the A* based heuristic search algorithm, which is combined with the execution of the timed Petri nets, is proposed to search for an optimal event sequence to achieve minimum-time discrete event control. Based on the obtained event-driven sequence, a Petri net (marked graph) is synthesized for coordinating discrete event control. The theoretical results which insure the desired qualitative properties of liveness, boundedness (safeness), and reversibility of resulting Petri net controller are obtained. The performance and sensitivity of the resulting Petri net controller are evaluated and illustrated through examples.

In Chapter 5, a hybrid heuristic search algorithm based on Petri nets for deadlock-free scheduling of flexible manufacturing systems is presented. Two different hybrid strategies are compared through examples. The issues such as deadlock, routing flexibility, multiple lot size, limited buffer size are explored. The developed method is compared with benchmark dispatching rules and depth-first search. A scheduling example for a

semiconductor test facility solved by Chen (1994) using integer programming and Lagrangian relaxation technique is adopted and solved based on our developed method.

In Chapter 6, FMS scheduling with material handling (loading/unloading) and buffer availability considered is presented. Deadlock arises from explicit recognition of material handling and buffer space resources. The inappropriate scheduling decisions may lead to a deadlock state in which any part flow is inhibited and external intervention is required to reestablish the product flow. To demonstrate the modeling capability of Petri nets, the example is adopted from a recent paper presented by Ramaswamy and Joshi (1996), which generates deadlock-free schedules using the mathematical programming techniques.

In Chapter 7, multi-criterion scheduling based on Petri nets is presented. The Petri net model resolves conflicting transition firings using fuzzy dispatching rules which obtain a compromise between the satisfaction of several criteria. The scheduling example is given to illustrate the method.

Finally, Chapter 8 discusses the contributions and limitations of this research along with suggestions for further research.

CHAPTER 2

LITERATURE REVIEW

2.1 Methodologies

Many methods exist for scheduling and control of flexible manufacturing systems, although very few methods have addressed the complexity of whole FMS operations. The general methods include mathematical programming method, heuristics dispatching and knowledge-based method, control theoretic method and Petri net based method.

2.1.1 Mathematical Programming Methods

Much effort is focused on scheduling of manufacturing systems using mathematical programming methods such as linear programming, integer programming and dynamic programming.

Luh and Hoitomt (1993) presented a Lagrangian relaxation technique for scheduling of manufacturing systems. Lagrangian relaxation is mathematical programming technique for performing constrained optimization. Three kinds of problems are examined in their research. The first kind considers scheduling single-operation jobs on identical machines. The second one is concerned with scheduling multiple-operation jobs on identical machines. The last one is a job shop problem, where multiple-operation jobs are scheduled on multiple machine types. Lagrangian relaxation is used to decompose each of the scheduling problems into job- or operation-level subproblems which are easier to solve than the original problem. Numerical results show that the methods obtain near-optimal

schedules in a timely fashion. An improved Lagrangian relaxation technique is presented by Czerwinski and Luh (1994) to make Lagrangian relaxation a viable approach to more complicated problems.

Chen (1994) formulated semiconductor manufacturing test floor environments as integer programming problems. Four scheduling models are proposed in his work. They are (1) scheduling for IC sort and test facilities with nonpreemptive assumption; (2) scheduling for IC sort and test facilities with preemption; (3) model 1 or 2 plus precedence constraints and (4) model 3 plus due windows. The objective is to minimize the total weighted tardiness or weighted quadratic tardiness and earliness of the schedule. The Lagrangian relaxation approach is used to solve the problems and generate better scheduling results compared with traditional heuristic dispatching rules.

Blazewicz *et al.* (1991) presented a dynamic programming approach for scheduling tasks and vehicles in a flexible manufacturing system. In the first step, the production schedule (i.e., the assignment of jobs to machines) is assumed to be known, and the objective is to find a feasible schedule for vehicles. Then a composite schedule, i.e., simultaneous assignment of vehicles and machines to jobs, is found. The considered system assumes every machine in the system is capable of processing any of the required machining operations.

Recently, Ramaswamy and Joshi (1996) applied integer programming techniques for dead-lock free scheduling of automated manufacturing workstations. Besides the classic constraints of precedence relations and processing times, they add one more constraint for ensuring that a job leaves a machine only when it has found space on the next machine.

Although both material handling and buffer space are explicitly considered in generated schedules, the proposed deadlock-free scheme is only applicable to problems with m machines and $\lfloor m/2 \rfloor$ buffers. Other characteristics of FMS such as multiple lot sizes, multiple buffers and routing flexibility are not explored in their work.

Basnet and Mize (1994) presented a critical review about the methodology for scheduling and control of flexible manufacturing systems. As they point out, the main problem with mathematical programming method is its formulation difficulties. The models do not consider the full complexity of general FMSs, such as shared resources, concurrency, routing flexibility, multiple lot sizes and deadlock states.

2.1.2 Heuristics Dispatching and Knowledge-Based Methods

Because of its NP-hard characteristics, it is very difficult or impossible to find the optimal solution for a sizable FMS scheduling problem. The dispatching rules, such as SPT (Shortest Processing Time), EDD (Earliest Due Date), S/RO (Slack per Remaining Operation) and FCFS (First Come First Served), are thus practically employed to determine the priority of jobs for processing by machines in flexible manufacturing. The flowtime, lateness, and tardiness have been used as measures of the effectiveness of dispatching rules. Discrete event simulation is proposed as a tool to evaluate the performance of different dispatching rules.

Montazeri and Wassenhove (1990) analyzed the performance of a number of dispatching rules using a modular simulator to mimic the operation of a real-life FMS. Ishii and Talavage (1994) used a mixed dispatching rule which can assign a different

dispatching rule for each machine in contrast with the approach in which a single dispatching rule is assigned for all machines. A search algorithm which selects an appropriate mixed dispatching rule using predictions based on discrete event simulation is developed. The effectiveness of the mixed dispatching rule approach is tested for a relatively simple FMS model. It should be tested using their more complex models before being applied to real FMSs.

Wu and Wysk (1988, 1989) described a multi-pass expert control system (MPECS) for FMS scheduling and control. The key elements of MPECS include an expert system to generate potential scheduling alternatives based on real-time shop information and scheduling knowledge, and a simulation model to evaluate alternative schedules based on the system's performance. Various criteria for selecting heuristic dispatching rules are stored in a knowledge-base. The major function of the simulation model is to evaluate control polices by examining the effect of the dispatching rules on an on-line test base. A series of simulation runs is carried out starting from the current state using each of the candidate dispatching rules for a user defined simulation window. At the end of all simulation passes, the best dispatching rule that results from the simulation is applied to the physical manufacturing system. The experiment shows the performance of MPECS is significantly better than the performance of the methods that use a single dispatching rule all the time. But the performance greatly depends on the length of the simulation windows, which is defined by the users.

Doulgeri *et al.* (1993) developed a knowledge-based scheduler for FMS which adopts the hierarchical approach and utilizes simulation techniques. The knowledge-based

scheduler consists of two basic modules, the knowledge-based FMS model and rule-based decision making module. In the heart of the knowledge-based FMS model are the static FMS model which contains a frame representation of the FMS elemental components and their time-independent attribute values and the dynamic FMS model which is object-oriented event-driven simulation. The rule-based decision module performs the FMS short-term production scheduling by interacting with the knowledge-based FMS model. The scheduler adopts a hierarchical approach, where the upper level issues commands concerning the multi-type introduction of new parts into the system and the lower level makes decisions concerning the detailed movement of parts through the system resources. The system is demonstrated in a flexible printed circuit board assembly system.

2.1.3 Control Theoretic Methods

Kimemia and Gershwin (1983) presented a multilevel hierarchical control scheme for the computer control of flexible manufacturing systems. In their proposed closed-loop control policy, parts are loaded into the system in such a way that the system is neither overloaded, nor congested and the long-term production objective is met. The flow control level determines the short-term production rates of each member of the part family. Because of the time-varying demand and reliability of the workstations, it involves a stochastic optimal control problem at this level. A part entering the FMS has one or more machine routings. The routing control level determines the flow rate on each path based on the arrival rate of the parts chosen by the flow control level. The lowest level of

control is a scheduling algorithm that schedules times at which parts are dispatched to maintain the flow rates chosen by the flow and route controllers.

Based on the hierarchical structure proposed by Kimemia and Gershwin (1983), Custodio *et al.* (1994) presented a fuzzy controller for production scheduling and control. The purpose of their controller is to get cumulative production to track cumulative demand while keeping the work-in-process low. The new idea of their method is to allow the use of multiple criteria, each with an assigned fuzzy weight. This is advantageous since the use of several different fuzzy criteria takes into account the influence of all variables.

In an FMS environment, it is important to decide when to introduce a part into the system. Overloaded parts into the system may lead to congestion, thus resulting in longer production times. On the other hand, too few parts in the system result in the under-utilization of equipment. The main concern of control theoretic based methods is the release of the parts, but no detailed allocation of multiple resources such as machines, robots, buffers and material handling systems is considered.

2.1.4 Petri Net Based Methods

Petri net theory has been applied for modeling, analysis, simulation, planning, scheduling, and control of flexible manufacturing systems (Narahari and Viswanadham 1985, Hillion and Proth 1989, Viswanadham *et al.* 1990, Banaszak and Krogh 1990, Zhou, DiCesare and Desrochers 1992, Lee and DiCesare 1994). A Petri net comprises two types of nodes, namely places and transitions. A place is represented by a circle and a transition by a bar. Places and transitions are connected by arcs. In order to study dynamic behavior of

the modeled system, each place contains a non-negative integer number of tokens. At any given time instance, the distribution of tokens on places, called Petri net marking, defines the current state of the modeled system. A significant advantage of Petri net based methods is its representation capability. Petri nets can explicitly and concisely model concurrent and asynchronous activities, multi-layer resource sharing, routing flexibility, limited buffers and precedence constraints in FMS. The changes of markings in the net describe the dynamic behaviors of the system. In the methods mentioned above, very few studies investigate deadlock problems in FMS scheduling and control because they are difficult to formulate using either mathematical programming methods or control theoretic methods. Petri nets provide an explicit way for considering deadlock situations in FMSs such that a deadlock-free scheduling and control system can be designed.

A. Scheduling Method

Shih and Sekiguchi (1991) presented a timed Petri net and beam search method to schedule an FMS. Beam search is an artificial intelligence technique for efficient searching in decision trees. When a transition in a timed Petri net is enabled, if any of its input places is a conflicted input place, the scheduling system calls for a beam search routine. The beam search routine then constructs partial schedules within the beam-depth. Based on the evaluation function, the quality of each partial schedule is evaluated and the best is returned. The cycle is repeated until a complete schedule is obtained. This method based on partial schedules does not guarantee global optimization.

Shen *et al.* (1992) presented a Petri net-based branch and bound method for scheduling the activities of a robot manipulator. To cope with the complexity of the problem, they truncate the original Petri net into a number of smaller size subnets. Once the Petri net is truncated, the analysis is conducted on each subnet individually. However, due to the existence of the dependency among the subnets, the combination of local optimal schedules does not necessarily yield a global optimal or even near-optimal schedule for the original system. Zhou, Chiu and Xiong (1995) also employed a Petri net based branch and bound method to schedule flexible manufacturing systems. In their method, instead of randomly selecting one decision candidate from candidate sets (enabled transition sets in Perti net based models), they select the one based on heuristic dispatching rules such as SPT. The generated schedule is transformed into a marked graph for cycle time analysis.

Lee and DiCesare (1994) presented a scheduling method using Petri nets and heuristic search. Once the Petri net model of the system is constructed, the scheduling algorithm expands the reachability graph from the initial marking until the generated portion of the reachability graph touches the final marking. Theoretically, an optimal schedule can be obtained by generating the reachability graph and finding the optimal path from the initial marking to the final one. But the entire reachability graph may be too large to generate even for a simple Petri net due to exponential growth of the number of states. Thanks to the proposed heuristic functions, only a portion of the reachability graph is generated. Three kinds of heuristic functions are presented. The first one favors markings that are deeper in the reachability graph. The second one favors a marking which has an operation

ending soon. The last one is a combination of the first and the second ones. These three heuristic functions do not guarantee the admissible condition (Pearl 1984), thus the proposed heuristic search algorithm does not guarantee to terminate with an optimal solution. No deadlock issues are discussed in their demonstrated examples because they always put an intermediate place which serves the role of a buffer with unlimited capacity between two operations.

Hatono *et al.* (1991) employed the stochastic Petri nets to describe the uncertain events of stochastic behaviors in FMS, such as failure of machine tools, repair time, and processing time. They develop a rule base to resolve conflicts among the enabled transitions. The proposed method cannot handle the routing flexibility and deadlock situation.

B. Modeling and Discrete Event Control

For modeling and discrete event control of a flexible manufacturing system, Narahari and Viswanadham (1985) presented a systematic bottom-up approach. They obtained their Petri net model by constructing a sub-Petri net model for each machine operation and then combining these subnets by the sharing of places. The analysis of resulting Petri net such as p-invariants can be based on the analysis of subnets. To avoid the verification of a Petri net's safeness and liveness, Krogh and Back (1986) proposed another bottom-up systematic approach by introducing modified Petri nets and decomposing a manufacturing process into operations and resources. Their method leads to a safe and live Petri net model by the union of elementary circuits along common paths. The method is not

applicable to a bounded Petri net where a place with more than one token is used to model buffers and machines with processing capacity exceeding 1.

Koh and DiCesare (1991) presented modular transformation methods for generalized Petri nets by introducing and using the concept of a live and bounded circuit (LB-circuit). An LB-circuit is a generalized version of a simple elementary circuit. Three transformation theorems are presented. The first one shows that LB-circuits can be fused into a live and bounded Petri net. The second one shows that two live and bounded Petri nets can be fused along a common elementary path while preserving liveness and boundedness. The last one shows that removing LB-circuits from the original net will not changing liveness and boundedness. But the proposed modular transformation methods are not applicable for synthesizing shared resources.

Zhou, DiCesare and Desrochers (1992) presented a hybrid synthesis methodology to design a bounded, live and reversible Petri net controller. The method begins with an initial net which captures important system interactions such as choice-synchronization. This initial net should be bounded, live, and reversible. The second step is refining the places and transitions in the net in a top-down manner to reach a level which includes detailed operations of the system. The last step is adding the resources places based on proposed parallel mutual exclusion (PME) or sequential mutual exclusion (SME) structures. For the system with multi-layer resource-sharing and different products sets manufactured concurrently, modeling of a Petri net with desirable properties becomes extremely difficult based on this hybrid method.

In contrast with Zhou's method which establishes the control policy in a static way to prevent the deadlock state, another method is deadlock avoidance method in which the possible deadlocks are avoided by proper operational control. Banaszak and Krogh (1990) presented a deadlock avoidance algorithm based on Petri net model. The algorithm is a feedback policy that uses the current states of the resources and the known operation sequence for the active jobs to inhibit requests for resources when they will potentially lead to circular wait conditions. The restrictive policy, however, is not a necessary condition and is therefore overly restrictive in some cases. Multiple resource holding and alternative routing are not considered in the proposed method.

Hsieh and Chang (1994) also presented a deadlock avoidance controller synthesis method. First, a controlled production Petri net model is constructed based on the bottom-up approach. This net is then decomposed into subnets to derive a necessary and sufficient liveness condition for the net. A sufficient validity test procedure is employed to check whether the execution of a control action is valid to maintain the liveness of the net. Finally, this sufficient test procedure is combined with the given dispatching policy to generate valid control actions for the FMS.

Venkatesh, Zhou and Caudill (1994) identified certain criteria to compare ladder logic diagrams and Petri nets for sequence controller design through a discrete manufacturing system and proposed a real time Petri nets for sequence control. They show the advantages of Petri nets based control from aspects of graphical complexity and adaptability, response time, properties checking, dynamic state tracking and system initialization.

2.2 Summary

We have reviewed some of previous work in scheduling and control of flexible manufacturing systems. Typical assumptions are still confined to the classical job shop environments for most methods. There is a need for developing methods which allow the consideration of various extensions to the classical job shop models such as multiple resources sharing, multiple lot sizes, buffer availability, material handling. routing flexibility and deadlock avoidance. Among all the methods, Petri net based methods show the potential to make major contributions to FMS operation. Petri nets can be used as an integrated tool for modeling, scheduling, control and performance analysis of flexible manufacturing systems. Petri nets can explicitly and concisely model the concurrent and asynchronous activities, multi-layer resource sharing, part contact states (loading/unloading), routing flexibility, limited buffers and precedence constraints in flexible manufacturing systems. Petri nets can also provide an explicit way for considering deadlock situations in FMSs, and thus facilitate the design of a deadlock-free scheduling and control system. Therefore, we investigate the scheduling and control of flexible manufacturing systems based on Perti nets in this research.

CHAPTER 3

MODELING MANUFACTURING SYSTEMS WITH PETRI NETS

Petri nets were named after Carl A. Petri who created in 1962 a net-like mathematical tool for the study of communication with automata. The further development made Petri nets become a promising graphical and mathematical modeling tool applicable to many systems that are characterized as being concurrent, asynchronous, distributed, parallel, nondeterministic, and/or stochastic (Murata 1989). Petri nets have been used extensively to model and analyze manufacturing systems. A recent overviews of applications of Petri nets in manufacturing areas can be seen in [Zurawski and Zhou 1994] [David and Alla 1994]. In this chapter, the fundamentals of Petri nets and their modeling methods in manufacturing systems are introduced to facilitate presentations of our research results. For more detail, the reader is referred to [Peterson 1981], [Murata 1989], [Zhou and DiCesare 1993], [Zurawski and Zhou 1994] and [David and Alla 1994].

3.1 Concepts and Properties of Petri Nets

A Petri net is defined as a bipartite directed graph containing places, transitions, and directed arcs connecting places to transitions and transitions to places. Pictorially, places are depicted by circles and transitions as bars or boxes. A place is an input place to a transitions if there exists a directed arc connecting this place to the transition. A place is an output place of a transition if there exists a directed arc connecting the transition to the place. Places contain tokens pictured by black dots. Each place may potentially hold either

none or a positive number of tokens. At any given time instance, the distribution of tokens on places, called Petri net marking, defines the current state of the modeled system. Thus a marked Petri net can be used to study dynamic behavior of the modeled discrete event systems.

Formally, a Petri net can be defined as PN = (P, T, I, O, M_0); where

- P = $\{p_1, p_2, \cdots, p_m\}$, $m > 0$ is a finite set of places;
- T = $\{t_1, t_2, \cdots, t_n\}$, $n > 0$ with P$\cup$T $\neq \varnothing$ and P$\cap$T = $\varnothing$ is a finite set of transitions;
- I: P×T $\rightarrow$ {0,1} is an input function or direct arcs from P to T;
- O: P×T $\rightarrow$ {0,1} is an output function or direct arcs from T to P;
- M: P $\rightarrow$ {0,1,2,$\cdots$} is a |P| dimensional vector with M(p) being the token count of place p. M_0 is an initial marking.

The behavior of many systems can be described in terms of systems states and their changes. In order to simulate the dynamic behavior of a system, a state or marking in a Petri net is changed according to the following transition (firing) rules:

(1) A transition t is enabled if $M(p_i) \geq I(p_i,t)$ for any $p_i \in$P.

(2) An enabled transition t can fire at marking M', and its firing yields a new marking,

$M(p) = M'(p) + O(p,t) - I(p,t)$, for arbitrary p from P.

The marking M is said to be reachable from M'. Given PN and its initial marking M_0, the reachability set is the set of all marking reachable from M_0 through various sequences of transition firings and is denoted by R(PN, M_0). Reachability set is a fundamental basis

for studying the dynamic properties of a system. For a marking $M \in R(PN, M_0)$, if no transition is enabled in M, than M is called a deadlock marking.

A pair of a place p and a transition t is called a self-loop if p is both an input and output place of t. A Petri net is said to be pure if it has no self-loops. A Petri net containing self-loops can be made pure by adding dummy places and transitions. The dynamic behavior of pure Petri nets can also be represented by matrix equations. The incidence matrix defines all interconnections between places and transitions in a Petri net. For a pure Perti net with m places and n transitions, the incidence matrix $C = O - I$ is an $m \times n$ matrix of integers. The entries of the incidence matrix are defined as follows: $c_{ij} = O(p_i,t_j) - I(p_i,t_j)$, where $O(p_i,t_j)$ is equal to the number of arcs connecting transition t_j to its output place p_i, and $I(p_i,t_j)$ is equal to the number of arcs connecting transition t_j to its input place p_i. When transitions t_j fires, $O(p_i,t_j)$ represents the number of tokens deposited on its output place p_i, $I(p_i,t_j)$ represents the number of tokens removed from its input place p_i, c_{ij} represents the change in the number of tokens in place p_i. Transition t_j is enabled at a marking M if

$$I(p_i,t_j) \leq M(p_i), \; i = 1, 2, \cdots, m.$$

The state equation for a Petri net represents a change in the distribution of tokens on places as a result of a transition firing. Since the jth column of the incidence matrix C denotes the change of the marking as a result of a firing transition t_j, the state equation is defined as follows:

$$M_k = M_{k-1} + Cu_k, \; k = 1, 2, \cdots .$$

M_k is an $m \times 1$ column vector representing a marking M_k immediately reachable from a marking M_{k-1} after firing transition t_j. The control vector (kth firing vector) u_k is an $n \times 1$ column of n – 1 0's and one nonzero entry, a 1 in the jth position indicating that transition t_j fires at the kth firing.

Petri nets as mathematical tools possess a number of properties. Some of the important properties are as follows.

A Petri net (PN, M_0) is said to be K-bounded or simply bounded if the number of tokens in each place does not exceed a finite number K for any marking reachable from M_0. A Petri net (PN, M_0) is said to be safe if it is 1-bounded. For bounded Petri net, from the initial marking M_0, there are a limited number of reachable markings which are obtainable via various sequence of transition firing.

A Petri net (PN, M_0) is said to be live if, no matter what marking has been reached from M_0, it is possible to ultimately fire any transition of the net by progressing through some further firing sequence. This means that a live Petri net guarantees deadlock-free operation, no matter what firing sequence is chosen.

A Petri net (PN, M_0) is said to be reversible if, for each marking m in R(PN, M_0), M_0 is reachable from M. Therefore, in a reversible net one can always get back to the initial marking.

The boundedness, liveness, and reversibility of Petri nets have their significance to manufacturing systems. Boundedness or safeness implies the absence of capacity overflows. Liveness implies the absence of deadlocks. This property guarantees that a system can successfully produce without being deadlocked. Reversibility implies that

cyclic behavior of a system and repetitive production in flexible manufacturing. It means the system can be initialized from any reachable state.

Among subclasses of Petri nets, there is a choice-free or conflict-free net called the marked graph. A marked graph is a PN (P, T, I, O, M_0) such that $\forall p \in P$, $t \in T$, $I(p,t) \leq 1$, $O(p,t) \leq 1$, and given any $p \in P$, $|\{ t \in T : O(p,t) = 1\}| = 1$, and $|\{ t \in T : I(p,t) = 1\}| = 1$.

The presence of the conflict structures (a structure involving a place having two, or more output transitions) in a Petri net requires a conflict resolution mechanism to select one transition to fire. Since this mechanism is, typically, based on a probabilistic function, the net becomes stochastic. While in a marked graph, each place has exactly one input transition and exactly one output transition, thus no conflict is possible. For this reason, among models that can represent concurrent activities, marked graphs are the most amenable to analysis.

In our research, we synthesize a marked graph as a discrete event controller based on a derived optimal event sequence. The important properties of marked graphs are presented in Chapter 4.

The ordinary Petri nets do not include any concept of time and only describe the logical structure of the modeled system. A timed Petri net enables a system to be described whose functioning is time dependent. For example, a certain time may elapse between the start and the end of an operation. If a mark in a certain place indicates that this operation is in progress, a timed Petri net enables this time to be taken into account.

Ramchandani (1973) first introduced Timed Petri nets (TPN's) by associating firing times to the transitions of ordinary Petri nets to study their steady-state behavior. Since then many researchers have reported work on deterministic or stochastic TPN models. For modeling of production systems, deterministic TPN is appropriate if the working time of a machine to treat a part is constant, while if we should consider the situation of failure of machine tools, stochastic TPN may be used because the duration of proper function (between two breakdowns) of a machine is random. Except for associating firing times to the transitions (T-timed), the timings can also be associated with the places (P-timed), or both.

For a P-timed Petri net, a timing d_i, possibly of zero value, is associated with each place p_i. When a token is deposited in place p_i, this token must remain in this place at least for a time d_i. This token is said to be unavailable for this time. When the time d_i has elapsed, the token then becomes available. Only available tokens are considered for enabling conditions. For a T-timed Petri net, a timing d_j, possibly of zero value, is associated with each transition t_j. When a transition t_j fires, the tokens removed from its input places to its output places are reserved for a time d_j. After elapsing this time, the reserved tokens become non-reserved tokens and can be considered for enabling conditions.

In this research, we use deterministic P-timed Petri nets modeling FMS for scheduling. The transitions in such nets can fire with a zero duration, which is consistent with the non-timed or ordinary definition of Petri nets. However it is always possible to transform a P-timed Petri net to a T-timed one, and vice versa.

The timed Petri nets, especially timed marked graphs, are very useful for performance analysis of modeled systems. The performance evaluation of marked graphs will be discussed in Chapter 4.

3.2 Petri Net Modeling

We consider discrete-parts manufacturing systems in which individual parts are clearly distinguishable. A manufacturing process is a set of activities which interact with a set of resources. The product process plan specifies a sequence of operations for processing a job by the system. The manufacturing system can manufacture multiple products of the same product type and can also concurrently manufacture products of multiple types.

3.2.1 Modeling Methods

Generally in Petri net modeling, places represent conditions and transitions represents events. In our approach for modeling manufacturing systems with Petri nets, a place represents a resource status or an operation, a transition represents either start or completion of an event or operation process, and the stop transition for one activity will be the same as the start transition for the next activity. Token(s) in the resource place indicates that the resource is available and no token indicates that it is not available. A token in the operation place represents that the operation is being executed and no token shows none being performed.

Example 3.1: Figure 3.1 shows a simple Perti net model. A robot unloads two kinds of parts from two intermediate buffers to an output station. The robot unloads a part from

either Buffer 1 or Buffer 2. As soon as the unloading is over, another part is made available and the robot randomly unloads a part again. The interpretation of places and transitions is shown in Table 3.1.

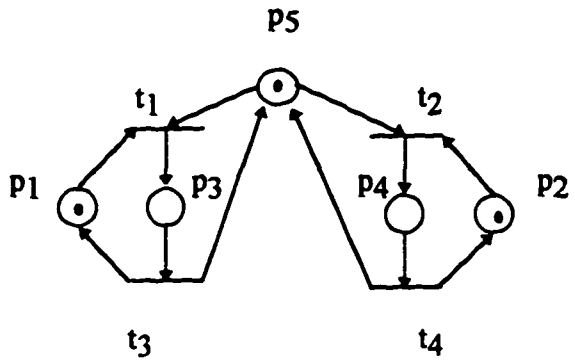


Figure 3.1 A Petri net model for a robot unloading parts in Example 3.1

Table 3.1: Interpretation of places and transitions in Figure 3.1

Places	Transitions
p_1: A part 1 on buffer for unloading	t_1: Unloading part 1 starts
p_2: A part 2 on buffer for unloading	t_2: Unloading part 2 starts
p_3: The part 1 being unloaded	t_3: Unloading part 1 ends
p_4: The part 2 being unloaded	t_4: Unloading part 2 ends
p_5: The robot ready to unload a part	

In Figure 3.1, places p_1, p_2 and p_5 model resource availability status. The marked resource place indicates the representing resource is available, and unmarked indicates unavailability. Places p_3 and p_4 model operations. Transitions t_1 and t_2 represent the starting of the operations. Transitions t_3 and t_4 represent the ending of the operations.

For this Petri net, P = {p_1, p_2, p_3, p_4, p_5}, T = {t_1, t_2, t_3, t_4}. The initial marking M_0 = $(1, 1, 0, 0, 1)^T$.

In the initial marking, both transitions t_1 and t_2 are enabled. If t_1 fires, the marking M_1 = $(0, 1, 1, 0, 0)^T$ is reached. If t_2 fires, the marking M_2 = $(1, 0, 0, 1, 0)^T$ is reached. At either M_1 or M_2 , only one transition is enabled, t_3 or t_4. Firing either of them leads the net to its initial marking. This Petri net is safe, live, and reversible based on the definitions of safeness, liveness, and reversibility.

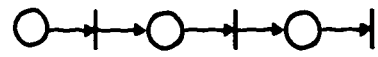
A certain order of activities needs to be followed by each job in manufacturing systems. For example, the activity sequence {operation 1, operation 2} should be followed by each job. Therefore, for Petri net modeling, the first important issue is the modeling of sequential activities for each job in the system.

The second modeling issue is synchronization. For example, Machine 1 will process material piece 1 only when it is present. It will never finish the process operation if the material is missing.

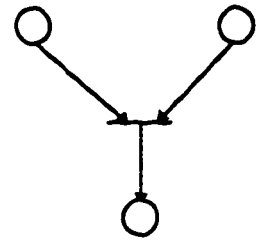
The third issue is modeling of concurrence. By concurrence we mean that there are parallel relationships among the concerned events. For example, two physical events 1) Machine 1 processes the first operation of Job 1, 2) Machine 2 processes the second operation of Job 2, are concurrent if both events may occur simultaneously. Two machines can operate concurrently if both can process tasks at the same time. High concurrency among system resources often implies high productivity.

The fourth modeling issue we are concerned with is conflict, when the sharing of resources is encountered. In this case, if two or more jobs require one shared resource at the same time, only one job can get the required resource.

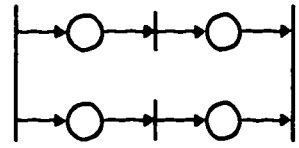
The Petri net models must take the various issues as discussed above into consideration. The usual approach is to create a Petri net model with which to analyze critical properties of interest. A more rigorous approach for Petri net modeling is to synthesize a Petri net of a system which has desirable properties such as boundedness and deadlock freeness. Examples of Petri net models for linear sequence, synchronization, concurrency, and mutual exclusion are shown in Figure 3.2.



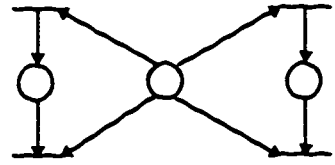
(a)



(b)



(c)



(d)

Figure 3.2 Examples of Petri net models for (a) linear sequence, (b) synchronization, (c) concurrency, and (d) mutual exclusion

Previous research on the Petri net modeling methodology can be summarized into three basic approaches: bottom-up (Agerwala and Choed-Amphai 1978), top-down (Valette 1979) and hybrid (Zhou, DiCesare and Desrochers 1992). A review of synthesis techniques for Petri nets with applications to manufacturing systems can be seen in [Jeng and DiCesare 1993].

In this research, the bottom-up method is used to synthesize the system for scheduling. First, the system is partitioned into sub-systems according to the job types, then sub-models are constructed for each sub-system, and a complete net model for the entire process is obtained by merging Petri nets of the sub-systems through the places representing the shared resources. For each sub-system (job type), a Petri net is constructed based on the following steps (Zhou and DiCesare 1993):

(1) Identify the operations and resources (machines/buffers) required;

(2) Order operations by the precedence relations if they exist;

(3) For each operation in order, create and label a place to represent its status, add a transition (start activity) with an output arc(s) to the places, add a transition (stop activity) with an input arc(s) from the places;

(4) For each kind of resources (machines/buffers), create and label a place. If an operation place is a starting activity to require the resource(s), add input arc(s) from that resource place to the starting transition of that operation. If an operation is the ending one to use the resources, add output arc(s) from the ending transition to the resource place(s);

(5) Specify the initial marking, and associate the timings with the operation places.

3.2.2 Petri Net Modeling for Scheduling

Let us take an example to illustrate Petri net modeling for scheduling.

Example 3.2: An FMS has two machine M_1, M_2 and one robot R. There are two jobs J_1 and J_2 which have two processes each. Table 3.2 shows the job requirements.

Table 3.2 Job Requirements for Example 3.2

Operations/Jobs	J_1	J_2
1	(M_1R, 4)	(M_1,1)
2	(M_2R, 1)	(M_2,4)

The first operation of Job 1 can be carried out at Machine 1 and needs 4 unit time. The second operation of Job 1 can be carried out at Machine 2 and needs 1 unit time. The first operation of Job 2 can be carried out at Machine 1 and needs 1 unit time. The second operation of Job 2 can be carried out at Machine 2 and needs 4 unit time. Both the first and second operations of Job 1 need the robot for holding. The size of the intermediate buffer for each job is 1. Figure 3.3 shows the Petri net model of sub-system Job 1 and Figure 3.4 shows the Petri net model of sub-system Job 2. The Petri net model for the whole system is obtained by merging the places representing Machine 1 and Machine 2 in two sub-models and shown in Figure 3.5. The interpretation of places and transitions is shown in Table 3.3.

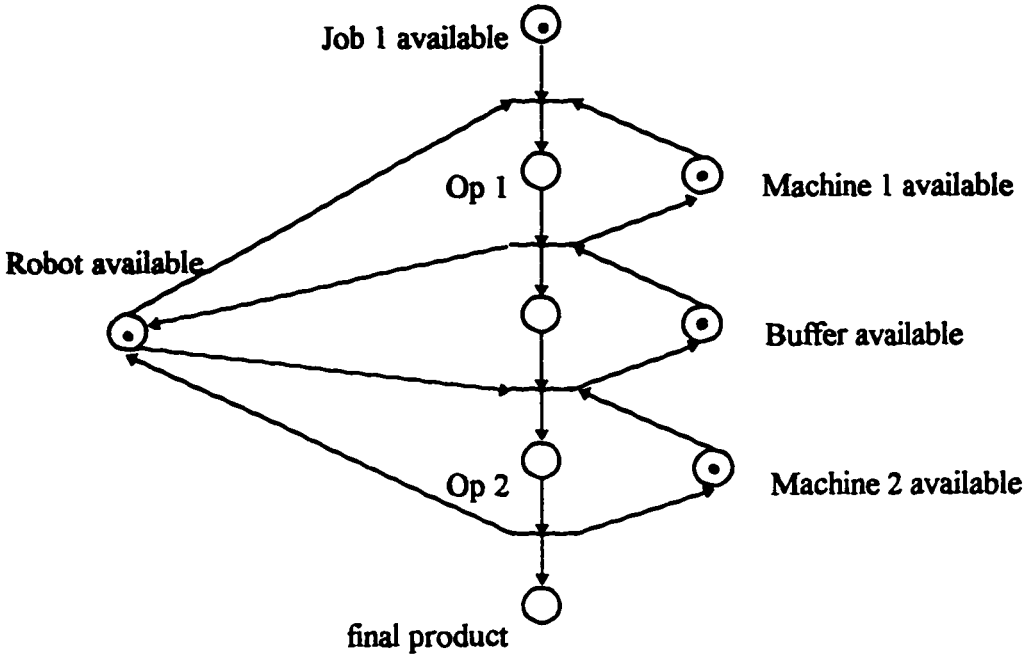


Figure 3.3 The Petri net model for sub-system of Job 1 in Example 3.2

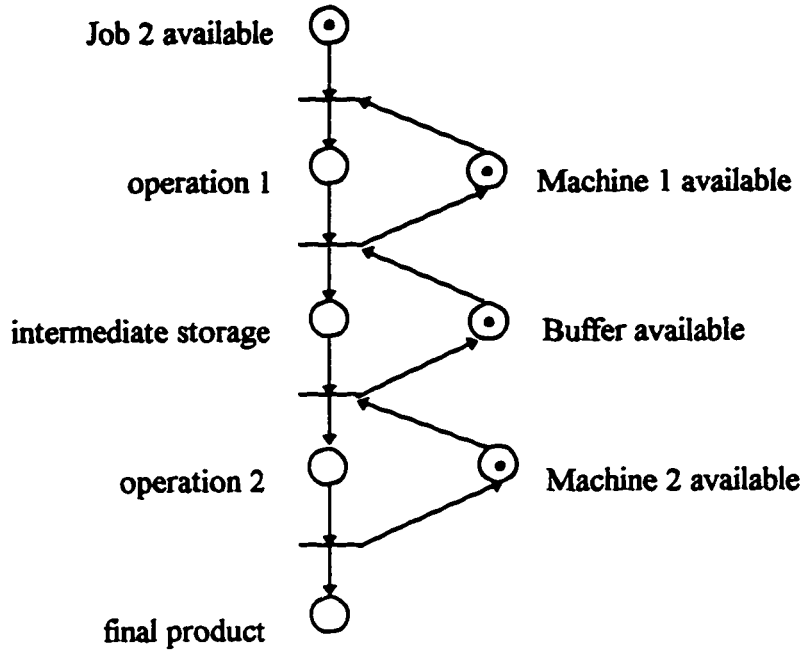


Figure 3.4 The Petri net model for sub-system of Job 2 in Example 3.2

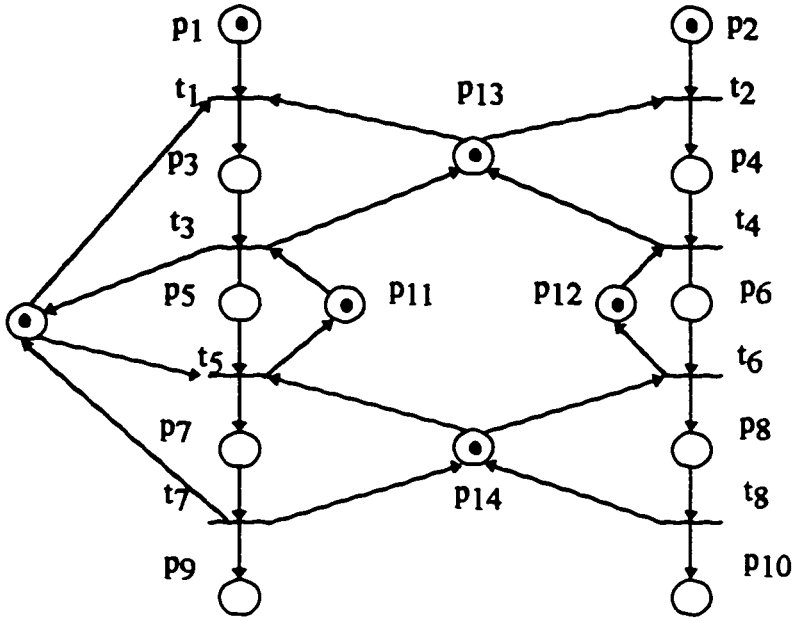


Figure 3.5 The whole Petri net model for Example 3.2

Table 3.3: Interpretation of places and transitions in Figure 3.4

Places	Transitions
p_1: Job 1 available	t_1: Operation 1 of Job 1 starts
p_2: Job 2 available	t_2: Operation 1 of Job 2 starts
p_3: Operation 1 of Job 1	t_3: Operation 1 of Job 1 finishes
p_4: Operation 1 of Job 2	t_4: Operation 1 of Job 2 finishes
p_5: Job 1 ready for the second operation	t_5: Operation 2 of Job 1 starts
p_6: Job 2 ready for the second operation	t_6: Operation 2 of Job 2 starts
p_7: Operation 2 of Job 1	t_7: Operation 2 of Job 1 finishes
p_8: Operation 2 of Job 2	t_8: Operation 2 of Job 2 finishes
p_9: Final product of Job 1	
p_{10}: Final product of Job 2	
p_{12}: Buffer of Job 1 available	
p_{12}: Buffer of Job 2 available	
p_{13}: Machine 1 available	
p_{14}: Machine 2 available	
p_{15}: Robot available	

The evolution of the system can be completely tracked by the reachability graph of the Petri net. Figure 3.6 shows a partial portion of the reachability graph for the Petri net model shown in Figure 3.5. In the reachability graph, both transition firing sequences of $t_1t_3t_2t_5t_4t_7t_6t_8$ and $t_2t_4t_1t_6t_3t_8t_5t_7$ give a path from the initial marking to the final marking. But they generate different performance of schedules. Figure 3.7(a) shows the schedule generated from transition firing sequence $t_1t_3t_2t_5t_4t_7t_6t_8$ with a makespan of 9. Figure 3.7(b) shows the schedule generated from transition firing sequence $t_2t_4t_1t_6t_3t_8t_5t_7$ with a makespan of 6. The notation $O_{i,j,k}$ in Figure 3.7 represents the j-th operation of the i-th job being performed at the k-th machine. Furthermore, if the lot size of Job 1 is 2, i.e., there are two tokens in the place p_1 in the initial state, the transition firing sequence $t_2t_4t_1t_6t_3t_1t_8$ leads the system into a deadlock state in which further part flow is inhibited. Figure 3.8 shows the evolution of the system states in terms of changes in the marking of the Petri net for the transition firing sequence $t_2t_4t_1t_6t_3t_1t_8$ which leads the system into a deadlock state.

Therefore, the main purpose of this dissertation is to investigate deadlock-free scheduling and control of flexible manufacturing systems by using Petri nets as a modeling framework.

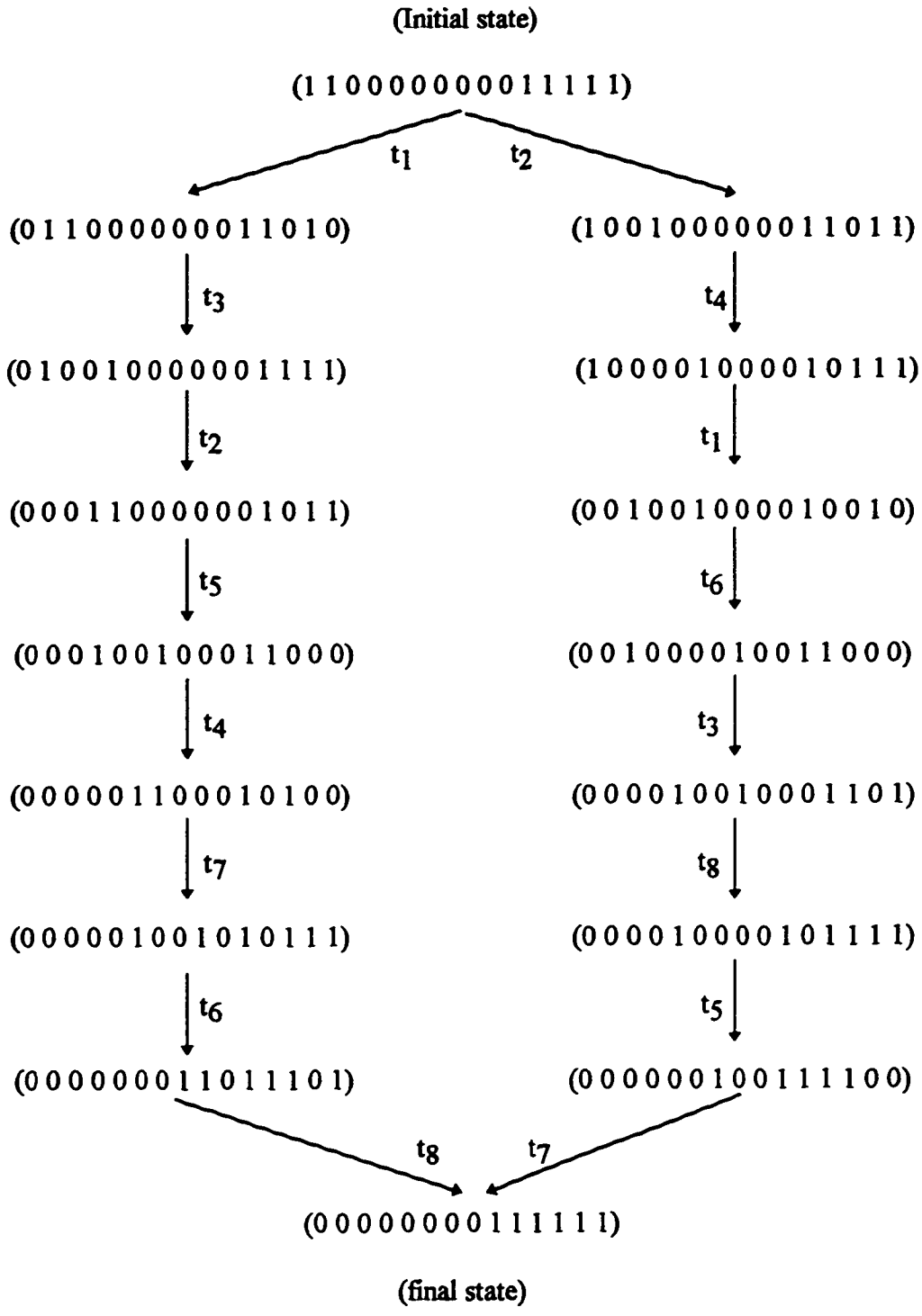
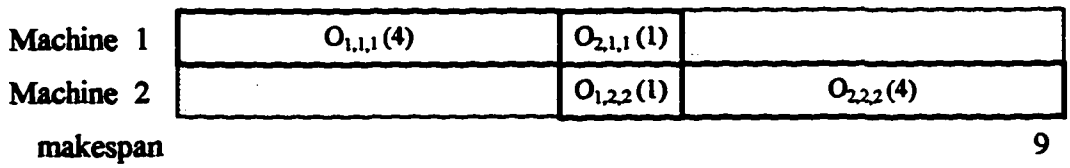
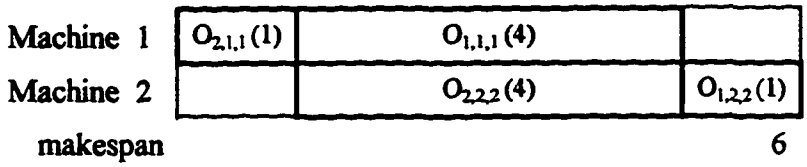


Figure 3.6 A partial portion of the reachability graph for the Petri net model shown in Figure 3.5 of Example 3.2



(a) transition firing sequence $t_1t_3t_2t_5t_4t_7t_6t_8$



(b) transition firing sequence $t_2t_4t_1t_6t_3t_8t_5t_7$

Figure 3.7 Schedules represented by two different transition firing sequences of Example 3.2

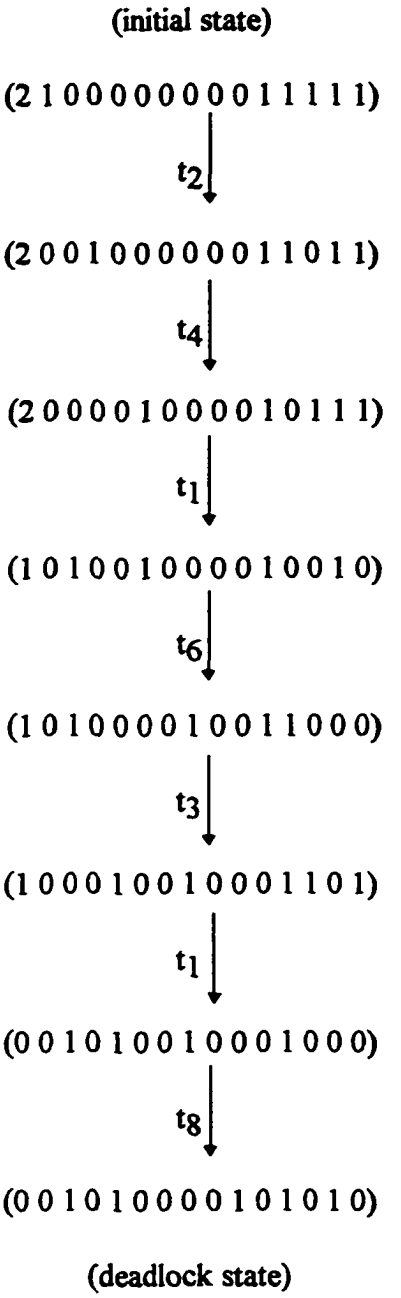


Figure 3.8 The evolution of the system states for the transition firing sequence $t_2t_4t_1t_6t_3t_1t_8$ which leads the system into a deadlock state in Example 3.2

CHAPTER 4

OPTIMIZATION OF DISCRETE EVENT CONTROLLER DESIGN

4.1 Introduction

For discrete event control of FMSs, an optimal control problem is to find an input event sequence that moves the system from a given initial state to a given final state while minimizing certain performance indices. Various notions of optimal control have been studied for discrete event systems (DESs). Passino and Antsaklis (1989) used valid behavior model and allowable behavior to describe DESs and proposed a metric space approach to heuristic search for an optimal solution. Lin and Ionescu (1992) considered optimization of controller design for discrete event systems in a temporal logic framework. Sengupta and Lafortune (1991) proposed graph-theoretic formulation of optimal discrete event control problems for a class of DESs.

Petri net theory has been applied for scheduling and discrete event control of flexible manufacturing systems. Petri nets can concisely model the concurrent and asynchronous activities, resource sharing, and precedence constraints in FMSs. Venkatesh, Zhou and Caudill (1994) identified certain criteria to compare ladder logic diagrams and Petri nets for sequence controller design through a discrete manufacturing system and proposed a real-time Petri net for sequence control. Zhou, DiCesare and Desrochers (1992) presented a hybrid synthesis methodology to design a bounded, live and reversible Petri net controller. But for the system with multi-layer resource-sharing and different product sets manufactured concurrently, modeling of a Petri net with desirable properties becomes

extremely difficult based on this hybrid method. Hillion and Proth (1989) used timed event-graphs, a special class of timed Petri nets, for modeling and analyzing job-shop systems. Sayat and Ladet (1993) employed colored Petri nets and Grafcet to describe different levels of production control to deal with different levels of complexity presenting at each level. Lee and DiCesare (1994) presented a Petri net-based heuristic scheduling method for flexible manufacturing, although it does not guarantee to terminate with an optimal solution.

The goal of this chapter is to formulate and solve the optimal discrete event controller synthesis problem for a flexible manufacturing system in a timed Petri net framework. The bottom-up method is used to model the system. Once the modeling is done, the A* based heuristic search algorithm which is combined with the execution of the timed Petri net is proposed to search for an optimal event sequence to achieve minimum-time discrete event control. Based on the obtained event-driven sequence, we use two levels of specification to design the optimal sequence controller for the presented FMS. The coordination control level consists of synchronization and parallelism of different sub-systems and is specified by decision-free Petri nets (marked graphs). The local control level consists of running elementary sequences for sub-systems, which are specified by the Sequential Function Charts (SFCs). The relation between two levels is realized by the logical conditions associated with some transitions in the coordination model and local control models. The specific objectives of this chapter are:

1. To present a design method for the synthesis of a optimal discrete event controller.

2. To introduce an A* based heuristic search algorithm for seeking the optimal event sequence based on the reachability graph of Petri nets.
3. To illustrate the design method through a flexible manufacturing system.
4. To develop theoretical results to insure the desired qualitative properties of boundedness (safeness), liveness, and reversibility in the resulting Petri net controller.
5. To evaluate the performance of the controller and make comparisons with the ones driven by dispatching rules.
6. To analyze the controller's sensitivity to randomness.

4.2 Design Method for Discrete Event Control

Due to its complexity, the control of a flexible manufacturing system is commonly decomposed into a hierarchy of decision levels, such as planning, scheduling, supervisory control, and local control. The discussion in this chapter focuses on optimal sequence control problem in FMSs at the levels of scheduling, supervisory and local control. The optimal control problem is to find an input event sequence that moves the system from a given initial state to a final state while minimizing certain performance indices. Based on the optimal event sequence, a sequence controller is designed for optimization of system performance.

4.2.1 Description of a Design Procedure

Figure 4.1 shows a two-level functional structure of sequence control. It is assumed that a host computer is responsible for coordination and synchronization of different sub-systems, such as machines, robots and AGVs. The control sequence implemented at this level is an optimal event sequence and can be specified by a decision-free Petri net (marked graph). The local control level consists of running elementary sequences for sub-systems. The sequence of operations executed by a local controller is specified by a sequential function chart (SFC) from which the controller program code, such as the relay ladder logic program, can be directly derived and implemented into a Programmable Logical Controller (PLC).

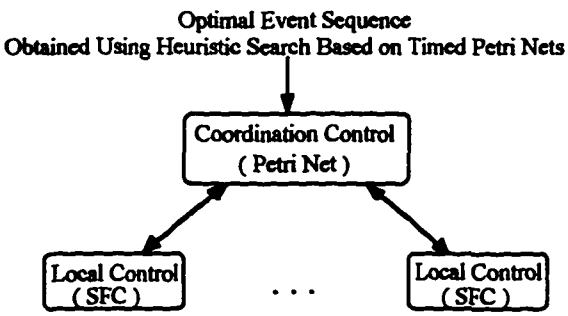


Figure 4.1 The sequence control structure

The design procedure for optimal sequence controllers is proposed as follows:

Step 1. Modeling of an FMS using timed Petri nets. The synthesis of Petri net models is based on a bottom-up approach which begins with the construction of subnets for component processes and proceeds to the final net by merging and/or linking all these subnets. The concurrency, conflicts, resource-sharing, and sequential operations are concisely represented in a Petri net model.

Step 2. Heuristic search of the reachability graph of a timed Petri net model for an optimal or near-optimal event sequence. All feasible event sequences are incorporated in the reachability graph of the Petri net model resulted from *Step 1*. The search for an optimal event sequence is NP-complete. Therefore, the heuristic search methods are employed to reduce computational effort.

Step 3. Synthesis of a choice-free Petri net model (marked graph) for event-driven coordination control based on the optimal event sequence. The event sequence obtained from *Step 2* optimally resolves the conflicts competing for shared resources among the processes. As a result, the system behavior can be described by a marked graph in which each place has exactly one input and one output transition. A marked graph is guaranteed to be live if and only if every circuit contains at least one token. This greatly reduces the analytical overhead for eliminating the deadlock states in the system. Therefore, compared with existing Petri net or other methods (Banaszak and Krogh 1990, Narahari and Viswanadham 1990, Zhou, DiCesare and Desrochers 1992, Wysk *et al.* 1994), real-time control implementation of a marked graph can easily guarantee deadlock-free system behavior. Moreover, there exist effective methods for performance analysis of timed

marked graphs (Ramamoorthy and Ho 1980, Hillion and Proth 1989) and performance bounds when the operation times suffer for randomness.

Step 4. Specification of local sequence controllers for each sub-system using sequential function charts. SFC is an industrial standard for describing the control logic of manufacturing devices (David and Alla 1992). It overcomes two drawbacks inherent in Petri nets: nondeterministic evolution and infinite creation of tokens. In SFC, transition firing is synchronous, and a step can only be active or inactive (binary state), as discussed in more detail later.

Note that a marked graph is generated in Step 3. The present method admits only sequential production processes, i.e., no routing flexibility.

4.2.2 Petri Net Modeling

For a given system, we construct its Petri net model based on the bottom-up method. A system is partitioned according to the job types, then a sub-model is constructed for each job type, and finally a complete net model for the entire system is obtained by merging Petri nets of job types through the places representing the shared resources. When an FMS consists of many machines and can deal with many types of jobs, modeling of a Petri net based on the above synthesis method cannot guarantee the liveness of the model. Let us illustrate it through a simple example which is depicted in Figure 4.2. The system consists of a robot, a machine and a load/unload station at which raw parts are always available. The robot loads a raw part from a loading station to machine, which carries out some

operations on the raw part. The finished part is unloaded by the robot from the machine to the unloading station. The Petri net model is shown in Figure 4.3.

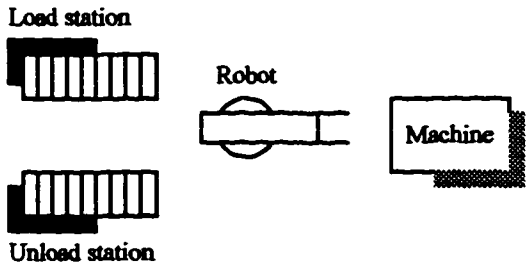


Figure 4.2 A system comprising a robot and a machine

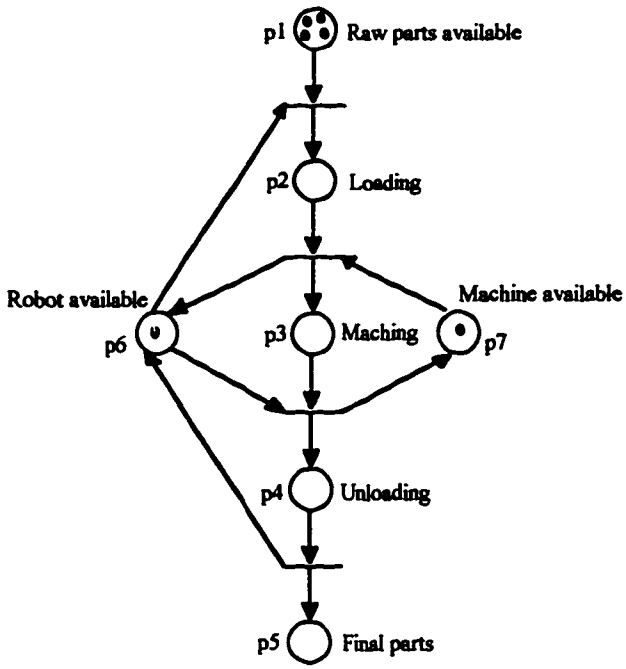


Figure 4.3 The Petri net model for the system depicted in Figure 4.2

Suppose that the initial marking is (4,0,0,0,0,1,1), i.e., both the machine and robot are available and there are four raw parts in the load station. Execute the following sequence of events starting with the initial state:

1) Robot carries a raw part from the load station;

2) Robot loads the part onto Machine and is released;

3) Machine starts operations on raw part;

4) Robot carries another raw part from the load station;

5) Machine finishes the operations on the first raw part and waits for Robot for unloading.

At this instant, Machine requests Robot for unloading and Robot waits for Machine for releasing the held parts. The marking is (2,1,1,0,0,0,0), which is a deadlock state. At this state, no further actions can occur.

A firing sequence of the transitions from an initial marking to a final marking can be obtained by searching for it over the reachability graph of the Petri net model if it exists. The sequence is then used to synthesize a decision-free and deadlock-free Petri net model for supervisory coordination control.

4.2.3 Sequential Function Chart

Sequential function chart or Grafcet (IEC, 1990, David and Alla 1992) was proposed to describe the functioning of logic controllers and their specification, and accepted as an international standard in 1990. Compared with Petri nets, SFC clearly represents inputs, outputs and their relations, and is appropriate for specifying a local logic controller which

consists of running elementary sequences. A sequential function chart consists of steps, transitions, and arcs. A step represents a partial state of the system and may be active or inactive. Actions are associated with the steps. The associated action is performed when the step is active, and remains idle when the step is inactive. A transition separates two successive steps, associated with a receptivity consisting of a logic condition or an external event, or an event and a condition. A transition is firable if and only if all the steps preceding the transition are active and the receptivity of the transition is true.

Figure 4.4 shows some basic design modules, sequential actions, synchronous actions and asynchronous actions, for SFC.

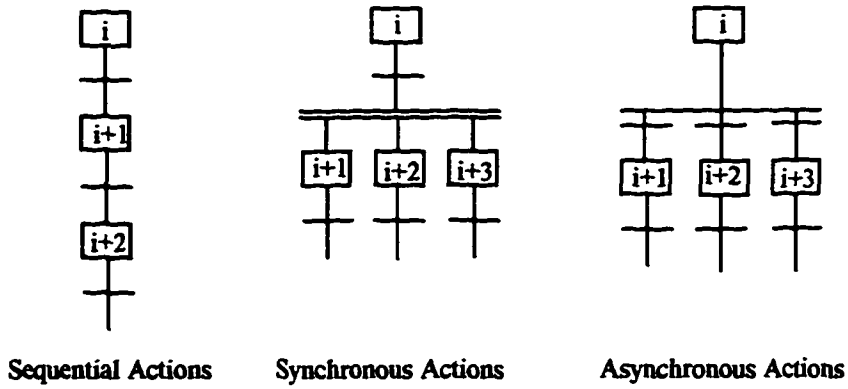


Figure 4.4 Some basic design modules of SFC

4.3 Heuristic Algorithm for Optimization of Event Sequence

For discrete event systems, an optimal control problem is to find an input event sequence that moves the system from a given initial state to a final state while optimizing a pre-defined performance index. Based on the obtained optimal event sequence, a sequence controller can be designed.

An optimal event sequence is sought in a timed Petri net framework to achieve minimum-time control. In the Petri net model of a system, firing of an enabled transition changes the token distribution (marking). A sequence of firings results in a sequence of markings, and all possible behaviors of the system can be completely tracked by the reachability graph of a net. The search space for the optimal event sequence is the reachability graph of the net, and the problem is to find a firing sequence of the transitions in the Petri net model from the initial marking to the final one. A heuristic search algorithm is developed by combining the Petri net execution and a best-first graph search algorithm A* (Pearl 1984). The most important aspect of the algorithm is the elimination from further consideration of some subsets of markings which may exist in the entire reachability graph. Thus the amount of computation and the memory requirements are reduced.

Algorithm 4.1:

1. Put the start node (initial marking) m_0 on OPEN.
2. If OPEN is empty, exit with failure.

3. Remove from OPEN and place on CLOSED a marking m for which f is the minimum.
4. If m is a goal node (final marking), exit successfully with the solution obtained by tracing back the pointers from m to m_0.
5. Otherwise find the enabled transitions at m, generate the successor markings for each enabled transition, and attach to them pointers back to m.
6. For every successor marking m' of m:

(a) Calculate $f(m')$.

(b) If m' was neither on OPEN nor on CLOSED, add it to OPEN. Assign the newly computed $f(m')$ to marking m'.

(c) If m' already resided on OPEN or CLOSED, compare the newly computed $f(m')$ with the value previously assigned to m'. If the old value is lower, discard the newly generated marking. If the new value is lower, substitute it for the old and direct its pointer along the current path. If the matching marking m' resided on CLOSED, move it back to OPEN.

7. Go to step 2.

The function $f(m)$ in Algorithm 4.1 is the sum of two terms $g(m)$ and $h(m)$. $f(m)$ is an estimate cost (makespan) from the initial marking to the final one along an optimal path which goes through the marking m. The first term, $g(m)$, is the cost of a firing sequence from the initial marking to the current one. The second term, $h(m)$ is an estimate cost of a

firing sequence from current marking m to the final marking, called *heuristic function*. The following heuristic function is used:

$$h(m) = \max_i \{ \xi_i(m), i = 1, 2, ..., N. \}$$

where $\xi_i(m)$ is the sum of operation times of those remaining operations for all jobs which are planned to be processed on the ith machine when the current system state is represented by marking m. N is the total number of machines. The purpose of a heuristic function is to guide the search process in the most profitable direction by suggesting which transition to fire first.

For the above heuristic function, $h(m)$ is a lower bound to all complete solutions descending from the current marking, i.e.,

$$h(m) \leq h^*(m), \forall m$$

where $h^*(m)$ is the optimal cost of paths going from the current marking m to the final marking. Hence, the employed heuristic function $h(m)$ is admissible, which guarantees for an optimal solution (Pearl 1984).

The list OPEN maintains markings that have been generated and had the heuristic function applied to them. It chooses which marking to expand next based on the combination of how good the marking itself looks as measured by $h(m)$ and how good the path to the marking is as measured by $g(m)$. If the newly generated marking is already on OPEN, it means a new firing sequence (path) to this marking from initial marking has been found. The path is updated to yield the smallest cost whenever the new path has a cost lower than the old path.

The list CLOSED maintains markings that have already been examined. When a new marking is generated, it is checked whether the marking has been generated before. If the newly generated marking is on CLOSED and the new path has a cost lower than the old path, this marking is put in OPEN for re-exploration.

At each step of the best-first search process, the most promising of the markings generated so far is selected. The reachability graph grows from the initial marking until it touches the final one. Because of the heuristic function, only portions of the reachability graph are generated. The more informed a heuristic function is, the smaller the number of generated markings is.

4.4 Illustration Through a Flexible Manufacturing System

Example 4.1: The design procedure presented in Section 4.2 is illustrated through an FMS. The layout of a flexible manufacturing system is shown in Figure 4.5. It consists of two entries, two exits, three machines, three robots, and a two AGV system. Two job (product) types J_1 and J_2 are to be carried out. The precedence relationships among the operations and operational time of each operation on the assigned machine for each job are shown in Table 4.1.

Table 4.1 Job Requirements of Example 4.1

Operation/Job	J_1	J_2
1	$(M_1,5)$	$(M_3,7)$
2	$(M_2,8)$	$(M_1,3)$
3	$(M_3,2)$	$(M_2,9)$

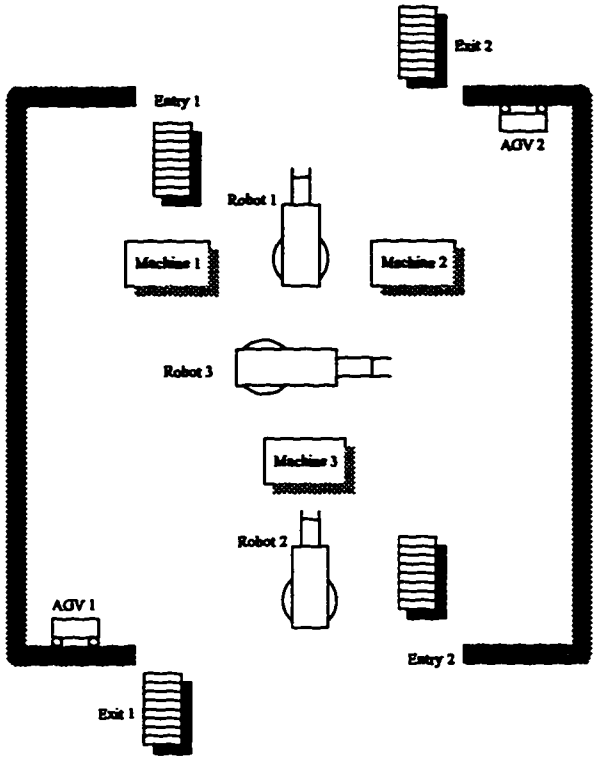


Figure 4.5 The layout of a flexible manufacturing system in Example 4.1

Entries: There are two entries, Entry 1 and Entry 2, for two types of raw materials which are made into two different kinds of products J_1 and J_2 respectively. Each raw material piece is fixtured to a pallet so that it can be transferred using robots and the AGV system. Both products J_1 and J_2 have one pallet in the system and an unlimited source of raw material is assumed.

Exits: There are two exits, Exit 1 and Exit 2, for finished products J_1 and J_2 respectively.

Machines: The first operation of J_1 is carried out at machine M_1, the second and third are carried out at machines M_2 and M_3 respectively. The first operation of J_2 is carried out

at machine M_3, the second and third are carried out at machines M_1 and M_2 respectively. All the operations are assumed non-preemptive.

Robots: Robot R_1 shared by M_1 and M_2 can be used to load M_1, to deliver raw material of product J_1 from Entry 1, and unload M_2 to send finished product J_2 fixtured to pallet to AGV 2. Robot R_2 is used to load M_3, to deliver raw material of product J_2 from Entry 2, and unload M_3 to send finished product J_1 fixtured to pallet to AGV 1. Robot R_3 is shared by M_1, M_2 and M_3 to convey intermediate parts. It performs the following functions: unloading M_1, loading M_2, unloading M_2, loading M_3 for job type J_1, and unloading M_3, loading M_1, unloading M_1, loading M_2 for job type J_2.

AGV System: Two AGVs have one pallet position each and are designed for the delivery of final parts and the release of pallets in the system. From M_3, AGV1 sends final product J_1 to Exit 1 and pallet back to Entry 1. From M_2, AGV2 sends final product J_2 to Exit 2 and pallet back to Entry 2. Since they take different paths, collision is avoided and both AGVs can work concurrently.

A. Petri Net Modeling

Based on the modeling method presented before, the Petri net models representing operation sequences for sub-system Job J_1 and J_2 are shown in Figure 4.6. The complete model for the entire automated manufacturing system is represented by merging the same places representing the shared resources in the Petri net models for sub-system Job J_1 and J_2 shown in Figure 4.7. Note that the following shared resource places p_{m3}, p_{r1}, p_{r2} and p_{r3} appear twice respectively in Figure 4.7 to conserve the legibility.

B. Heuristic search based on timed Petri nets

Using Algorithm 4.1 proposed in Section 4.3, we obtain the following optimal input event sequences for cyclic production:

Machine 1: <Operation 1 of Job 1, Operation 2 of Job 2>;

Machine 2: <Operation 2 of Job 1, Operation 3 of Job 2>;

Machine 3: <Operation 1 of Job 2, Operation 3 of Job 1>;

Robot 1: <Acquiring from Entry 1, Loading Machine 1, Unloading Machine 2, Loading AGV 2>;

Robot 2: <Acquiring from Entry 2, Loading Machine 3, Unloading Machine 3, Loading AGV 1>;

Robot 3: <Acquiring from Machine 1, Loading Machine 2, Acquiring from Machine 3, Loading Machine 1, Acquiring from Machine 2, Loading Machine 3, Acquiring from Machine 1, Loading Machine 2>.

The concurrency of these events are explicitly handled in the Petri net formalism. Based on the sequence control structure proposed in Section 4.2, two levels of specification, coordination control level and local control level are used to specify the optimal sequence controller.

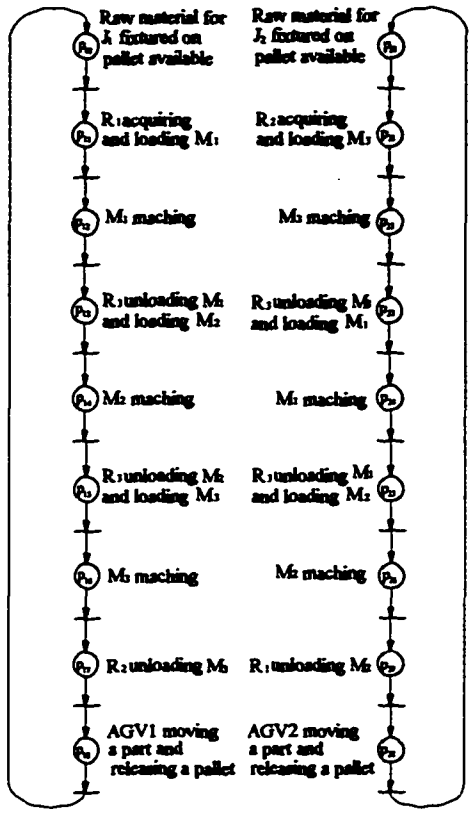


Figure 4.6 The operation sequences for Job 1 (*left*) and Job 2 (*right*)

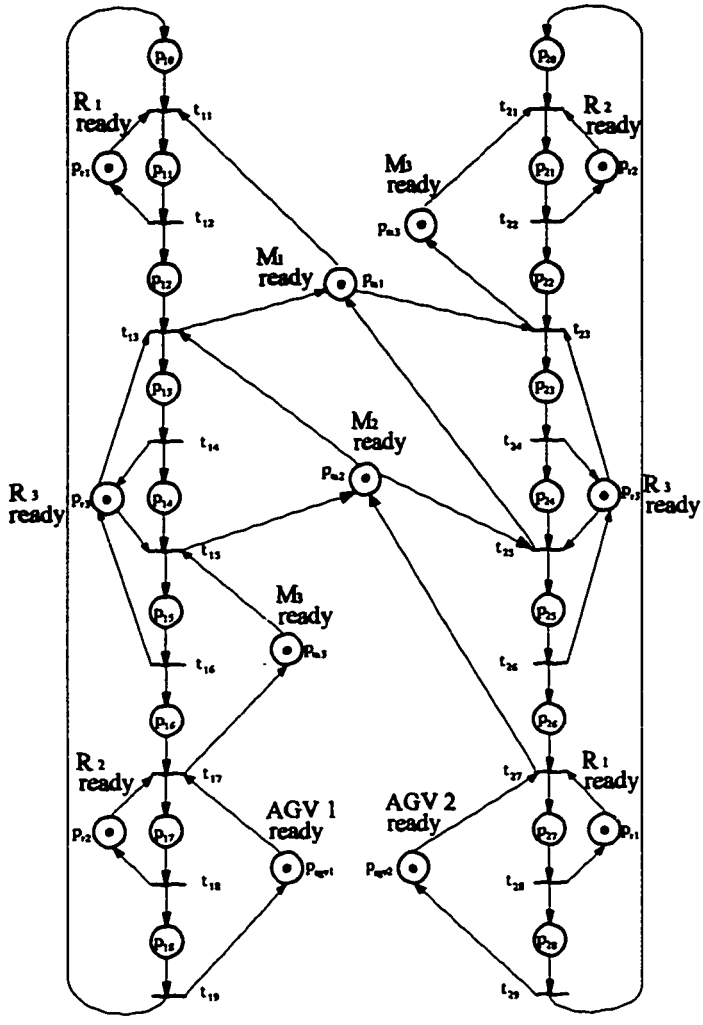


Figure 4.7 The Petri net model for the entire system

C. Synthesis of Marked Graph for Event-Driven Coordination Control

Based on the obtained optimal event sequence, a marked graph is synthesized for the case of both products having one pallet in the system. Figure 4.8 shows the Petri net model for coordination control which consists of synchronization and parallelism of different sub-systems. The presented Petri net is a marked graph in which each place has exactly one input and one output transition. The marked graph model of coordination control is developed as follows:

(i) Model the cyclic manufacturing process for each job type, we obtain the processing circuit $p_{10}t_{11}p_{11}t_{12}p_{12}t_{13}p_{13}t_{14}p_{14}t_{15}p_{15}t_{16}p_{16}t_{17}p_{17}t_{18}p_{10}$ for job type J_1, and the processing circuit $p_{20}t_{21}p_{21}t_{22}p_{22}t_{23}p_{23}t_{24}p_{24}t_{25}p_{25}t_{26}p_{26}t_{27}p_{27}t_{28}p_{20}$ for job type J_2. In each processing circuit, a place represents an event and a transition represents either start or completion of an event.

(ii) Model the sequencing of the part types for each machine according to the obtained optimal input event sequence. Three command circuits are for three machines obtained. The command circuit $c_{11}t_{11}p_{11}t_{12}p_{12}t_{13}c_{12}t_{23}p_{23}t_{24}p_{24}t_{25}c_{11}$ schedules the operations of Machine *1* and corresponding loading and unloading operations performed by robots. Similarly, the command circuit $c_{21}t_{13}p_{13}t_{14}p_{14}t_{15}c_{22}t_{25}p_{25}t_{26}p_{26}t_{27}c_{21}$ for Machine *2* and the command circuit $c_{31}t_{21}p_{21}t_{22}p_{22}t_{23}c_{32}t_{15}p_{15}t_{16}p_{16}t_{17}c_{31}$ for Machine *3* are constructed.

(iii) Associate Boolean conditions with transitions in the net, the logic condition of a transition can be all true logic *1* or the state of some specified steps of SFCs at the local control level.

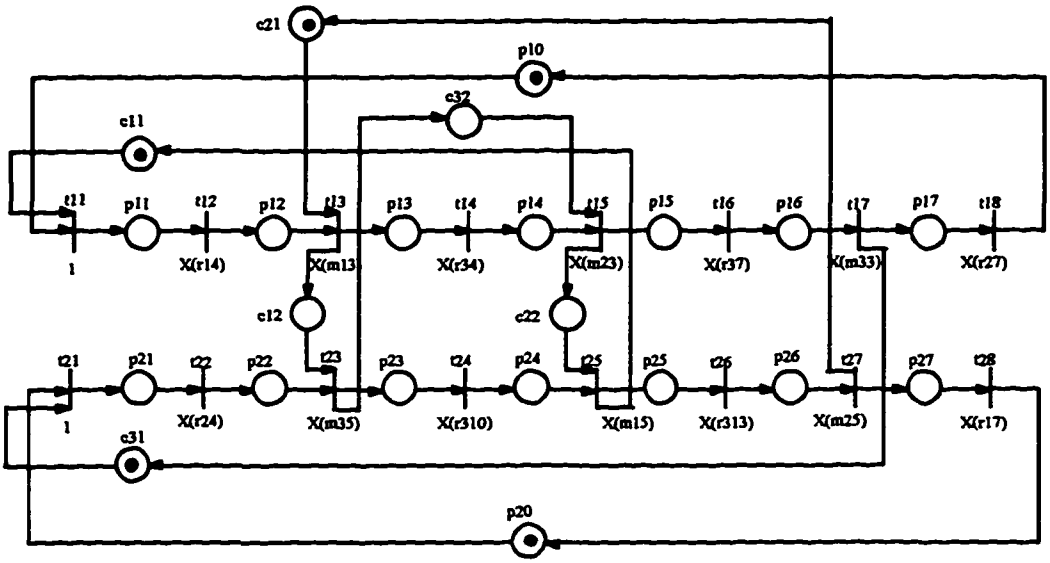
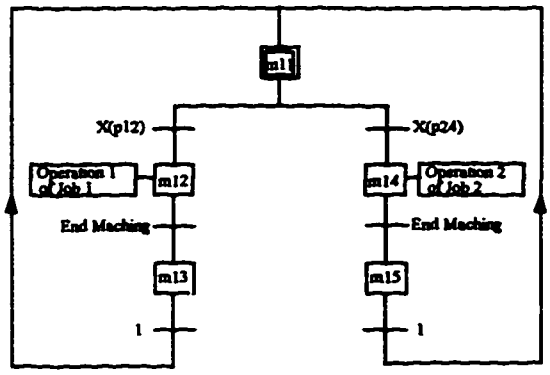


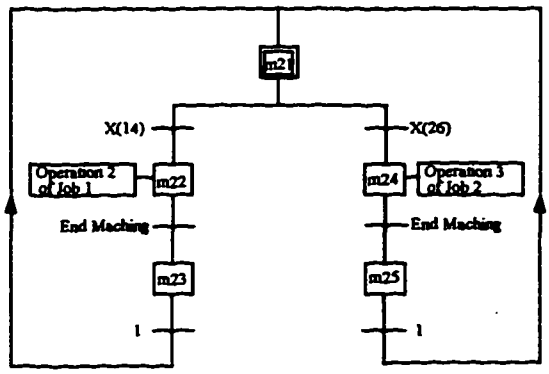
Figure 4.8 Petri net (Marked graph) model for coordination control in Example 4.1

D. Specification of Local Sequence Controllers

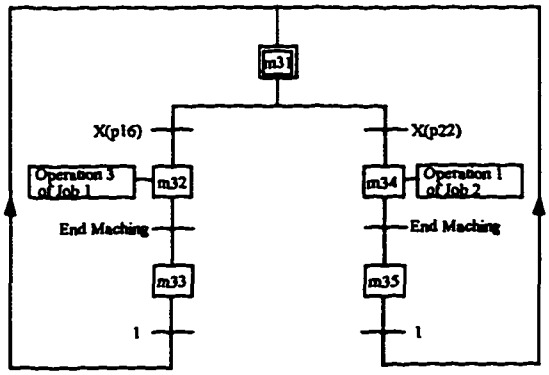
We use sequential function charts to specify local controllers. Figures 4.9(a), (b) and (c) show the sequential function chart models for local control of Machines *1*, *2* and *3*, respectively. Figures 4.10(a), (b) and (c) show the models for local control of Robots *1*, *2* and *3*, respectively. The relation between two levels is realized by the logical conditions associated with some transitions in the coordination model and local control models. The Boolean variable *X(i)* is equal to *1* when and only when place (step) *i* is marked (active). For example, firing of transition t_{11} in Figure 4.8 marks place p_{11} and makes $X(p_{11})$ true. This initiates local controller of Robot *1* in Figure 4.10 (a), which, in turn starts Robot 1 for picking up a part from Entry *1* and then loading Machine *1*. The event of end of loading Machine *1* makes step r_{14} active. This makes the condition related with transition t_{12} in coordination model true. Firing transition t_{12} marks place p_{12} and $X(p_{12})$ becomes true, which in turn makes Machine *1* process operation *1* of Job *1* based on the local controller of Machine *1* in Figure 4.9(a), and so on.



(a)

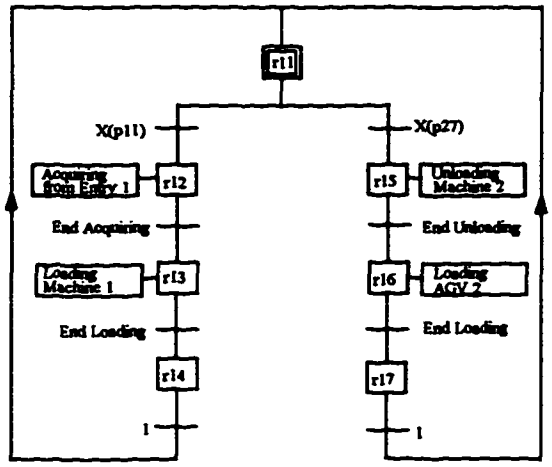


(b)

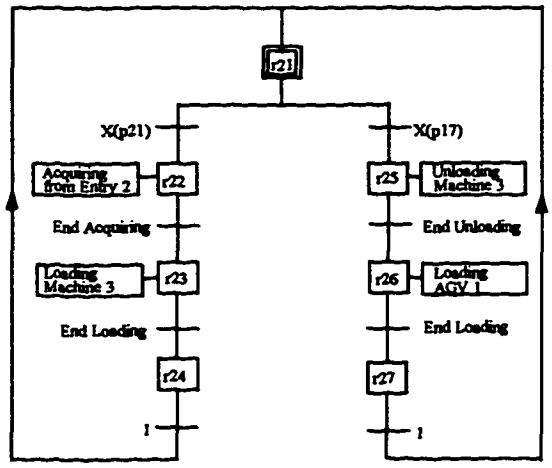


(c)

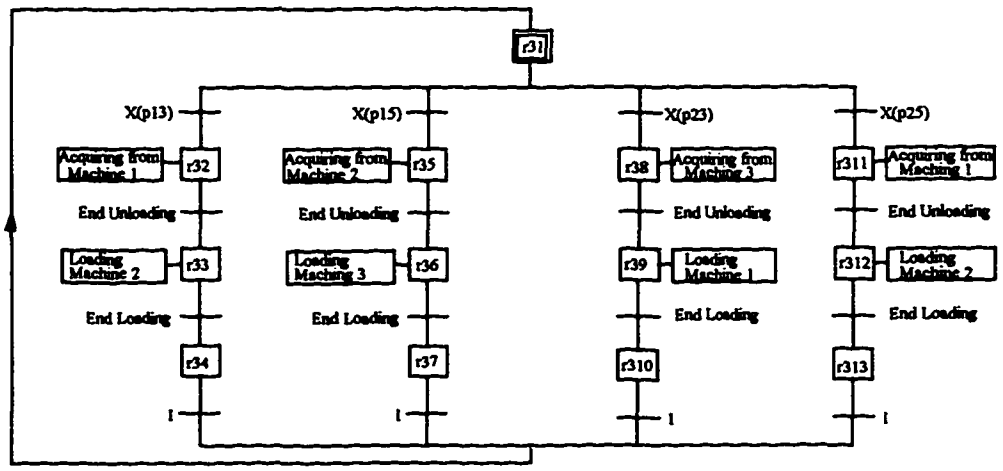
Figure 4.9 The SFC models for local control of Machine 1 (a), Machine 2 (b) and Machine 3 (c) in Example 4.1



(a)



(b)



(c)

Figure 4.10 The SFC models for local control of Robots 1 (a), Robot 2 (b), Robot 3 (c)

4.5 Development of Theoretical Results

We consider a class of FMSs where each job in the system has a fixed production sequence, i.e., no routing flexibility. The work-in-process of each job is limited to 1. We have proposed a synthesis methodology to construct a Petri net (marked graph) as a coordinating discrete-event controller in Section 4.2. The method is demonstrated through an FMS example in Section 4.4. This section presents the main theoretical results to insure the desired qualitative properties of boundedness (safeness), liveness, and reversibility in the resulting Petri net controller. These properties have their significant meanings in manufacturing. Boundedness or safeness guarantees a stable discrete manufacturing process and no capacity overflow. For instance, the boundedness of a place modeling a buffer or queue insures that there will be no overflow, and the safeness of an operation place guarantees that there is no attempt to request execution of an ongoing process (Zhou and DiCesare 1993). Liveness implies a system free from deadlock. Reversibility ensures a cyclic manufacturing system with the ability to initialize from any reachable state and has implications for error recovery in the manufacturing context.

Definition 4.1: Given PN = (P, T, I, O, M_0), a *node* is either a place in P or a transition in T. An *elementary path* is a sequence of nodes: $x_1x_2...x_n$, $n \geq 1$, such that there is an arc from x_i to x_{i+1}, where $1 \leq i < n$. if $n > 1$, $x_i = x_j$ implies that $i = j$, $1 \leq i, j < n$. An *elementary circuit* is a sequence of nodes: $x_1x_2...x_n$, $n > 1$, such that $x_i = x_j$, where $1 \leq i < j < n$, implies that $i = 1$ and $j = n$.

Definition 4.2: An *operation place path* is an elementary path consisting of one place and two transitions. The place in an operation path is called *operation place*. The

operation place in an operation path has an input transition called *starting transition* representing the start of an operation and an output transition called *ending transition* representing the end of an operation.

Definition 4.3: A *processing circuit* is an elementary circuit which models the cyclic production of a job according to its precedence relations. The place representing the availability of a job is called *job resource place*. The token count in a processing circuit is equal to 1 with an initial token deposited in the job resource place.

Definition 4.4: A *command circuit* is an elementary circuit which models the control flow of a shared resource according to the derived sequencing of the jobs on that resource. The token count in a command circuit is equal to 1 with an initial token deposited in the place preceding the first operation place.

Given an FMS with m types of resources and n types of jobs, there exist m command circuits, denoted by C^1, C^2, ..., C^m and n processing circuits, denoted by P^1, P^2, ..., P^m.

Definition 4.5 [Murata 1989]: A marked graph is an ordinary Petri net (P, T, I, O) such that $\forall p \in P$, $t \in T$, $I(p,t) \leq 1$, $O(p,t) \leq 1$, and given any $p \in P$, $|\{ t \in T : O(p,t) = 1\}| = 1$, and $|\{ t \in T : I(p,t) = 1\}| = 1$.

Marked graphs are a subclass of Petri nets characterized by the fact that any place has exactly one input and one output transition. A marked graph with initial marking M_0 is represented by (MG, M_0).

The following four properties about marked graphs are known [Murata 1989].

Property 4.1: For a marked graph, the token count in an elementary circuit is invariant under any firing.

By Property 4.1, If there are no tokens on an elementary circuit at the initial marking, then this elementary circuit remains token-free. Thus, the transitions on this elementary circuit will never be enabled.

Property 4.2: A marked graph (MG, M_0) is live *iff* M_0 puts at least one token on each elementary circuit in MG.

Property 4.3: A live marked graph is reversible.

Property 4.4: The maximum number of tokens that a place can have in a marked graph (MG, M_0) is equal to the minimum number of tokens placed by M_0 on an elementary circuit containing this place.

Theorem 4.1: Given m command circuits $C^1, C^2, ..., C^m$ and n processing circuits $P^1, P^2, ..., P^m$, suppose that a Petri net Z is obtained by merging these subnets along all common operation place paths, then Z is a marked graph.

Proof: In all command and processing circuits, any place has exactly one input and one output transition, any transition has exactly one input and one output place. By merging these subnets along all common operation place paths, each starting common transition has exactly two input and one output places, each ending common transition has exactly one input and two output places. But for each place, it still has exactly one input and one output transition. Therefore, the resulting Petri net Z is a marked graph.

Theorem 4.2: Given m command circuits $C^1, C^2, ..., C^m$ and n processing circuits $P^1, P^2, ..., P^m$. Suppose that a Petri net Z is obtained by merging these subnets along all common operation place paths, then Z is safe, live and reversible.

Proof: From Theorem 4.1, the net Z is a marked graph.

1) *Safeness*: For any place p in Z, it should be contained in a processing circuit or a command circuit. According Property 4.1 and Definition 4.3-4, the token count in a processing circuit or a command circuit is invariant for any marking reachable from the initial marking. Therefore according to Property 4.4, the maximum number of tokens that place p can have is 1 for any marking reachable from the initial marking. This proves the safeness of Z.

2) *Liveness*: According to Property 4.2, to prove the liveness, we just need to show that there exists at least one token on each elementary circuit in Z.

Suppose that the net consisting of processing circuits P^1, P^2, ..., P^m only is denoted by Z^0. Then the command circuits C^1, C^2, ..., C^m are merged to Z^0 one by one. When command circuit C^k, $1 \le k \le m$ is merged to Z^{k-1}, the resulting net is denoted by Z^k.

First, when $k = 0$, the token count in each elementary circuit in Z^0 is one, the conclusion is true. When $k = 1$, the elementary circuits in Z^1 consist of P^1, P^2, ..., P^m and C^1, no other mixed circuits exist. The conclusion is true.

Second, suppose that, for $n = k$, the conclusion is true, i.e., each circuit in Z^k contains at least one token. The following shows that the conclusion is true for $n = k+1$.

The newly added circuits which do not exist in Z^k must be those circuits which contain some places in C^k. If it contains the marked place in C^k, that circuit has at least one token. If it contains no marked place in C^k, starting with the place $p \in C^k$, along the circuit, assuming it has to come to a transition which is shared between C^k and Z^k. Starting from that transition, it has to proceed to one of the marked places of Z^k. This proves that any circuit in Z^{k+1} contains at least one token.

Therefore, Z is live.

3) *Reversibility*: We have proved that Z is live, then Z is also reversible according to Property 4.3.

4.6 Performance Evaluation

For timed marked graphs, there exists already the formula to find the system cycle time (Ramamoorthy and Ho 1980, Hillion and Proth 1989). For a marked graph which has time delays in its transition or place, the system cycle time C is given by

$C =$ Max $\{ T_i / N_i : i = 1,2,\ldots,n \}$ where

$T_i =$ Sum of the transition and place delays in circuit γ_i,

$N_i =$ Total number of tokens in the places in circuit γ_i, and

$n =$ Number of circuits in the marked graph.

There are three types of circuits in a marked graph which models the manufacturing system. Processing circuits model the manufacturing process of the sequence of each job. Command circuits model the sequencing of the jobs on the machines. If a circuit includes nodes of both processing and command circuit, then such a circuit is called a mixed circuit (Hillion and Proth 1989). Knowing the circuits and the time delays in transitions and/or places, we can evaluate the system performance by the above formula. A linear programming formula can also be used for performance evaluation (Morioka and Yamada 1991).

We use an example to demonstrate the evaluation of the resulting controller's performance and sensitivity to randomness. To make a comparison with the control system driven by heuristic dispatching rules, the following traditional job-shop system is used as an example. This is because no commonly used dispatching rules can generate effective and deadlock-free scheduling decisions for the systems with multi-layer resource-sharing, such as one in Example 4.1.

Example 4.2: Let us consider an FMS with three machines, M_1, M_2 and M_3. There are four jobs, J_1, J_2, J_3 and J_4 which have three processes each. Table 4.2 shows the job requirements.

Table 4.2 Job Requirements of Example 4.2

Operations/Jobs	J_1	J_2	J_3	J_4
1	(M_1,4)	(M_2,1)	(M_3,3)	(M_2,3)
2	(M_2,3)	(M_1,4)	(M_2,2)	(M_3,3)
3	(M_3,2)	(M_3,4)	(M_1,3)	(M_1,1)

Figure 4.11 shows the Petri net model for the sub-system Job 1. Similarly we can get Petri net models for Job 2, Job 3, and Job 4. The complete Petri net model for the system is obtained by merging these sub-models.

Using Algorithm 4.1, we obtain the following optimal input event sequences for each machine:

Machine 1: <Operation 2 of Job 2, Operation 1 of Job 1, Operation 3 of Job 4, Operation 3 of Job 3>;

Machine 2: <Operation 1 of Job 2, Operation 1 of Job 4, Operation 2 of Job 3, Operation 2 of Job 1>;

Machine 3: <Operation 1 of Job 3, Operation 2 of Job 4, Operation 3 of Job 2, Operation 3 of Job 1>.

These sequences consists of three command circuits in the discrete event controller represented by a marked graph. A token in a command circuit represents the availability of the machine to process a specific job. Since a machine is assumed to process only one job at a time, there can be only one token in each command circuit. The sequences consisting of processing circuits are determined by the technological precedence of job requirements as follows:

Job 1: <Waiting in the buffer, Processing in Machine 1, Waiting in the buffer, Processing in Machine 2, Waiting in the buffer, Processing in Machine 3, Finishing the job>;

Job 2: <Waiting in the buffer, Processing in Machine 2, Waiting in the buffer, Processing in Machine 1, Waiting in the buffer, Processing in Machine 3, Finishing the job>;

Job 3: <Waiting in the buffer, Processing in Machine 3, Waiting in the buffer, Processing in Machine 2, Waiting in the buffer, Processing in Machine 1, Finishing the job>;

Job 4: <Waiting in the buffer, Processing in Machine 2, Waiting in the buffer, Processing in Machine 3, Waiting in the buffer, Processing in Machine 1, Finishing the job>.

The jobs in-process are represented by the tokens circulating in the processing circuits. The marked graph for the discrete event control of FMS in Example 4.2 is shown in Figure 4.12.

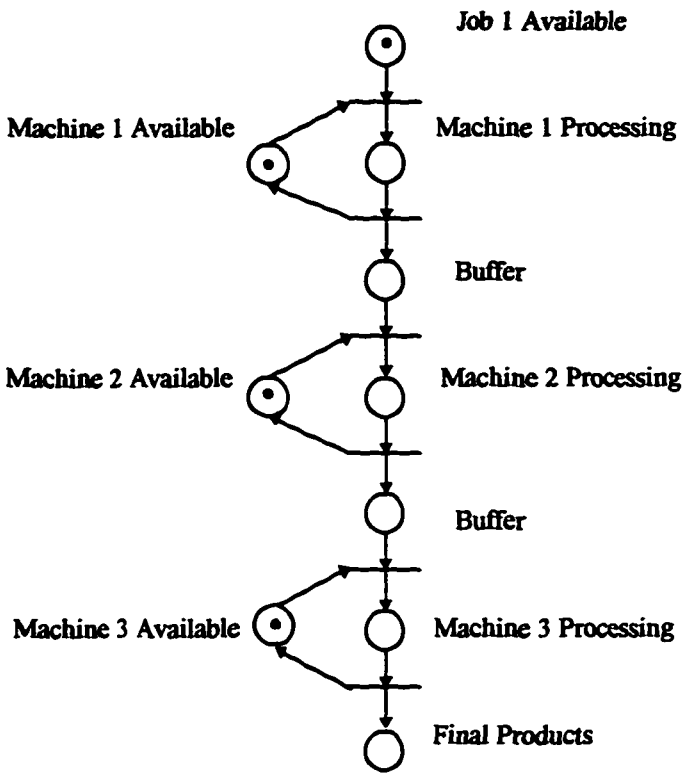


Figure 4.11 The petri net model of the sub-system Job 1 in Example 4.2

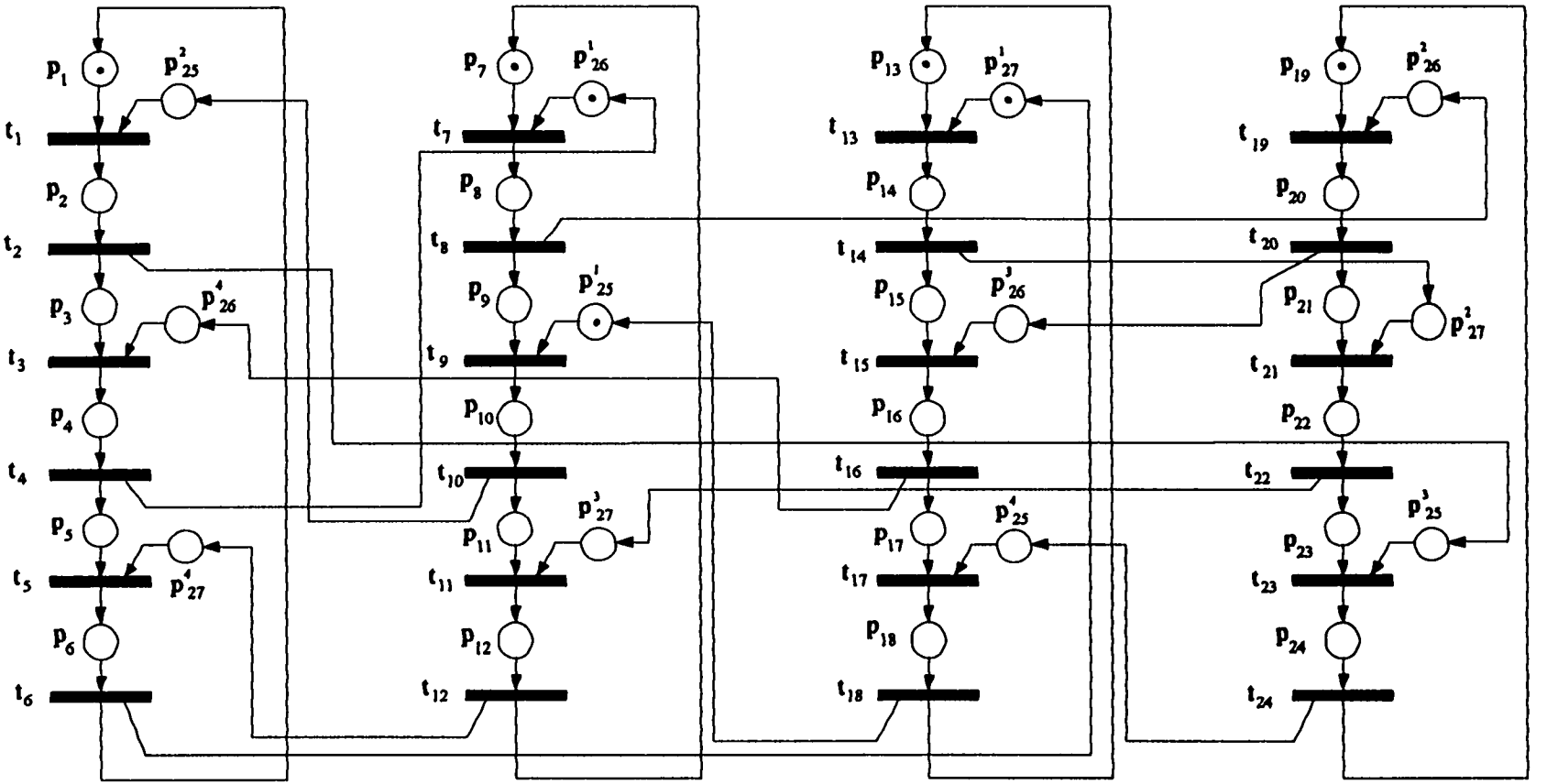


Figure 4.12 The marked graph Controller of Example 4.2

Based on the formula presented above, the cycle time of the marked graph shown in Figure 4.12 is 12. The system throughput (production rate) is 4/12 (0.333).

To make a comparison, two benchmark dispatching rules are employed for scheduling and control. One is SPT (Shortest Processing Time), which sequences jobs by the imminent processing time and gives the priority to the job with the minimum processing time in the input queue of an available machine. SPT is a widely used rule that has been found to perform reasonably well on a number of performance measures in a variety of manufacturing environments (Blackstone, *et al.* 1982, Askin and Standridge 1993). Another one is LWKR (Least Work Remaining), which sequences jobs by the total processing time of unfinished operations and gives the priority to the job with the smallest total processing time in the input queue of an available machine. Varying the lot size for each job from 10 to 100, we obtain the average production rate 0.313 for SPT and 0.311 for LWKR. The comparison result for production rates obtained from different methods is shown in Table 4.3.

Table 4.3 The performance comparison of Example 4.2

Production Rate		
LWKR	SPT	Marked Graph
0.311	0.313	0.333

4.7 Sensitivity to Randomness

The obtained marked graph is an inherently deadlock free discrete event controller. For real-time control of FMS, this greatly reduces the operational control burdens comparing with other Petri net based deadlock avoidance controllers (Banaszak and Krogh 1990, Narahari and Viswanadham 1990, Hsieh and Chang 1994). Although the marked graph based controller provides valuable advantages in both aspects of real-time implementation and throughput optimization, its performance greatly depends on the deterministic conditions of functioning. It is clear that for practical implementation of the controller, some randomness can happen such as processing time variations and machine breakdowns. Because the marked graph controller is based on the event-driven philosophy instead of the time-driven which specifies a list of times at which certain activities are to occur, so it is tolerable of disturbances. But the system performance such as throughput will degrade when disturbances exist.

With processing time variations, the delays associated with places or transitions are stochastic, the notion of cycle time disappears. Various upper and lower bounds of the average cycle time of a general stochastic marked graph are derived (Campos, Chiola and Silva 1991, Baccelli and Liu 1992, Xie 1994).

For the marked graph controller obtained from our proposed design method, we have the following performance evaluation results based on Xie's work (1994).

Given mean values and standard deviations of the processing times, the upper and lower bounds of average cycle time are as follows:

$$\pi^{D}(M_0) \leq \pi(M_0) \leq \pi^{D}(M_0) + \Sigma_{i \in I} \sigma_i,$$

where $\pi(M_0)$ represents the average cycle time of the marked graph for the given initial marking M_0 considering the randomness of processing times. The lower bound $\pi^D(M_0)$ is equal to the exact cycle time in the deterministic case and computed by using the mean values of processing times as deterministic processing times. The upper bound consists of two terms, the first term is the cycle time of deterministic case $\pi^D(M_0)$, and the second term is the addition of standard deviations of processing times for all operations belong to the operation set I. This upper bound converges to the exact average cycle time as the standard deviations tend to zero. This shows a fact that the marked graph with less uncertainty has smaller average cycle time. Thus for a fixed lot sizes, the makespan will increase when the uncertainty of processing times increases. We use a simulation experiment to illustrate this fact.

Example 4.3: For the system presented in Example 4.2, we consider the variations of processing times. The mean values of processing times are given as the deterministic processing times in Example 4.2. The deviations from these mean values are generated as:

$$\textit{Percentage of variations} * \textit{mean value} * \textit{random number},$$

where the random number is generated from a random variable with uniform distribution defined on [-1, 1].

Varying the percentage of variation, we simulate the system for 2000 times in SUN Sparc station. We make a comparison between the marked graph controller and the one driven by the dispatching rule SPT. It should be noted that the employed marked graph is the one we derived in Example 4.2 for the deterministic case, i.e. the sequencing of the jobs on each machine is fixed even if there are variations of processing times. While when

the dispatching rule SPT is employed, the sequencing of the jobs on each machine is changed due to variations of processing times. This is because the marked graph controller generally is synthesized off-line and implemented on-line, while dispatching rules are often used as on-line rules. For the fixed lot size (30, 30, 30, 30), the makespan versus the percentage of variations obtained from two methods is shown in Figure 4.13. Because of uncertainty, the system performance will degrade for both cases. But the on-line dispatching rule SPT is less sensitive to the variations of processing times than the marked graph. This is because SPT rule adapts the sequencing of the jobs on each machine to the variations of processing times, while the marked graph fixes the sequencing of the jobs on each machine, which is derived in the deterministic case. But within about 34% of variations, the marked graph still performs better than SPT rule for the testing lot size (30, 30, 30, 30).

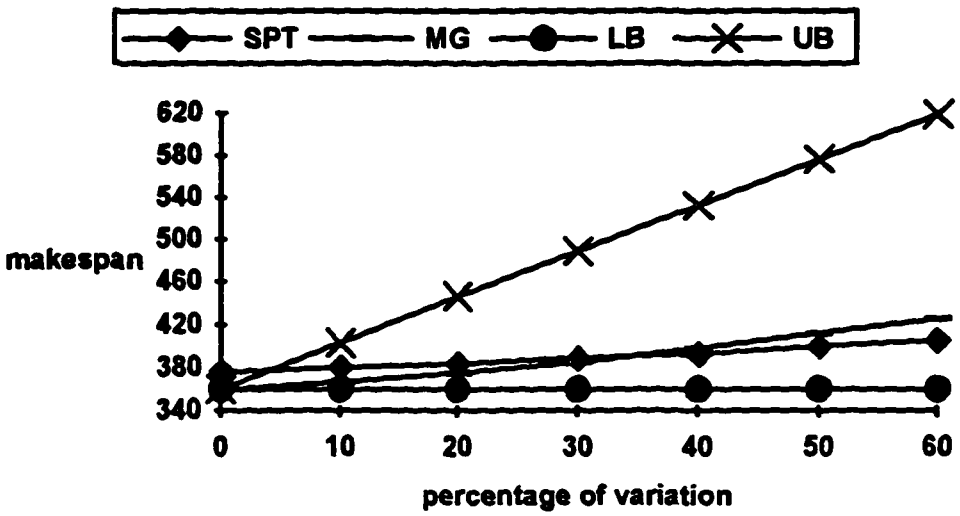


Figure 4.13 Senstivity to processing time variations in Example 4.3

4.8 Summary

This chapter starts with a bottom-up approach and search for the best performance sequence of events and then synthesize the desirable Petri net controllers. The method insures the desired qualitative properties of liveness, boundedness (safeness), and reversibility in the resulting system, which imply freedom from deadlock, no capacity overflow, and cyclic behavior, respectively. This precludes the costly mathematical analysis for these properties and reduces on-line computation overhead to avoid deadlocks. The performances and sensitivities of resulting Petri nets, thus corresponding control systems, are evaluated. Even though there are several studies in this aspect (Krogh and Beck 1986, Koh and DiCesare 1991, Zhou, DiCesare and Desrochers 1992), for the system with multi-layer resource-sharing and different products sets manufactured concurrently, modeling of a Petri net controller with desirable properties becomes extremely difficult based on their methods. Their methods focus on the logical behavior only.

Future research will include investigation of stochastic Petri nets to describe stochastic behavior, such as failures of machine tools, repair time, variations of processing time. The presented work is based on deterministic timed Petri nets and does not handle the stochastic situations. The work on the evaluation of the sensitivity in this chapter is a good start to this problem.

CHAPTER 5

A HYBRID HEURISTIC SEARCH ALGORITHM FOR SCHEDULING FMS

5.1 Introduction

Scheduling problems arise when multiple kinds of part types are machined respectively by multiple kinds of shared resources according to their technological precedence constraints. We need to determine the optimal input sequence of jobs and resource usage for a given job mix. Note that the required ordering of operations within each job must be preserved. Production scheduling problems are very complex and have been proved to be NP-hard problems (France 1982).

A new application area for production scheduling theory comes from flexible manufacturing systems. An FMS can be defined as an integrated manufacturing system consisting of automated material handling devices and numerically controlled machines that can simultaneously process medium-sized volumes of a variety of part types. Comparing with the classical job shop scheduling problem, the FMS scheduling problem has the following new features (Leon, *et al.* 1994):

- General resource models: machines, buffer space and material handling equipment must be included in a unified model.
- Part contact states: the loading, unloading and movement of parts through the manufacturing system must be scheduled.
- Deadlock states: deadlock arises from the explicit recognition of material handling and buffer space resources. A deadlock-free schedule should be obtained.

- Dynamic machine routing: machine routings specify the machines that are required for each operation of a given job. Routing flexibility in FMS makes machine routing a dynamic decision process.

To cope with the complexity and flexibility of FMS, many researchers have proposed various methods for its scheduling. Because of its NP-hard characteristics, it is very difficult or impossible to find the optimal solution for a sizable FMS scheduling problem. An efficient heuristic method is necessary to systematically work out a sub-optimal solution. Current scheduling approaches such as mathematical programming models (Luh and Hoitomt 1993, Sawik 1990) can seek effective solutions to well-formulated optimization problems. They, however, have formulation difficulties in handling shared resources, deadlock constraints and routing flexibility. Approaches such as queuing theory (Berman and Maimon 1986, Jafari 1987) and simulation (Kim 1994, Wu and Wysk 1989) cannot obtain an exact solution or the solution may be far from optimal.

Petri net theory has been applied for modeling, performance analysis and discrete event control of flexible manufacturing systems. There are also some works on scheduling. Shen *et al.* (1992) present a branch and bound search scheme based on Petri nets. The presented algorithm need a great amount of computer memory, since the size of the reachability graph of a Petri net increases very fast with its size. Zhou, Chiu and Xiong (1995) also employed a Petri net based branch and bound method to schedule flexible manufacturing systems. In their method, instead of randomly selecting one decision candidate from candidate sets (enabled transition sets in Perti net based models), they select the one based on heuristic dispatching rules such as SPT. Lee and DiCesare (1994)

present a Petri net-based heuristic scheduling method for flexible manufacturing, although the heuristic functions given in that paper do not guarantee the admissibility, the condition for an optimal solution (Pearl 1984). Deadlocks arising from limited buffer space resources are not investigated in these previous works.

Petri nets can concisely model the dynamics of flexible manufacturing, multiple kinds of resources (machines, robots, AGVs and buffer space) and constraints of systems in a single unified model. The deadlock states are explicitly defined in the Petri net framework, so no more equations are employed to describe deadlock avoidance constraints. The goal of this chapter is to present a hybrid heuristic search algorithm based on Petri nets for scheduling FMSs. The objectives of this chapter are:

1. To introduce a backtracking (BT) search and make a comparison with the best-first (BF) search through an example.
2. To propose a hybrid search scheme which combines the heuristic best-first search and controlled backtracking search in a Petri net framework.
3. To present a comparison between two different hybrid strategies: BF-BT combination and BT-BF combination.
4. To present an FMS scheduling case with routing flexibility.
5. To present a scheduling example for a semiconductor test facility.

5.2 Best First Search and Backtracking Search

An event-driven schedule is searched in a timed Petri nets (TPN) framework to achieve minimum or near minimum makespan. This chapter employs deterministic timed Petri nets

by associating time delays with places. The transitions can be fired with a zero duration which is consistent with the definition of non-timed Petri nets. In the Petri net model of a system, firing of an enabled transition changes the token distribution (marking). A sequence of firings results in a sequence of markings, and all possible behaviors of the system can be completely tracked by the reachability graph of the net. The search space for the optimal event sequence is the reachability graph of the net, and the problem is to find a firing sequence of the transitions in the Petri net model from the initial marking to the final one.

We have presented an admissible heuristic algorithm based on best-first (BF) strategy in Chapter 4. For completeness, we present it here again.

Algorithm 5.1 (Best-First):

1. Put the start node (initial marking) m_0 on OPEN.
2. If OPEN is empty, exit with failure.
3. Remove from OPEN and place on CLOSED a marking m for which f is minimum.
4. If marking m is a goal node (final marking), exit successfully with the solution obtained by tracing back the pointers from marking m to marking m_0.
5. Otherwise find the enabled transitions of the marking m, generate the successor markings for each enabled transition, and attach to them pointers back to m.
6. For every successor marking m' of marking m:

(a) Calculate $f(m')$.

(b) If m' was neither on OPEN nor on CLOSED, add it to OPEN. Assign the newly computed $f(m')$ to marking m'.

(c) If m' already resided on OPEN or CLOSED, compare the newly computed $f(m')$ with the value previously assigned to m'. If the old value is lower, discard the newly generated marking. If the new value is lower, substitute it for the old and direct its pointer along the current path. If the matching marking m' resided on CLOSED, move it back to OPEN.

7. Go to step 2.

At each step of the best-first search process, we select the most promising of the markings we have generated so far. This is done by applying an appropriate heuristic function to each of them. We then expand the chosen marking by firing all enabled transitions under this marking. If one of successor markings is a final marking, we can quit. If not, all those new markings are added to the set of markings generated so far. Again the most promising marking is selected and the process continues.

Once the Petri net model of the system is constructed, given initial and final markings, an optimal schedule can be obtained using the above algorithm. But for a sizable FMS scheduling problem, it is very difficult or impossible to find the optimal solution in a reasonable amount of time and memory space. This chapter develops a search algorithm by combining the heuristic best-first strategy with the controlled backtracking strategy based on the execution of the Petri nets. The backtracking method applies the last-in-first-out policy to node generation instead of node expansion. When a marking is first selected

for exploration, only one of its enabled transitions is chosen to fire, and thus only one of its successor markings is generated. This newly generated marking is again submitted for exploration. When the generated marking meets some stopping criterion, the search process backtracks to the closest unexpanded marking which still has unfired enabled transitions.

Algorithm 5.2 (Backtracking):

1. Put the start node (initial marking) m_0 on OPEN.
2. If OPEN is empty, exit with failure.
3. Examine the topmost marking from OPEN and call it m.
4. If the depth of m is equal to the depth-bound or if all enabled transitions under marking m have already been fired, remove m from OPEN and go to step 2; otherwise continue.
5. Generate a new marking m' by firing an enabled transition not previously fired under marking m. Put m' on top of OPEN and provide a pointer back to m.
6. Mark m to indicate that the above transition has been fired.
7. If marking m' is a goal node (final marking), exit successfully with the solution obtained by tracing back the pointers from marking m' to marking m_0.
8. If m' is a deadlock marking, remove it from OPEN.
9. Go to step 2.

Both Algorithm 5.1 (best-first) and Algorithm 5.2 (backtracking) allow recovery from disappointing search avenues to reaccess previously suspended alternative markings. If no enabled transition is found for a marking, it means this marking represents a deadlock. The search process will explore another marking on the list OPEN. If all the markings in OPEN are exhausted, it means there is no path connecting the given initial and final markings. The best-first search strategy examines, before each decision, the entire set of available alternative markings, those newly generated as well as all those suspended in the past. The backtracking search strategy is committed to maintaining in storage only a single path containing the set of alternative markings leading to the current marking. It proceeds forward heedlessly to find a feasible schedule without considering the optimality. Since only the markings on the current firing sequence are stored, it requires less memory.

Example 5.1: We use a scheduling example to compare the computation complexity and optimality of Algorithm 5.1 and Algorithm 5.2. The problem is to schedule an FMS with three machines, M_1, M_2 and M_3. There are four jobs, J_1, J_2, J_3 and J_4 which have three processes each. Table 5.1 shows the job requirements.

Table 5.1 Job Requirements for Example 5.1

Operations/Jobs	J_1	J_2	J_3	J_4
1	(M_1,2)	(M_3,4)	(M_1,3)	(M_2,3)
2	(M_2,3)	(M_1,2)	(M_3,5)	(M_3,4)
3	(M_3,4)	(M_2,2)	(M_2,3)	(M_1,3)

Figure 5.1 shows the Petri net model for the sub-system Job 1. Similarly we can get Petri net models for Job 2, Job 3, and Job 4. The complete Petri net model for the system is obtained by merging these sub-models. Several different job sizes of this example are tested and makespans, numbers of generated markings and CPU times are shown in Table 5.2.

Table 5.2 Scheduling results for Example 5.1

lot sizes				makespan		number of markings		CPU time (sec) (Sun SPARC 20)	
J_1	J_2	J_3	J_4	BF	BT	BF	BT	BF	BT
1	1	1	1	17	21	155	25	0.16	0.06
2	2	1	1	25	33	501	37	0.56	0.1
5	5	2	2	58	105	3437	85	14	0.16
8	8	4	4	100	198	9438	145	112	0.23
10	10	6	6	134	274	23092	193	720	0.38

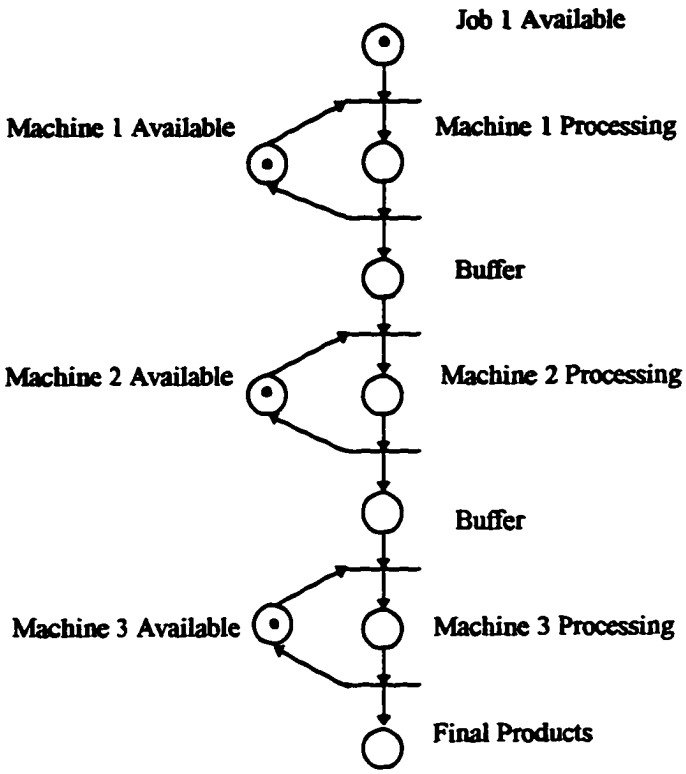


Figure 5.1 The petri net model of the sub-system Job 1 in Example 5.1

From the Table 5.2, we see that Algorithm 5.1 find the optimal solutions at the expense of computation complexity, while Algorithm 5.2 reduce the computation complexity at the expense of optimality. For many practical FMS scheduling problems, it is desired to get a good solution (even not optimal) in a reasonable amount of time and storage. This suggests that a combination of best-first search and backtracking search should be implemented.

5.3 Hybrid Heuristic Search Algorithms

The need to combine BF and BT strategies is a result of computational considerations. For a sizable FMS scheduling problem, if we cannot afford the memory space and computation time required by a pure BF strategy, we can employ a BF-BT combination that cuts down the storage requirement and computation time at the expense of narrowing the evaluation scope.

In the following Algorithm 5.3, the heuristic best-first search strategy is applied at the top of reachability graph of the timed Petri net model and a backtracking search strategy at the bottom. We begin with BF search until a depth-bound dep_0 is reached. Then BT search is employed using the best present marking as a starting node. If it fails to find a solution, we return to get the second best marking on OPEN as a new root for a BT search, and so on.

Algorithm 5.3 (Hybrid BF-BT):

1. Put the start node (initial marking) m_0 on OPEN.

2. If OPEN is empty, exit with failure.
3. Remove from OPEN and place on CLOSED a marking m for which f is minimum.
4. If marking m is a goal node (final marking), exit successfully with the solution obtained by tracing back the pointers from marking m to marking m_0 .
5. If the depth of marking m is greater than the depth-bound dep_0, go to Step 9; otherwise continue.
6. Find the enabled transitions of the marking m, generate the successor markings for each enabled transition, and attach to them pointers back to m.
7. For every successor marking m' of marking m:

(a) Calculate $f(m')$.

(b) If m' was neither on OPEN nor on CLOSED, add it to OPEN. Assign the newly computed $f(m')$ to marking m'.

(c) If m' already resided on OPEN or CLOSED, compare the newly computed $f(m')$ with the value previously assigned to m'. If the old value is lower, discard the newly generated marking. If the new value is lower, substitute it for the old and direct its pointer along the current path. If the matching marking m' resided on CLOSED, move it back to OPEN.

8. Go to Step 2.
9. Take the marking m as the root node for BT search, put it on OPEN0.
10. If OPEN0 is empty, go to Step 2.
11. Examine the topmost marking from OPEN0 and call it m'.

12. If all enabled transitions of marking m' have been selected to fire, remove it from OPEN0 and go to Step 10.
13. Generate a successor marking m'' for one enabled transition not firing before, calculate $g(m'')$, put m'' on top of OPEN0 and provide a pointer back to m'.
14. If marking m'' is a goal node (final marking), exit successfully with the solution obtained by tracing back the pointers from marking m'' to the initial marking m_0.
15. If m'' is a deadlock marking, remove it from OPEN0.
16. Go to Step 10.

An opposite approach is starting a backtracking search on the top of the reachability graph followed by heuristic best-first ending. This strategy is implemented in Algorithm 5.4. We begin BT until a depth-bound dep_0 is reached. Then we employ the heuristic BF search from the current marking until it returns the final marking. If the BF search fails to find a solution, we return to backtracking and again use BF upon reaching the depth-bound dep_0.

Algorithm 5.4 (Hybrid BT-BF)

1. Put the start node (initial marking) m_0 on OPEN0.
2. If OPEN0 is empty, exit with failure.
3. Examine the topmost marking from OPEN0 and call it m.

4. If all enabled transitions under marking m have already been fired, remove m from OPEN0 and go to step 2; otherwise continue.
5. If the depth of marking m is greater than the depth-bound dep_0, go to Step 10; otherwise continue.
6. Generate a new marking m' by firing an enabled transition not previously fired under marking m. Put m' on top of OPEN0 and provide a pointer back to m.
7. Mark m to indicate that the above transition has been fired.
8. If m' is a deadlock marking, remove it from OPEN0.
9. Go to step 2.
10. Take the marking m from BT search as the start node m_0 and put it on OPEN.
11. If OPEN is empty, back to Step 2 and return to backtracking search.
12. Remove from OPEN and place on CLOSED a marking m for which f is minimum.
13. If marking m is a goal node (final marking), exit successfully with the solution obtained by tracing back the pointers from marking m to marking m_0 .
14. Otherwise find the enabled transitions of the marking m, generate the successor markings for each enabled transition, and attach to them pointers back to m.
15. For every successor marking m' of marking m:

(a) Calculate $f(m')$.

(b) If m' was neither on OPEN nor on CLOSED, add it to OPEN. Assign the newly computed $f(m')$ to marking m'.

(c) If m' already resided on OPEN or CLOSED, compare the newly computed $f(m')$ with the value previously assigned to m'. If the old value is lower, discard the

newly generated marking. If the new value is lower, substitute it for the old and direct its pointer along the current path. If the matching marking m' resided on CLOSED, move it back to OPEN.

16. Go to step 2.

In both Algorithms 5.3 and 5.4, the heuristic function $h(m)$ is a lower bound to all complete solutions descending from the current marking. This is a guarantee for an optimal solution if a pure BF strategy is applied. The backtracking strategy is controllable through the depth-bound dep_0, i.e., if one can afford the memory space required by a pure BF strategy, only the pure BF search is employed, and so an optimal schedule is obtained. Otherwise, a hybrid BF-BT or BT-BF combination can be implemented that cuts down the storage requirement at the cost of narrowing the evaluation scope.

In the following example, we make a comparison between Algorithms 5.3 and 5.4. We set the different depth bound to see the relations between the optimality and computation complexity.

Example 5.2: Compare the schedule quality of Algorithm 5.3 and 5.4 based on the FMS schedule problem presented in the example 5.1.

The three sets of lot size (5, 5, 2, 2), (8, 8, 4, 4) and (10, 10, 6, 6) are tested. We employ both Algorithms 5.3 and 5.4. The scheduling results of makespan, number of generated markings and computation time are shown in Table 5.3, 5.4 and 5.5 for the lot size (5, 5, 2, 2), (8, 8, 4, 4) and (10, 10, 6, 6) respectively. The optimal makespans for different cases obtained from pure BF search in Table 5.2 are also shown in these tables.

Table 5.3 Scheduling results of Example 5.2 for lot size (5, 5, 2, 2)

Depth for BF search	makespan		number of markings		CPU time (sec) (Sun SPARC 20)		Optimal makespan
	BF-BT	BT-BF	BF-BT	BT-BF	BF-BT	BT-BF	pure BF
20	94	88	571	248	0.65	0.38	58
40	85	80	1607	484	4	0.8	58
50	79	70	2132	1247	6	3.6	58
60	74	64	2775	1520	8	6.5	58
80	64	62	3308	1687	11	7	58

Table 5.4 Scheduling results of Example 5.2 for lot size (8, 8, 4, 4)

Depth for BF search	makespan		number of markings		CPU time (sec) (Sun SPARC 20)		Optimal makespan
	BF-BT	BT-BF	BF-BT	BT-BF	BF-BT	BT-BF	pure BF
40	168	163	3888	585	24	1.4	100
60	154	140	5234	1590	38	7	100
80	140	121	7699	2873	49	18	100
100	127	112	8819	4545	90	36	100
120	108	104	9233	8045	104	76	100

Table 5.5 Scheduling results of Example 5.2 for lot size (10, 10, 6, 6)

Depth for BF search	makespan		number of markings		CPU time (sec) (Sun SPARC 20)		Optimal makespan
	BF-BT	BT-BF	BF-BT	BT-BF	BF-BT	BT-BF	pure BF
80	206	209	6281	1254	64	5	134
100	198	181	12341	2315	240	16	134
120	180	162	16602	8495	480	139	134
140	169	150	20155	11368	540	390	134
160	153	148	21797	18875	660	560	134

Both Algorithm 5.3 (BF-BT) and 5.4 (BT-BF) cut down the computation complexity by narrowing the evaluation scope at the expense of losing the optimality. The relations of computation complexity (number of generated markings and computation time) reduced versus optimality lost are shown in Figure 5.2, 5.3 and 5.4 for three different sets of lot size (5, 5, 2, 2), (8, 8, 4, 4) and (10, 10, 6, 6) respectively. In these figures, the percentage of optimality lost, which is the comparison of the makespan, is equal to

$$\frac{Hybrid - BF}{BF} * 100\%$$

and the percentage of computation complexity reduced, which is the comparison of the storage (number of generated markings) or computation time, is equal to

$$\frac{BF - Hybrid}{BF} * 100\%$$

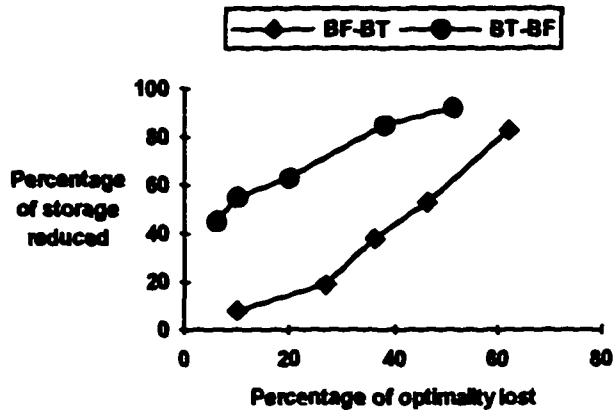


Figure 5.2(a) Percentage of storage reduced versus percentage of optimality lost for lot size (5, 5, 2, 2) in Example 5.2

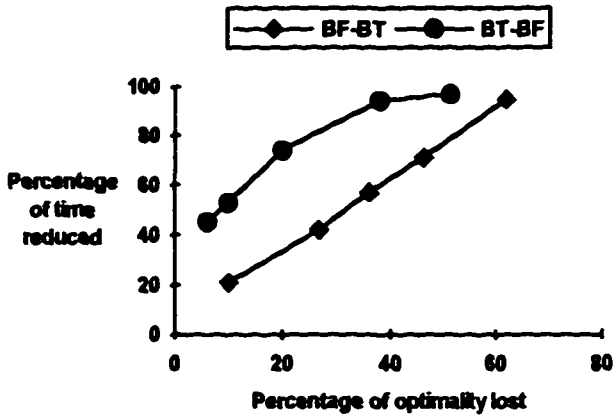


Figure 5.2(b) Percentage of computation time reduced versus percentage of optimality lost for lot size (5, 5, 2, 2) in Example 5.2

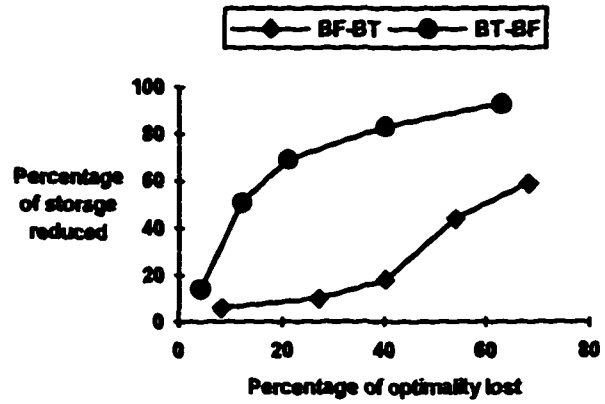


Figure 5.3(a) Percentage of storage reduced versus percentage of optimality lost for lot size (8, 8, 4, 4) in Example 5.2

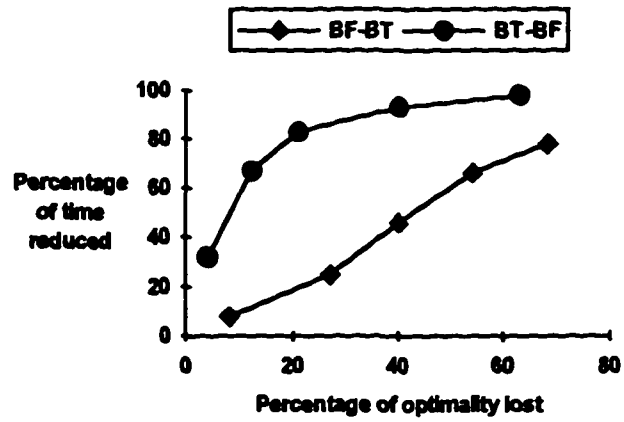


Figure 5.3(b) Percentage of computation time reduced versus percentage of optimality lost for lot size (8, 8, 4, 4) in Example 5.2

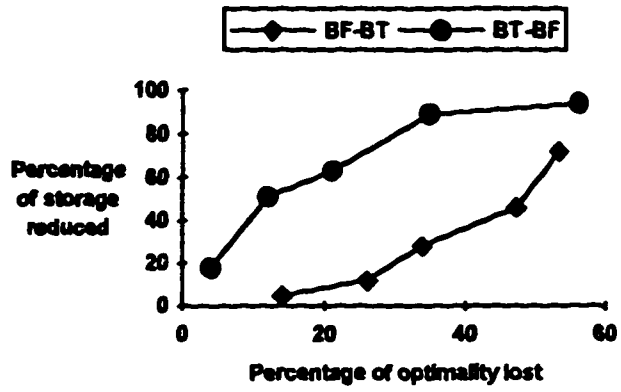


Figure 5.4(a) Percentage of storage reduced versus percentage of optimality lost for lot size (10, 10, 6, 6) in Example 5.2

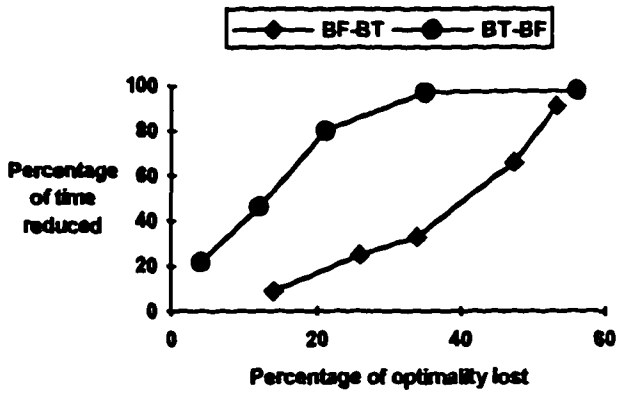


Figure 5.4(b) Percentage of computation time reduced versus percentage of optimality lost for lot size (10, 10, 6, 6) in Example 5.2

From the testing results the following conclusions are drawn. The hybrid heuristic search which employs the heuristic best-first search at the bottom of the Petri net reachability graph (Algorithm 5.4) performs much better than the one which employs the heuristic best-first search at the top of the Petri net reachability graph (Algorithm 5.3). This is due to two reasons. One is that the performance of heuristic best-first search is at its best when its guiding heuristic is more informed, and this usually happens at the bottom of the search graph (Pear 1984). Thus BT-BF search greatly reduces the computation complexity comparing with BF-BT search which employs the heuristic best-first search at the top of the search graph. Another reason is that there are fewer firing transitions for the markings at the bottom of Petri net reachability graph than those at the top. This is because at the late stages of a scheduling task, the reduced number of remaining operations reduces the number of choices. Hence, the number of alternatives considered in each decision for BT-BF search is less than the one for BF-BT search. However, the important decisions with respect to the quality of a schedule may happen at the early stages of the scheduling activity, this increases the likelihood of missing the critical candidates for BT-BF search which employs backtracking search instead of best-first search at the early stage.

5.4 Scheduling an FMS with Routing Flexibility

The order in which a job visits different machines is predetermined in the classical job shop scheduling problem. Routing flexibility is a new feature of FMS scheduling. In a flexible manufacturing system, each operation of a job may be performed by any one of several

machines. Using the alternate routings in an FMS has the potential of increasing throughput rate by eliminating bottlenecks that block product flow, and prevent the whole system dead because of some machine breakdowns. However this added degree of freedom in an FMS increases the complexity of scheduling. Here additional choices associated with the technological constraints, in addition to the choices associated with machines should be effectively resolved.

Example 5.3: We consider an FMS with three multipurpose machines M_1, M_2 and M_3. There are four jobs, J_1, J_2, J_3 and J_4. The first three jobs have three processes each and the last one, J_4, has only two processes. Table 5.6 shows the job requirements. The operation times are shown in Table 5.7, where $OP_{i,j,k}$ represents the jth operation of the ith job is performed by the kth machine.

Table 5.6 Job Requirements for Example 5.3

Operations/Jobs	J_1	J_2	J_3	J_4
1	M_1/M_2	M_2	M_1/M_2	M_1
2	M_2/M_3	M_1/M_3	M_3	M_2/M_3
3	M_1	M_3	$M_1/M_2/M_3$	N/A

Table 5.7 Operation times for Example 5.3

Operation	Time	Operation	Time	Operation	Time	Operation	Time
$OP_{1,1,1}$	10	$OP_{2,1,2}$	5	$OP_{3,1,1}$	4	$OP_{4,1,1}$	11
$OP_{1,1,2}$	12	$OP_{2,2,1}$	9	$OP_{3,1,2}$	8	$OP_{4,2,2}$	9
$OP_{1,2,2}$	7	$OP_{2,2,3}$	13	$OP_{3,2,3}$	6	$OP_{4,2,3}$	9
$OP_{1,2,3}$	10	$OP_{2,3,3}$	8	$OP_{3,3,1}$	6		
$OP_{1,3,1}$	5			$OP_{3,3,2}$	2		
				$OP_{3,3,3}$	7		

We note that a job can be carried out more than one routing in Table 5.6. For instance, the first process of job J_1 can be performed at either M_1 or M_2. The second process of job J_1 can be performed at either M_2 or M_3, and the third process performed at M_1 only. The Petri net model of each job type is shown in Figures 5.5-5.8. The complete Petri net model for the system can be obtained by merging the places representing the shared machines in Figures 5.5-5.8.

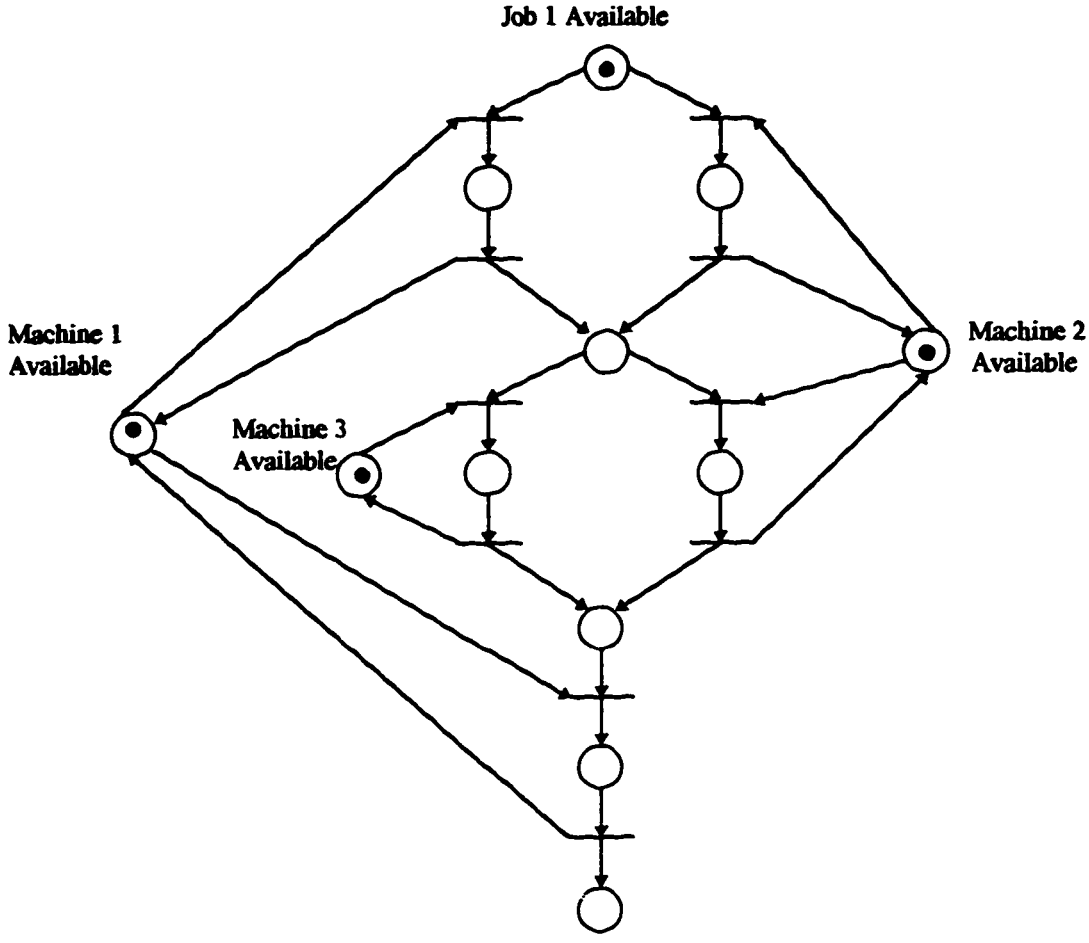


Figure 5.5 The Petri net model for sub-system Job 1 of Example 5.3

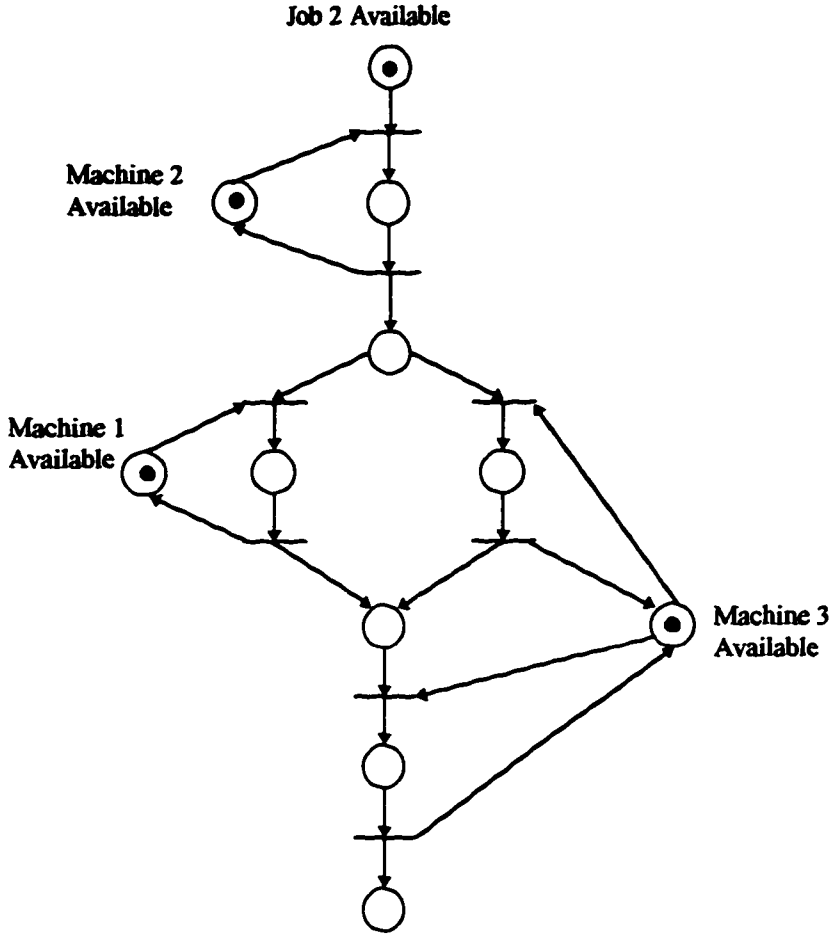


Figure 5.6 The Petri net model for sub-system Job 2 of Example 5.3

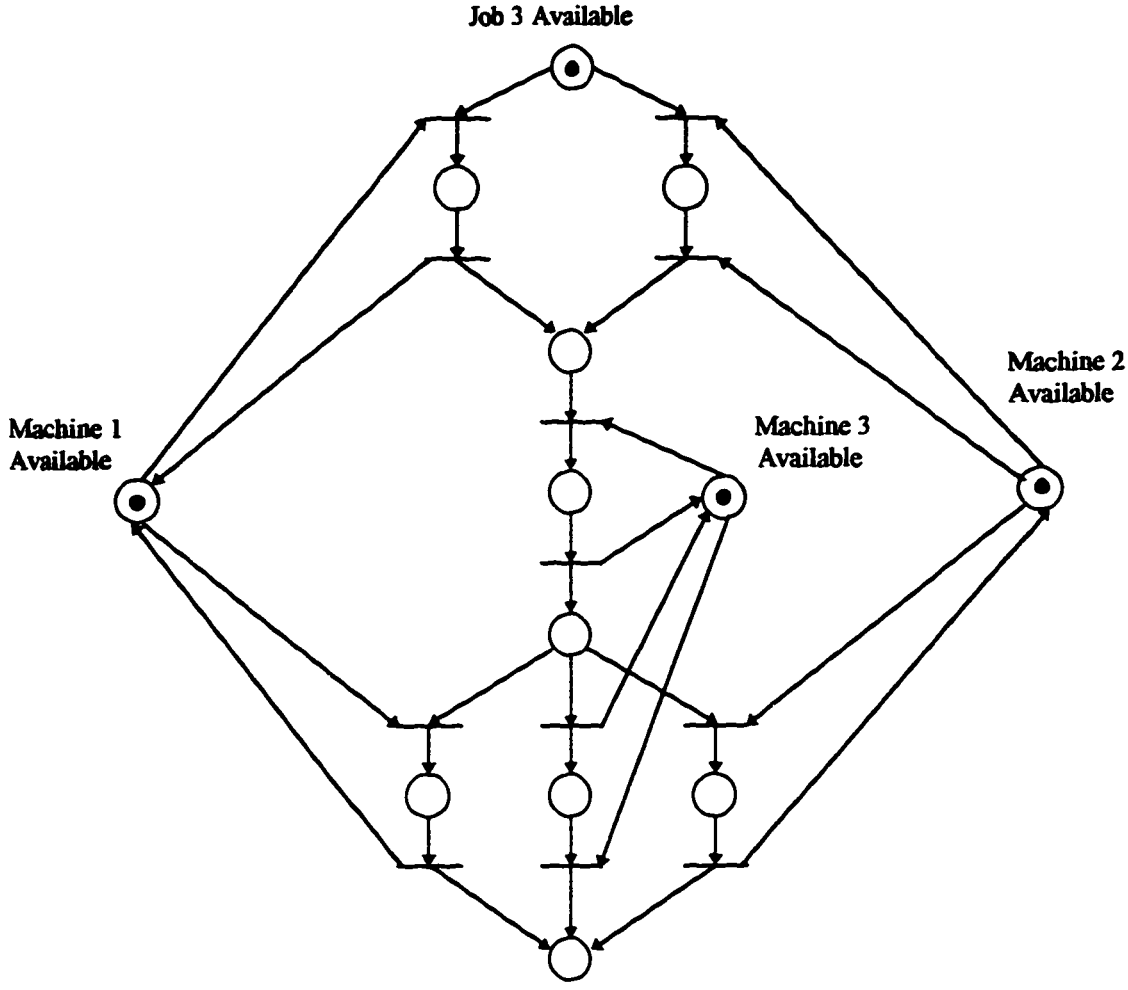


Figure 5.7 The Petri net model for sub-system Job 3 of Example 5.3

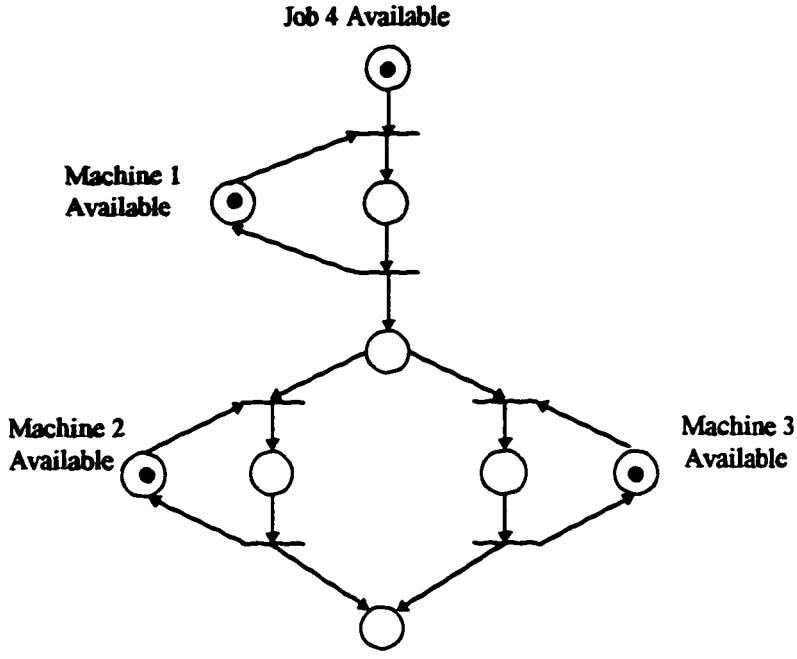


Figure 5.8 The Petri net model for sub-system Job 4 of Example 5.3

The hybrid heuristic search algorithm 5.4 (BT-BF) is used to solve the above problem considering different lot sizes. In each case, the depth bound is set to the half of the depth of a reachability graph. The depth of reachability graph is computed by multiplying the number of transitions in the Petri net model and the lot size. The hybrid BT-BF is compared with the standard depth-first search and heuristic dispatching rules.

We employ the following benchmark dispatching rules.

(i) A heuristic that chooses the fastest machine which can perform an operation if more than one machine exits, and then the shortest processing time (SPT) rule is used to sequence the operations among the parts waiting in the input buffer of a machine.

(ii) A heuristic that chooses a machine whose input buffer currently has the shortest queue, and then SPT rule is used to sequence the operations among the parts waiting in the input buffer of a machine.

Several different lot sizes of Example 5.3 are tested using the hybrid heuristic BT-BF, depth-first search and dispatching rules (i) and (ii). The results of the comparison are given in Table 5.8. In all cases tested, the presented hybrid method generates schedules with the shortest makespan. The depth-first search generates the worst results. The heuristic dispatching methods perform worse than the hybrid search, but better than the depth-first search. This is because the depth-first search explores its path using the totally uninformed knowledge. The dispatching rules seek the solutions using the local heuristics, while the hybrid method using the global information by ordering the decision candidates based on the performance indices. The heuristic dispatching rule that chooses a machine whose input buffer currently has the shortest queue performs better than the one that chooses the

fastest machine. This is expected because the heuristic that chooses a machine whose input buffer currently has the shortest queue is a dynamic rule, while the heuristic that chooses the fastest machine is a static one. Dynamic rules change priority indices with time and queue characteristics, whereas static ones keep priority indices constant as jobs travel through the plant.

Table 5.8 The scheduling results of Example 5.3 using different methods

Lot Size	Makespan			
	Depth-First	Dispatching(i)	Dispatching(ii)	Hybrid(BT-BF)
(1,1,1,1)	49	46	37	34
(5,5,5,5)	313	204	161	152
(10,10,10,10)	713	399	311	296
(20,20,20,20)	1513	789	613	597
(30,30,30,30)	2313	1179	921	874
(40,40,40,40)	3113	1569	1223	1172
(50,50,50,50)	3913	1959	1525	1468

For the computation results shown in Table 5.8, it is supposed that the buffer size for each machine is unlimited. Deadlock is completely avoided because large amounts of in-process storage are provided. However, it will cause excessive work in-process and an inefficient manufacturing system can result. Deadlock can arise from the explicit recognition of buffer space resources. The presented hybrid method always generates a

deadlock free schedule because of the explicit representation of deadlock states in the Petri net framework and backtracking capability in the search procedure. While deadlock could happen when employing the dispatching rules for scheduling. It is because the commonly used dispatching rules are "single pass" rules, namely, that once a decision is made by applying a rule, it will not reconsider the alternative courses of action.

For the above example, let's take the lot size case (30,30,30,30) and consider finite buffer size for each machine. Varying the buffer size N, scheduling results are obtained by applying the hybrid method and two dispatching methods, and shown in Table 5.9. From the table, we can see that the minimum buffer capacity 10 is required to avoid deadlock for the heuristic rule that chooses the fastest machine, and 8 for the heuristic rule that chooses a machine whose input buffer currently has the shortest queue. Figure 5.9 shows the comparison of makespan by varying the buffer capacity for lot size case (30,30,30,30) in Example 5.3. It not only shows that the hybrid method gives better performance of makespan than two dispatching methods for all range of the buffer size, and also that the makespan generated by the hybrid method constantly varies starting from a very small buffer capacity. The performance of dispatching method that chooses a machine whose input buffer currently has the shortest queue highly depends on the buffer size. The makespan of generated schedules arrives its best and keeps constant after the buffer size becomes 22, a very big number for lot size case (30,30,30,30).

Table 5.9 The scheduling results of Example 5.3 for finite buffer capacity

Buffer Size	Makespan		
	Dispatching(i)	Dispatching(ii)	Hybrid(BT-BF)
2	deadlock	deadlock	885
4	deadlock	deadlock	883
6	deadlock	deadlock	880
8	deadlock	1032	875
10	1189	1014	874
12	1179	981	874
14	1179	975	874
16	1179	963	874
18	1179	942	874
20	1179	930	874
22	1179	921	874
24	1179	921	874

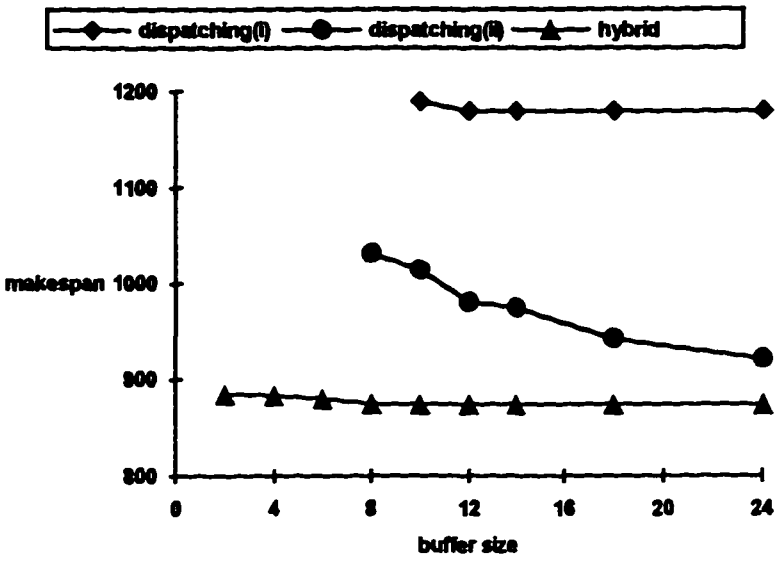


Figure 5.9 The comparison of makespan for the varying buffer capacity in the lot size case (30,30,30,30) of Example 5.3

5.5 Scheduling for a Semiconductor Test Facility

Semiconductor manufacturing is probably the most complicated manufacturing procedure in today's industry (Chen 1994). There are four main stages in a typical Integrated Circuit (IC) manufacturing process: wafer fabrication, wafer sort, assembly cycle, and final test. Production scheduling research in IC manufacturing has been conducted only in recent years (Uzsoy *et al.* 1991, Uzsoy *et al.* 1992, Lee *et al.* 1992, Chen 1994). Chen (1994) modeled the scheduling problem for IC sort and test facilities as an integer programming problem and used the Lagrangian relaxation technique to solve it. In this section, we adopt a scheduling example from [Chen 1994] to show Petri net's applicability in this area.

5.5.1 System Description

The first stage of IC production is called wafer fabrication. In wafer fabrication, the integrated circuits are manufactured on a silicon or gallium arsenide wafer using photolithography, etching, diffusion, and ion implantation processes. In the next stage, wafer sort, the individual circuits (dice) on a wafer are tested for functionality by means of electrical probes. Dice that fail to meet specifications are marked with an ink dot. The wafer then goes to assembly cycle, where the wafer is sawed; the defective dice are discarded; the good dice are bounded to the lead frames; the wires are bounded and then encapsulations are followed. After the assembly cycle, each IC ship is subjected to final tests to determine whether or not it is operating at the required specifications.

Example 5.4: The presented scheduling example (adopted from [Chen 1994]) will focus on the stages of wafer sort and final test. Because these two stages share some

expensive facilities such as testers, many companies perform them on the same test floor. Normally, a task for wafer sort requires a combination of a tester, prober, and some hardware facilities while a task for final test requires a combination of a tester, handler and some other hardware facilities. In this example, there are four types of tester, T1, T2, T3 and T4, two types of prober, P1 and P2, five types of handler, H1, H2, H3, H4 and H5, and seven types of hardware, Ha1, Ha2, Ha3, Ha4, Ha5, Ha6 and Ha7. The resource information is obtained from a real IC sort and test floor in San Jose, CA. Table 5.10 shows the number of each type of resource. Table 5.11 shows the possible resource combinations for wafer sort and final test. Each combination consists of a workcenter and looks as a single machine for scheduling. There are 30 jobs with a total of 90 operations to be scheduled. Table 5.12 shows the job requirements.

Table 5.10 The number of each type of facility for wafer sort and final test of Example 5.4

Facility	Quantity	Facility	Quantity
T1	8	H4	4
T2	4	H5	2
T3	4	Ha1	4
T4	4	Ha2	3
P1	6	Ha3	4
P2	4	Ha4	4
H1	6	Ha5	3
H2	4	Ha6	2
H3	4	Ha7	5

Table 5.11 Workcenters for wafer sort and final test of Example 5.4

Workcenter	Resource Combination
MS1	P1+T1+Ha1
MS2	P1+T2+Ha1
MS3	P2+T3+Ha2
MS4	P2+T4+Ha2
MT1	H1+T1+Ha3
MT2	H1+T2+Ha3
MT3	H1+T4+Ha1
MT4	H2+T1+Ha4
MT5	H2+T2+Ha5
MT6	H3+T1+Ha5
MT7	H3+T2+Ha6
MT8	H3+T3+Ha6
MT9	H4+T1+Ha7
MT10	H4+T2+Ha7
MT11	H4+T3+Ha3
MT12	H4+T4+Ha4
MT13	H5+T1+Ha4
MT14	H5+T2+Ha7
MT15	H5+T3+Ha7
MT16	H5+T4+Ha2

Table 5.12 Job requirements of Example 5.4

Job	Operation	Opr Time	Job	Operation	Opr Time
1	MS1	4	16	MT4	3
	MS2	3		MT3	4
	MT9	2		MT1	6
2	MS3	5	17	MT1	2
	MS2	2		MT5	4
	MT10	4		MT4	3
3	MS4	6	18	MT5	3
	MT8	3		MT4	4
	MT15	2		MT3	2
4	MS2	1	19	MT9	3
	MT16	2		MT8	4
	MT14	3		MT6	6
5	MS1	2	20	MT7	4
	MT10	3		MT2	3
	MT11	2		MT4	2
6	MS2	4	21	MT5	2
	MT7	7		MT6	4
	MT13	2		MT9	2
7	MS1	3	22	MT4	3
	MS2	2		MT7	5
	MT12	5		MT6	1
8	MS2	5	23	MT2	2
	MT8	3		MT1	3
	MT10	4		MT5	7
9	MT2	1	24	MT3	4
	MT1	2		MT4	2
	MT3	4		MT6	5
10	MT1	3	25	MT6	1
	MT3	2		MT1	4
	MT2	6		MT5	1
11	MT4	3	26	MT10	3
	MT1	3		MT13	4
	MT5	1		MT7	3
12	MT2	7	27	MT15	2
	MT1	2		MT16	3
	MT3	6		MT9	5
13	MS1	3	28	MT12	4
	MT4	2		MT8	2
	MT8	9		MT5	4
14	MT2	2	29	MT8	4
	MT1	3		MT9	2
	MT4	5		MT2	7
15	MS2	5	30	MT13	2
	MS3	4		MT11	6
	MS4	1		MT12	8

5.5.2 Scheduling Results Using Petri Nets

The Petri net model for sub-system job J1 is shown in Figure 5.10. The complete Petri net model for the whole system can be obtained by merging the shared resources (represented by Petri net places) of sub-systems from job J1 through job J30. Appendix lists C statements which generate the input function and output function of the complete Petri net model. The structure of graphical Petri net model is completely described by its input and output functions.

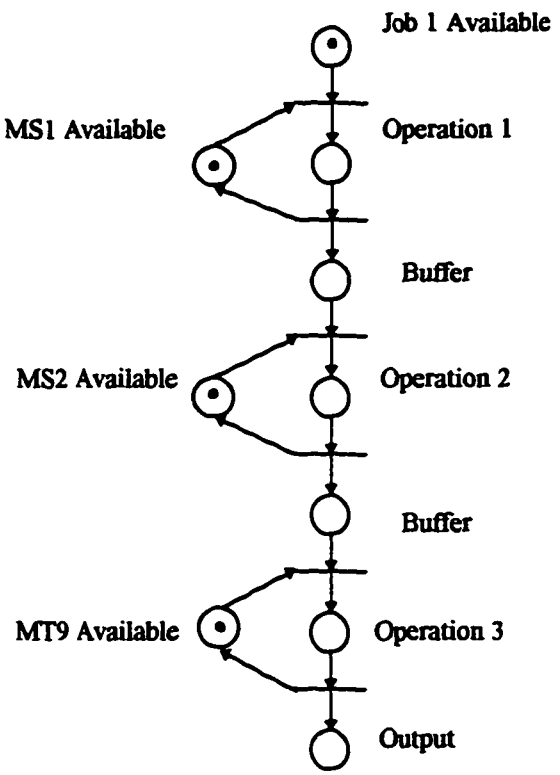


Figure 5.10 The Petri net model for the sub-system J1 in Example 5.4

The hybrid algorithm 5.4 is used to solve the above problem. Because the lot size we consider here is 1 for each job, the computation is not heavy. Thus we set the depth bound to 0 to perform a pure best-first search. Table 5.13 shows the scheduling results in the form of a transition firing sequence. The makespan of the resulting schedule is 30. The computation time for this schedule on a SUN Sparc 20 is 29 CPU sec. Chen (1994) modeled this scheduling problem as an integer programming problem and used the Lagrangian relaxation technique to solve it. The makespan of reported schedule is also 30, and computation time on a SUN Sparc 5 is 483 CPU sec. Due to the different computation platforms, the exact comparison of the computation is difficult to obtain. It is believed that these two approaches are similarly efficient for this example.

Table 5.13 The scheduling results of Example 5.4

Trans.	Firing time	Trans.	Firing time	Trans.	Firing time	Trans.	Firing time	Trans.	Firing Time	Trans.	Firing time
t_{29}	0	t_{85}	3	t_{32}	6	t_{142}	10	t_{162}	17	t_{167}	23
t_{28}	0	t_{51}	3	t_{4}	6	t_{176}	11	t_{99}	17	t_{153}	23
t_{27}	0	t_{48}	3	t_{174}	7	t_{151}	11	t_{90}	17	t_{101}	23
t_{26}	0	t_{42}	3	t_{115}	7	t_{140}	11	t_{120}	17	t_{131}	23
t_{25}	0	t_{72}	3	t_{113}	7	t_{64}	11	t_{87}	17	t_{45}	24
t_{24}	0	t_{6}	3	t_{143}	7	t_{175}	12	t_{77}	17	t_{10}	24
t_{23}	0	t_{58}	4	t_{136}	7	t_{173}	12	t_{3}	17	t_{92}	25
t_{22}	0	t_{88}	4	t_{109}	7	t_{169}	12	t_{33}	18	t_{122}	25
t_{21}	0	t_{78}	4	t_{148}	7	t_{141}	12	t_{63}	18	t_{157}	26
t_{20}	0	t_{57}	4	t_{91}	7	t_{103}	12	t_{5}	18	t_{159}	26
t_{19}	0	t_{53}	4	t_{121}	7	t_{133}	12	t_{168}	19	t_{152}	26
t_{18}	0	t_{49}	4	t_{66}	7	t_{30}	12	t_{150}	19	t_{40}	26
t_{16}	0	t_{81}	4	t_{17}	7	t_{9}	12	t_{117}	19	t_{70}	26
t_{14}	0	t_{43}	4	t_{119}	8	t_{13}	12	t_{147}	19	t_{8}	26
t_{12}	0	t_{79}	4	t_{149}	8	t_{171}	13	t_{67}	19	t_{38}	27
t_{2}	0	t_{116}	5	t_{108}	8	t_{170}	13	t_{93}	20	t_{161}	27
t_{1}	0	t_{102}	5	t_{132}	8	t_{138}	13	t_{123}	20	t_{95}	27
t_{54}	1	t_{83}	5	t_{34}	8	t_{178}	14	t_{156}	21	t_{125}	27
t_{59}	2	t_{44}	5	t_{0}	8	t_{94}	14	t_{107}	21	t_{75}	27
t_{89}	2	t_{31}	5	t_{112}	9	t_{124}	14	t_{137}	21	t_{100}	28
t_{56}	2	t_{74}	5	t_{111}	9	t_{37}	14	t_{41}	21	t_{130}	28
t_{86}	2	t_{61}	5	t_{145}	9	t_{60}	14	t_{129}	21	t_{68}	28
t_{52}	2	t_{118}	6	t_{104}	9	t_{11}	14	t_{71}	21	t_{160}	28
t_{50}	2	t_{146}	6	t_{134}	9	t_{39}	15	t_{15}	21	t_{155}	28
t_{80}	2	t_{114}	6	t_{96}	9	t_{69}	15	t_{97}	22	t_{98}	29
t_{46}	2	t_{110}	6	t_{73}	9	t_{179}	16	t_{127}	22	t_{105}	29
t_{84}	2	t_{106}	6	t_{166}	10	t_{154}	16	t_{62}	22	t_{135}	29
t_{76}	2	t_{144}	6	t_{164}	10	t_{126}	16	t_{35}	22	t_{128}	29
t_{13}	2	t_{82}	6	t_{139}	10	t_{172}	17	t_{65}	22	t_{158}	30
t_{55}	3	t_{36}	6	t_{47}	10	t_{163}	17	t_{177}	23	t_{165}	30

5.6 Summary

This chapter investigates FMS scheduling in a Petri net framework. Timed Petri nets provide an efficient method for representing concurrent activities, shared resources, precedence constraints and routing flexibility in FMS. We use a hybrid heuristic algorithm to search for an optimal or near-optimal deadlock-free schedule of an FMS in a Petri net scheme. The searching scheme is controllable, i.e., if one can afford the memory space required by a pure BF strategy, the pure BF search can be used to locate an optimal schedule. Otherwise, the hybrid BF-BT or BT-BF combination can be implemented, which can cut down the storage requirement at the cost of a smaller evaluation scope. The comparison of the presented hybrid method with depth-first search and commonly used dispatching rules is presented through an FMS scheduling example with routing flexibility. It shows that the performance of schedules generated by the presented hybrid method is significantly better than ones generated by depth-first search and two commonly used dispatching rules. Moreover, the hybrid method always generates a deadlock free schedule over the range of buffer capacity, while deadlocks can not be avoided until large amounts of in-process storage are provided for the dispatching methods.

Further work will be conducted in developing more efficient heuristic functions for Petri net based FMS scheduling problems, and setting different performance indices such as minimization of tardiness. The robustness of the resulting systems will also be investigated.

CHAPTER 6

SCHEDULING FMS WITH MATERIAL HANDLING AND BUFFER AVAILABILITY CONSIDERED

6.1 Introduction

Even though scheduling of flow-shops and job-shops has been extensively studied by many researchers (Baker 1974, French 1982, Carlier and Pinson 1989, Dudek *et al.* 1992, Van Laarhoven *et al.* 1992, Luh and Hoitomt 1993), most scheduling algorithms ignore both material handling and limited buffer space constraints. These algorithms are appropriate for manufacturing environments in which human intervention is significant and the equipment used is manual or hard automation (Leon and Wu 1994). For scheduling of automated manufacturing systems, explicit recognition should be given to auxiliary resources such as material handling and buffer space. This will increase the scheduling complexity because deadlock arises from explicit recognition of material handling and buffer space resources. The inappropriate scheduling decisions may lead to a deadlock state in which any part flow is inhibited and external intervention is required to reestablish the product flow. The methods for deadlock prevention and on-line avoidance have been investigated by some researchers (Banaszak and Krogh 1990, Viswanadham *et al.* 1990, Wysk *et al.* 1991, Zhou and DiCesare 1992, Hsieh and Chang 1994). These methods separate the deadlock control problem from the scheduling problem and ignore the schedules of resource allocations.

The purpose of this chapter is to schedule and control an automated manufacturing system considering both material handling and buffer space. To demonstrate the modeling

capability of Petri nets, the example is adopted from a recent paper presented by Ramaswamy and Joshi (1996), which generates deadlock-free schedules using the mathematical programming techniques.

6.2 System Description

Example 6.1 (Ramaswamy and Joshi 1996): An automated manufacturing system illustrated in Figure 6.1 has 3 machines, one robot and one part load/unload station. The robot is responsible for handling parts between machines, loading from the load station and unloading to the unload station. There are four jobs as shown in Table 6.1. The operation and transporting times are given in Table 6.2, where $O_{i,j,k}$ representing the jth operation of the ith job being performed by the kth machine, L_i representing the loading of the ith job from the load station, U_i representing the unloading of the ith job to the unload station, and $R_{i,j}$ representing the transporting the ith job for its jth operation.

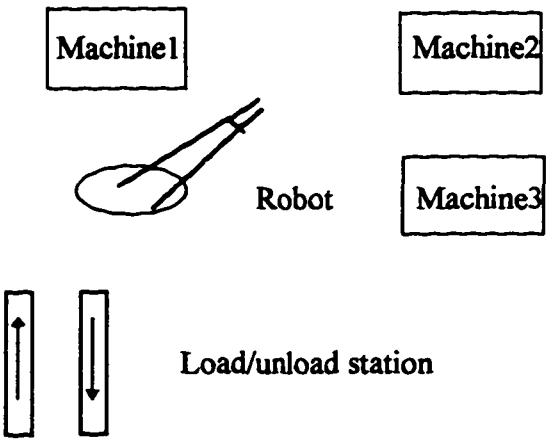


Figure 6.1 An automated manufacturing system for Example 6.1

Table 6.1 Job requirements for Example 6.1

Operations/Jobs	J_1	J_2	J_3	J_4
1	M_1	M_2	M_1	M_3
2	M_2	M_1	M_2	M_2
3	M_3	M_3	M_3	M_1

Table 6.2 Operation and transporting times for Example 6.1

Operation	Time	Transport	Time
$O_{1,1,1}$	40	L_1	5
$O_{1,2,2}$	100	$R_{1,2}$	3
$O_{1,3,3}$	36	$R_{1,3}$	5
$O_{2,1,2}$	65	U_1	4
$O_{2,2,1}$	45	L_2	5
$O_{2,3,3}$	98	$R_{2,2}$	3
$O_{3,1,1}$	212	$R_{2,3}$	6
$O_{3,2,2}$	73	U_2	4
$O_{3,3,3}$	32	L_3	6
$O_{4,1,3}$	35	$R_{3,2}$	7
$O_{4,2,2}$	65	$R_{3,3}$	4
$O_{4,3,1}$	55	U_3	5
		L_4	4
		$R_{4,2}$	3
		$R_{4,3}$	5
		U_4	5

6.3 Deadlock-prone and Deadlock-free Schedules

For the above system, if we follow the traditional assumptions in which the material handling action is ignored and unlimited intermediate storage is available, we can obtain an optimal schedule for minimizing the makespan by employing Algorithm 4.1 in Chapter 4. The Petri net sub-model for job J1 is shown in Figure 6.2. Similarly we can get Petri net sub-model for job J2, J3 and J4. The complete Petri net model for the system is obtained by merging sub-models through shared resources. The Gantt chart of the resulting optimal schedule is shown in Figure 6.3.

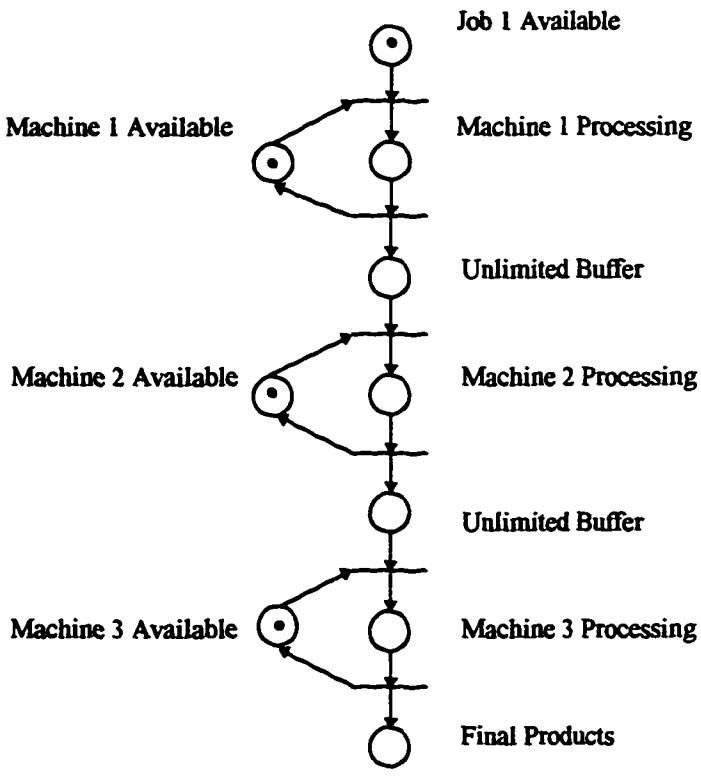


Figure 6.2 The Petri net model for the sub-system J1 under the assumptions that the material handling action is ignored and unlimited buffer space is available in Example 6.1

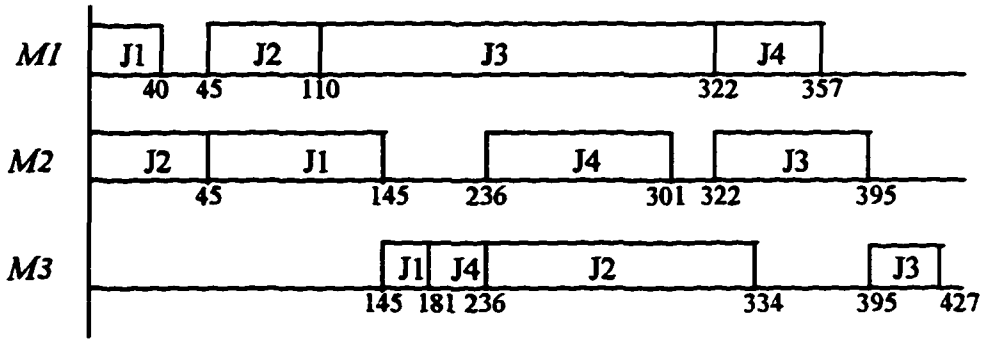


Figure 6.3 The optimal schedule without considering material handling and buffer availability in Example 6.1

In a practical manufacturing environment, the assumption of unlimited buffer space is unrealistic. For an automated manufacturing cell like Example 6.1, the number of intermediate storage slots is limited or even zero. Figure 6.4 shows the Petri net model for sub-system job J1 under the assumption of no intermediate storage is provided. For the schedule shown in Figure 6.3, it will lead into the deadlock state if no intermediate storage is provided. So it is an infeasible schedule even though the constraints for precedence relations and processing times are satisfied. The Petri net model for the intersection of job J1 and J2 shown in Figure 6.5 can clearly illustrate this situation. Figure 6.5(a) represents the initial state where all machines and jobs are available. According to the schedule shown in Figure 6.3, at the time instant 0, both enabled transitions t1 and t2 fire, which represents job J1 starts its first operation on Machine 1 and J2 on Machine 2. Job J1 finishes its first operation on Machine 1 at time instant 40 and then is waiting for its second operation on Machine 2, while job J2 finishes its first operation on Machine 2 at time instant 45 and is waiting for its second operation on Machine 1. This circulating

waiting situation leads into a deadlock state in which neither transition t3 nor t4 is firable as shown in Figure 6.5(b).

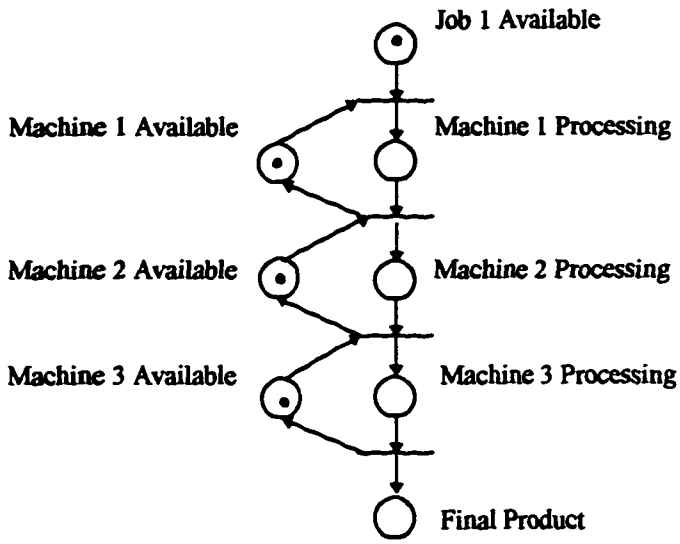


Figure 6.4 The Petri net model for the sub-system J1 under the assumption of no intermediate storage is provided in Example 6.1

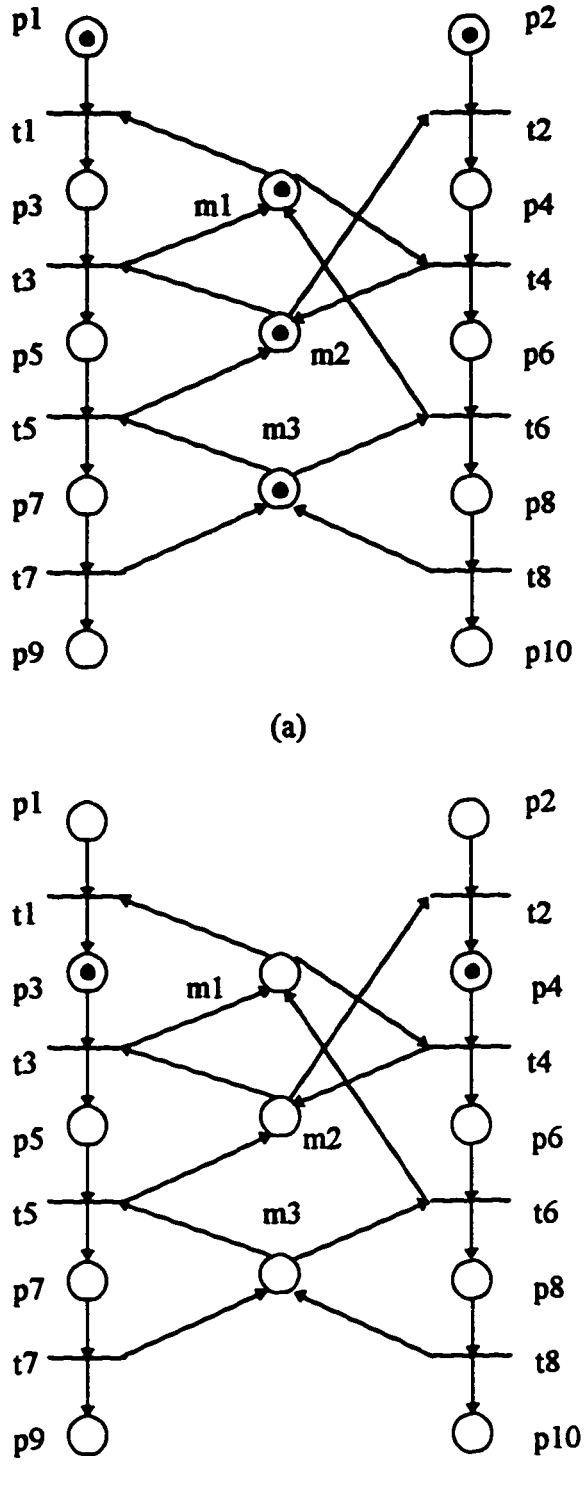


Figure 6.5 The initial state (a) and deadlock state (b) for no buffer case in Example 6.1

Modeling the sub-system J2, J3 and J4 as J1 depicted in Figure 6.4 and merging the sub-models, we can obtain an optimal deadlock-free schedule by employing the Algorithm 4.1 presented in Chapter 4. The resulting deadlock-free schedule is shown in Figure 6.6 in the form of Gantt chart.

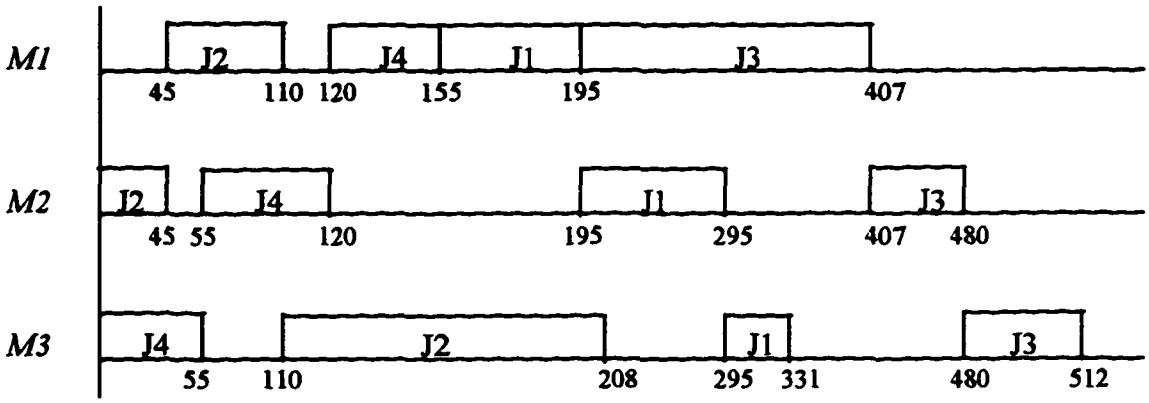


Figure 6.6 The optimal deadlock-free schedule for no buffer case in Example 6.1

6.4 Multiple Lot Sizes and Finite Buffer Sizes

In the above section, we model and schedule the system when its buffer size is infinite and when it is zero. Its job lot size for each job is 1. The Petri net models can explicitly and easily characterize features such as multiple lot sizes and finite buffer sizes in a practical manufacturing environment, while mathematical programming techniques have formulation difficulties for these features. In this section, we model and schedule the system of Example 6.1 for the cases of multiple lot sizes and finite buffer sizes which are not explored in Ramaswamy and Joshi's work (1996). In [Ramaswamy and Joshi 1996],

the lot size of each job is limited to 1, and the proposed deadlock-free scheme is only applicable to problems with m machines and $\lfloor m/2 \rfloor$ buffers.

Figure 6.7 shows the Petri net model for sub-system J1. Similarly we can construct models for J2, J3 and J4 and then merge them. The lot size is represented by the number of tokens in the place representing the number of jobs available and the buffer size by the number of tokens in the place representing the number of buffer spaces available. The system with different scenarios of lot sizes and buffer sizes is conveniently and visually modeled only by varying the available token of those corresponding places in the initial marking. For example in Figure 6.7, the lot size for the job J1 is 4 and the size of two intermediate buffer is 2.

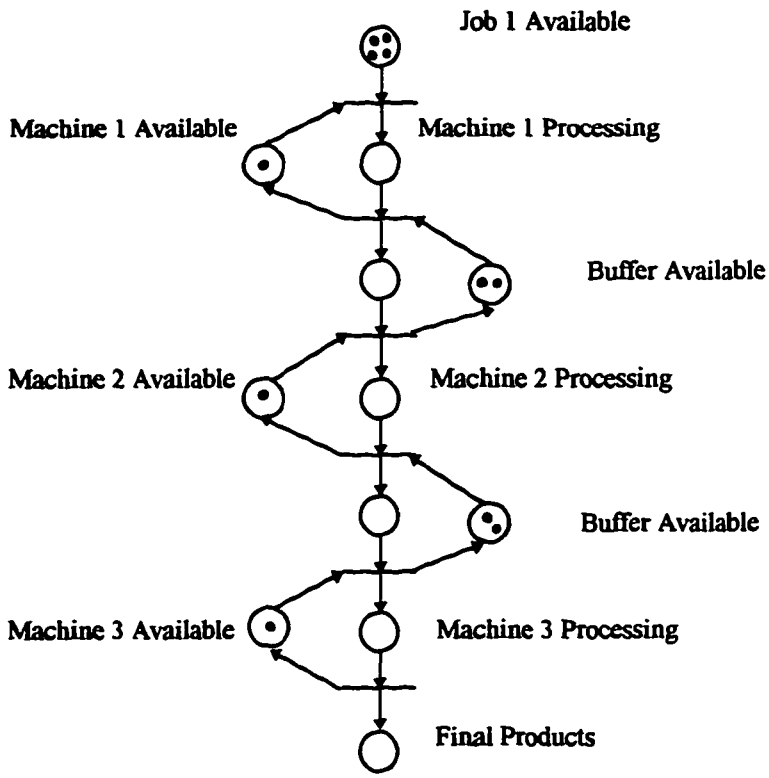


Figure 6.7 The Petri net model for the sub-system J1 with multiple lot sizes and finite intermediate buffer sizes in Example 6.1

Table 6.3 shows scheduling results for several different lot sizes of this example, and the size of intermediate buffers is set to 2. Note that we even can set the size of some jobs to zero without changing the Petri net model but the initial marking. Figure 6.8 shows scheduling results for a fixed lot size (20, 20, 20, 20) with a varying buffer size. All generated schedules are deadlock-free because of the use of the Petri net framework and backtracking capability of developed algorithms.

Table 6.3 The scheduling results for several different lot sizes of Example 6.1

Lot Size				Makespan
J1	J2	J3	J4	
2	2	0	2	455
5	4	6	3	1942
10	10	10	10	3638
20	20	20	20	7171

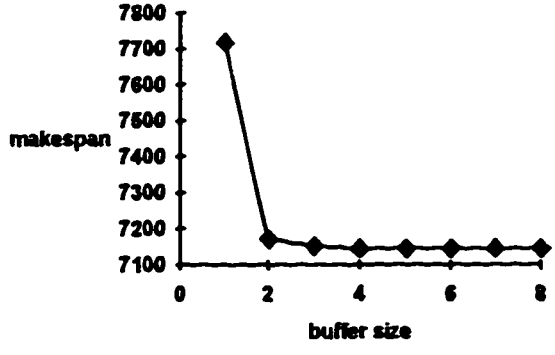


Figure 6.8 The scheduling results of lot size (20, 20, 20, 20) for the varying buffer size in Example 6.1

6.5 Scheduling the Operations of Material Handling

To schedule the operations of material handling, Figure 6.9 shows the Petri net model for the sub-system J1 with the material handler, i.e., robot in this example, as a shared resource. The Petri net model for the whole system is obtained by merging sub-models. Using Algorithm 4.1, we obtain the following optimal deadlock-free event sequences for each shared resource.

Machine 1: <Operation 2 of Job 2, Operation 3 of Job 4, Operation 1 of Job 1, Operation 2 of Job 3>;

Machine 2: <Operation 1 of Job 2, Operation 2 of Job 4, Operation 2 of Job 1, Operation 2 of Job 3>;

Machine 3: <Operation 1 of Job 4, Operation 3 of Job 2, Operation 3 of Job 1, Operation 3 of Job 3>;

Robot: <Transport Job 4 from load station to Machine 3, Transport Job 2 from load station to Machine 2, Transport Job 2 from Machine 2 to Machine 1, Transport Job 4 from Machine 3 to Machine 2, Transport Job 2 from Machine 1 to Machine 3, Transport Job 4 from Machine 2 to Machine 1, Transport Job 4 from Machine 1 to unload station, Transport Job 1 from load station to Machine 1, Transport Job 1 from Machine 1 to Machine 2, Transport Job 3 from load station to Machine 1, Transport Job 2 from Machine 3 to unload station, Transport Job 1 from Machine 2 to Machine 3, Transport Job 1 from Machine 3 to unload station, Transport Job 3 from Machine 1 to Machine 2, Transport Job 3 from Machine 2 to Machine 3, Transport Job 3 from Machine 3 to unload station>.

The optimal deadlock-free schedule is shown in Figure 6.10 in the form of Gantt chart and the makespan is 560. By employing Algorithm 4.1, the computation time is 0.13 CPU seconds to generate the schedule in Figure 6.6 when only buffer availablity is considered, 0.41 CPU seconds to generate the schedule in Figure 6.10 when both material handling and buffer availablity are considered in SUN Sparc 20. In [Ramaswamy and Joshi 1996], the CPU time is increased from 0.71 seconds to 67.0 seconds in IBM ES/3090-600S for the above two schedules. It is clear that the use of Petri nets for optimal deadlock-free scheduling results in a significantly small variation in computation. This is not the case for mathematical programming case, however.

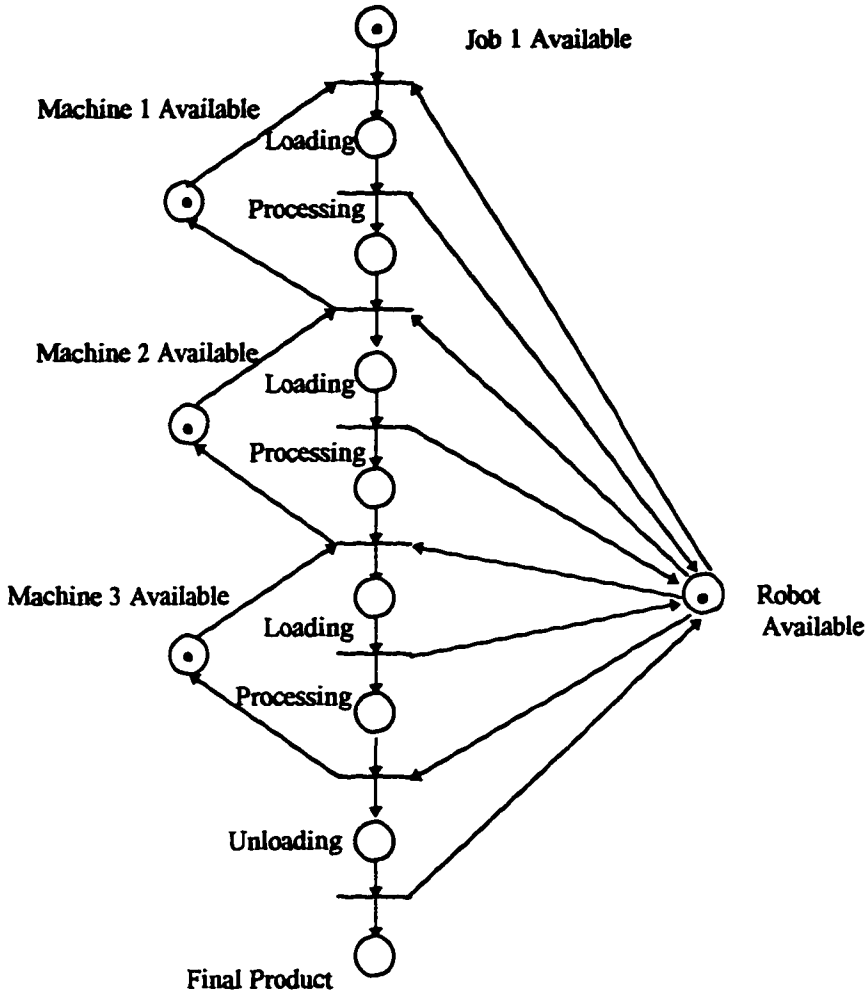


Figure 6.9 The Petri net model for the sub-system J1 with material handling operations in Example 6.1

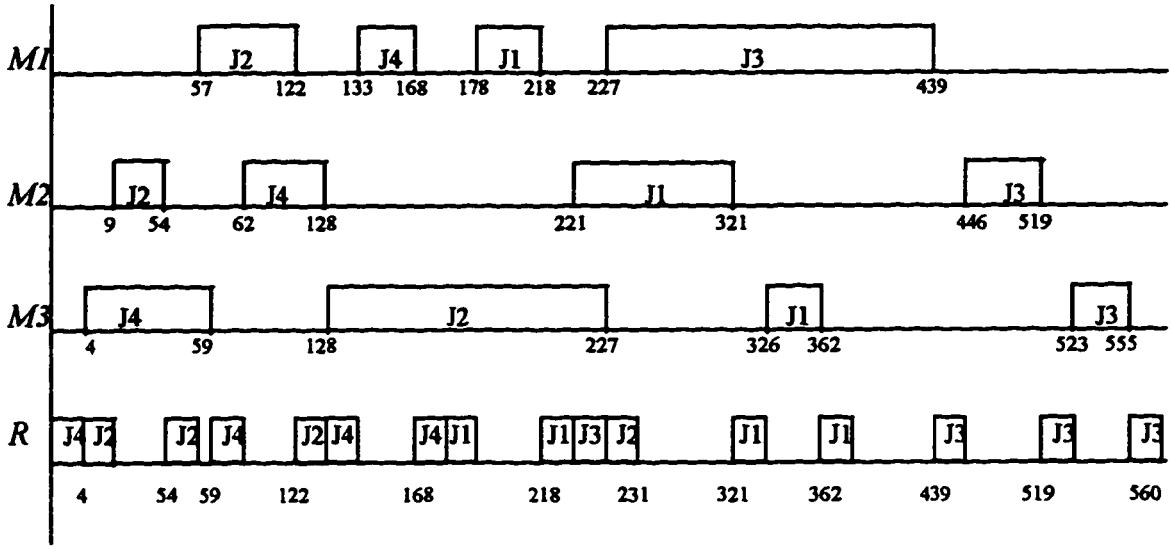


Figure 6.10 The optimal deadlock-free schedule including the operations of material handling in Example 6.1

CHAPTER 7

MULTI-CRITERION SCHEDULING BASED ON PETRI NETS AND FUZZY DISPATCHING RULES

7.1 Introduction

Studies on multi-criterion scheduling are of relatively recent origin. For simple structured problems such as single-machine scheduling, branch-and-bound based algorithms are reported to determine the optimal schedule with respect to a linear combination of two scheduling objectives (Sen and Gupta 1983, Chen *et al* 1994). However, depending on the size of problem, it is difficult or impossible to derive an optimal schedule for a multi-criterion problem. Computer simulation using heuristic dispatching rules has been commonly used for FMS scheduling (Montazeri and Wassenhove 1990). The dispatching rules, such as SPT (Shortest Processing Time), EDD (Earliest Due Date), S/RO (Slack per Remaining Operation), and FCFS (First Come First Served), are employed to resolve conflicts between the jobs in the input queue of available machine tools. These rules can be classified as being static or dynamic, e.g., SPT and EDD (assuming processing times and due dates are fixed) are static, while S/RO is dynamic. Each of these dispatching rules aims at satisfying a single criterion. A rule that performs well when one measure is used may not do well for another measure (Blackstone *et al.* 1982). Fuzzy logic based methods are reported to deal with multicriteria decision-making problems (Watanabe, Tokumaru and Nakajima 1992, Grabot and Geneste 1994, Custodio *et al* 1994). Considering the linguistic characteristics of criteria, Watanabe *et al.* (1992) employed fuzzy inference to take both profit and slack criteria into account.

The goal of this chapter is to propose a way to employ fuzzy dispatching rules in a Petri net framework. It allows to obtain a compromise between the satisfaction of several criteria. Petri nets can concisely model the concurrent and asynchronous activities, shared resources, and precedence constraints in FMS. Associating the time with places or transitions in a Petri net allows it to describe a system whose functioning is time-dependent. Since each transition in a conflict set corresponds to each part type which competes for an available resource for the next operation, the dispatching rules are employed to select one of the enabled transitions to fire in each conflict set. Considering the fact that no dispatching rule has been shown to generate good performance simultaneously for several criteria, combination rules derived from fuzzy logic are used. The specific objectives of this chapter are:

1. To derive fuzzy dispatching rules from elementary dispatching rules based on fuzzy logic.
2. To present an algorithm for multi-criterion scheduling based on timed (place) Petri nets. The Petri net model resolves conflicting transition firings using fuzzy dispatching rules.
3. To illustrate the method through a scheduling example.

7.2 Fuzzy Dispatching Rules

Simulation research on the analysis of performance of different dispatching rules has been reported in the literature (Blackstone, Phillips and Hogg 1982, Montazeri and Wassenhove 1990, Karsiti *et al.* 1992). These studies give very few general results, since the performance of dispatching rules depends strongly on the criterion chosen and the

environment of manufacturing systems. Generally simple dispatching rules are separated into three classes, rules involving processing time, rules involving due dates and rules involving neither processing times nor due dates.

Rules involving processing time. Some of these are:

(a) Shortest processing time: Select the job with the shortest processing time at the current operation.

(b) Least total remaining processing time: Select the job with the least total remaining processing time.

(c) Most total remaining processing time: Select the job with the largest total remaining processing time.

Rules involving due dates. Some of these are:

(a) Earliest due date: Select the job with the earliest due date.

(b) Slack time: Select the job with the lowest slack time.

(c) Slack per remaining operation: Select a job with the smallest ratio of slack to operations remaining to be performed.

Rules involving neither processing times nor due dates. Some of these are:

(a) First come, first served: Select a job that has been in the machine's queue the longest.

(b) First in system, first served: Select a job that has bee on the shop floor the longest.

(c) Random: Select a job at random.

The purpose here is not to give an extensive performance evaluation of these dispatching rules, which have been investigated in the literature mentioned above. We shall

focus our attention on some of the most common rules to demonstrate the advantage of combined rules for multi-criterion scheduling.

The researchers have proposed several ways to combine elementary dispatching rules (Blackstone, Phillips and Hogg 1982). In this chapter, based on fuzzy logic, we derive a fuzzy dispatching rule which can describe multiple-variety and the linguistic-form characteristics of the scheduling objectives in flexible manufacturing. We summarize some concepts and methods of fuzzy set and fuzzy logic needed to present the results in this thesis (Lee 1990, Klir and Folger 1991).

A *crisp set* assigns a value of either 1 or 0 to each individual in the universal set to discriminate between members and nonmembers of the set. If the values assigned to the elements of the universal set fall within a specified range and indicate the membership grade of these elements in the set, we obtain a *fuzzy set*. A fuzzy set F in a universe of discourse U is characterized by a *membership function* μ_F which takes values in the interval [0,1], namely, μ_F: $U \rightarrow [0,1]$.

The use of fuzzy sets provides a basis for the manipulation of *linguistic variables* which may be vague and imprecise. The values of a linguistic variable are defined in *linguistic terms*. For example, if *operation time* is interpreted as a linguistic variable, then its values could be defined in the term set {short, long, very long, ...}, while each term is characterized by a *fuzzy number*. A fuzzy number is a convex and normalized fuzzy set defined on real line R whose membership function is piecewise continuous.

The priority of job processing might often be characterized by a set of linguistic description rules based on expert knowledge. The rules are usually taken in the form of

IF (a set of conditions are satisfied) THEN (a set of consequences can be inferred).

We consider a rule base that has two fuzzy rules as follows:

$$R_1: \text{ if } x \text{ is } A_1 \text{ and } y \text{ is } B_1 \text{ then } z \text{ is } C_1,$$

$$R_2: \text{ if } x \text{ is } A_2 \text{ and } y \text{ is } B_2 \text{ then } z \text{ is } C_2.$$

where x, y and z are linguistic variables representing the process state variables and the output control variable, and A_i, B_i and C_i are the linguistic values of the linguistic variables x, y and z, $i = 1, 2$ respectively.

Now we have two crisp inputs x_0 and y_0, the contribution of the first and second rules to the consequence can be expressed using the firing strengths α_1 and α_2 with

$$\alpha_1 = \mu_{A1}(x_0) \wedge \mu_{B1}(y_0),$$

$$\alpha_2 = \mu_{A2}(x_0) \wedge \mu_{B2}(y_0).$$

where "$\wedge$" representing the minimum operation or the algebraic product.

Tsukamoto (1979) proposed a fuzzy reasoning method when the membership functions of fuzzy sets A_i, B_i and C_i are monotonous. Supposed that the result inferred from the first rule is α_1 such that $\alpha_1 = \mu_{C1}(z_1)$, and the result inferred from the second rule is α_2 such that $\alpha_2 = \mu_{C2}(z_2)$, a crisp consequent output is given by

$$z_0 = \frac{\alpha_1 z_1 + \alpha_2 z_2}{\alpha_1 + \alpha_2}. \qquad (1)$$

Based on SPT (Shortest Processing Time) and S/RO (Slack per Remaining Operation), two fuzzy dispatching rules for determining the priority of the jobs are introduced as follows:

If (imminent processing time is short) & (slack per remaining operation is short) then (priority is high),

If (imminent processing time is long) & (slack per remaining operation is long) then (priority is low).

where job slack equals to the due date minus current time and remaining processing time.

The linguistic variables *imminent processing time, slack per remaining operation* and *priority* are characterized by the membership functions of their corresponding value terms which are shown in Figure 7.1 (a), (b) and (c) respectively.

Using the membership functions given in Figure 7.1, precise priority of a job in conflict can be obtained through formula (1) for the given crisp imminent processing time and slack per remaining operation of corresponding job, where we employ the algebraic product as operation "$\wedge$" for preserving the contribution of each input variable.

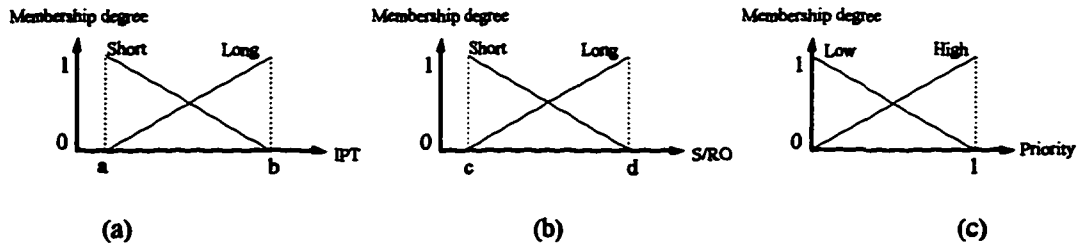


Figure 7.1 Membership functions

7.3 Scheduling Using Timed Petri Nets and Fuzzy Dispatching Rules

We use deterministic P-timed Petri nets modeling FMS for scheduling. In our modeling process, a place represents a resource status or an operation, a transition represents either start or completion of an event or operation process, and the stop transition for one activity will be the same as the start transition for the next activity. Token(s) in the resource place indicates that the resource is available and no token indicates that it is not available. A token in the operation place represents that the operation is being executed and no token shows none being performed.

In a P-timed Petri net, at any time t, the present marking m is the sum of the available tokens and unavailable tokens, which represent the concurrency of operations associated with the places. By keeping track of time for marked places, a transition is enabled for the present marking only if it is enabled by the available tokens. For the scheme of functioning at the maximal speed, a transition is fired as soon as it is enabled, and this firing has a zero duration. Firing a transition is carried out by removing one available token from each input place and depositing a token to each output place. The deposited token in place p_i is unavailable for time interval (t, t+d_i), where t is the current time and d_i is the timing delay

associated with place p_i. In our FMS modeling, this token unavailable interval corresponds to the working duration of a machine processing a part. A *structural conflict* exists when two or more transitions share the same place as an input, e.g., in the case of sharing of a common resource in an FMS scheduling problem. An *(effective) conflicting set* $\Gamma(m)$ is defined as a set of enabled transitions for the marking m, if for every pair of transitions in the set, firing of one transition disables another. Each transition in a conflicting set corresponds to a start activity of a job type which competes for an available resource (machine) for imminent operation. The *dynamic* priority obtained from the fuzzy dispatching rules is used to select one of the enabled transitions to fire in each conflicting set. Based on the execution scheme of timed Petri nets functioning at the maximal speed, we give the following scheduling algorithm for P-timed Petri nets modeling an FMS. The schedules generated from the Petri nets functioning at maximal speed are *nondelay schedules*. The nondelay schedules are ones such that a machine is never idle when its queue is nonempty. In this chapter, the problems are confined to FMS with fixed routings. In this case, every pair of conflicting sets $\Gamma_i(m)$ and $\Gamma_j(m)$ are disjoint.

Algorithm 7.1:

Step 1: Initialization of the marking. The time-ordered sequence only contains the initial time $t = 0$. All the initial tokens are available and $J = \varnothing$. Go to *Step 3*.

Step 2: Consider the first time t of the time-order sequence.

Step 2.1: If the marking m is the final marking, then *End*. The schedule is a list of operation start times which are the firing instants of transitions representing the start events of corresponding operations. Otherwise,

Step 2.2: Add set J of the tokens which become available at instant t to the set of tokens already available.

Step 3: Erase instant t from the time-ordered sequence.

Step 3.1: If the set of enabled transitions is empty, go to *Step 2*. Otherwise,

Step 3.2: Determine the (effective) conflicting sets $\Gamma_1(m)$, $\Gamma_2(m)$, ...$\Gamma_l(m)$.

Step 3.3: For every conflicting set $\Gamma_i(m)$ (i=1,2,...l), Using the *fuzzy* dispatching rules determine the *crisp* firing priority (*fuzzy reasoning, formula (1)*) of each transition in the set. The transition with the highest priority is selected to fire (if two or more, select one at random).

Step 3.4: Fire *all* transitions selected in *Step 3.3*. Add, to the time-ordered sequence, the instants where the tokens deposited become available. Go to *Step 2*.

7.4 An Example

Example 7.1: Consider a four-machine, four-job scheduling problem shown in Table 7.1.

Table 7.1 Job Requirements of Example 7.1

	Job			
Operation	J1	J2	J3	J4
1	(M1,5)	(M2,8)	(M2,3)	(M1,2)
2	(M2,3)	(M4,6)	(M4,6)	(M3,2)
3	(M3,7)	(M3,2)	(M3,4)	(M2,7)
4	(M4,1)	(M1,5)	(M1,4)	(M4,3)
Due Date	35	35	35	35

In this example, we have four machines M1, M2, M3 and M4, four jobs J1, J2, J3 and J4. The precedence relationships among the operations and working time of each operation on the assigned machine for each job are shown in the table. For an FMS having control over due dates, the due date information for each job is also indicated in the table.

The goal is to find a schedule that obtains a compromise between the satisfaction of several criteria. Among them are minimizing *average flow time*, the time required to complete all jobs (*makespan*), *average tardiness* and *maximum tardiness*.

The bottom-up method is used to synthesize the system, i.e., the system is partitioned into sub-systems according to the job types, then sub-models are constructed for each sub-systems, and a complete net model for the entire process is obtained by merging Petri nets of the sub-systems through the places representing the shared machines. The Petri net sub-model for job J1 is shown in Figure 7.2. Similarly we can get Petri net model for job J2, J3 and J4. The complete Perti net model for the system is obtained by merging Petri nets of the job types J1, J2, J3 and J4.

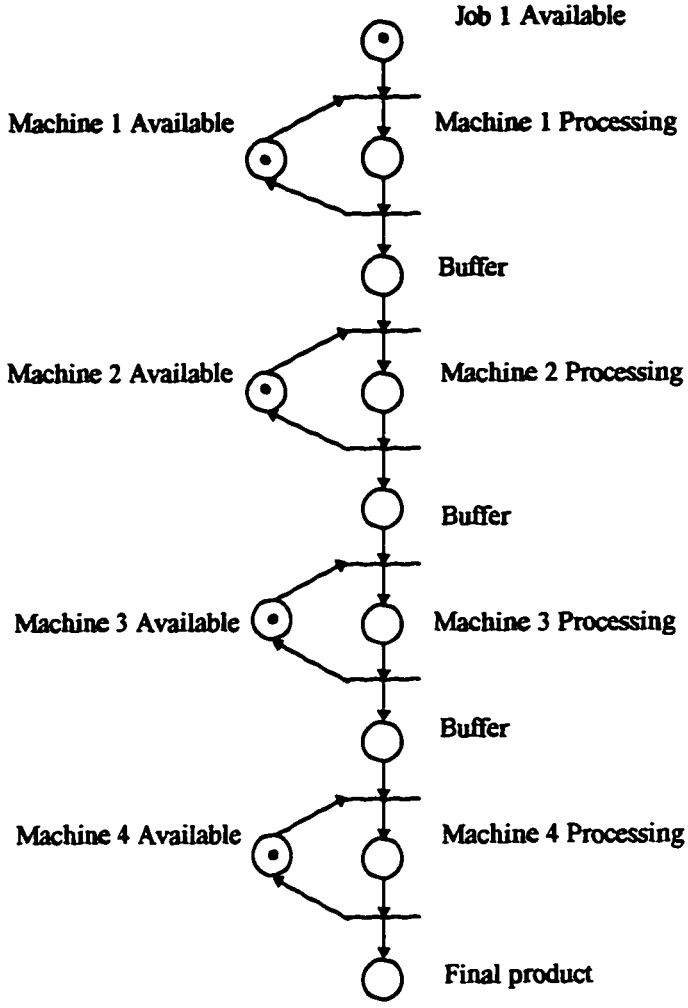


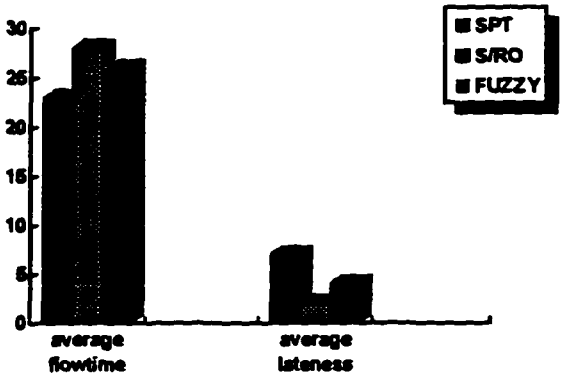
Figure 7.2 The petri net model of the sub-system Job 1 in Example 7.1

Based on the Petri nets models, SPT, S/RO and fuzzy dispatching rules are tested for scheduling problem of the above example. The used performance criteria are average flowtime and average lateness. The lateness is the amount of time by which the completion time of the job exceeds its due data, with a negative lateness indicating an early completion. We use an absolute value of lateness for each job in computing the average lateness in light of the just-in-time concept. Figure 7.3 (a), (b) and (c) show the performance results using different dispatching rules for each job size 1, 5 and 20 respectively, assuming each job has the same lot size in these four cases. The fuzzy dispatching rules which combine SPT and S/RO based on fuzzy logic obtain a compromise between the average flowtime and the average lateness.

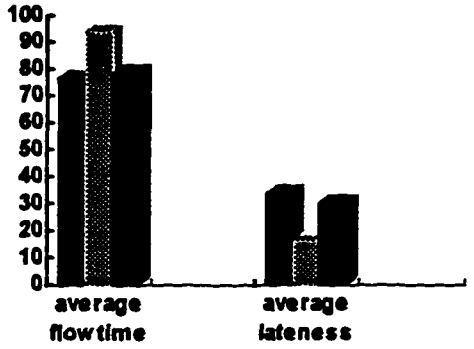
7.4 Summary

Heuristic dispatching rules are often adopted to determine the priority of jobs for processing in flexible manufacturing. Fuzzy dispatching rules can represent the multiple-variety and the linguistic-form characteristics of the scheduling objectives. This research combines the Petri nets and heuristic dispatching rules into a unified scheme to explore the modeling ability of Petri nets and decision efficiency of dispatching rules. Compared with the simulation model (Grabot *et al.* 1994), our model is easier to develop and can be directly implemented into Petri net controllers.

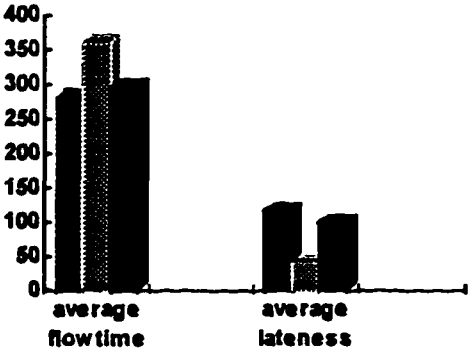
However, the present research just demonstrates a direction, the simple fuzzy dispatching rules should be developed into a more comprehensive fuzzy dispatching rule base. But because of unavailability of expert knowledge, we do not explore it further.



(a)



(b)



(c)

Figure 7.3 Average flowtime and lateness with each job size 1 (a), 5 (b) and 20 (c)

The above algorithm is implemented based on single pass priority dispatching rules, in which, once a decision made by the operation of the rule, it is implemented without reconsideration of alternative courses of action. Hence it cannot prevent the deadlock states. For the computation results shown in Figure 7.4, unlimited amounts of in-process storage are supposed. However, deadlock can arise from the explicit recognition of buffer space resources, which we have demonstrated and resolved in Chapter 5. That is why we develop deadlock-free scheduling algorithms in Chapters 4 and 5.

CHAPTER 8

CONCLUSIONS

8.1 Contributions

A flexible manufacturing system (FMS) is a computerized production system that can simultaneously manufacture multiple types of products using various resources such as robots and multi-purpose machines. The central problems associated with design of flexible manufacturing systems are related to process planning, scheduling, coordination control, and monitoring. This thesis presents a Petri net based method for deadlock-free scheduling and discrete event control of flexible manufacturing systems. Petri nets are a graphical and mathematical modeling tool applicable to many systems. Petri nets can explicitly and concisely model the concurrent and asynchronous activities, multi-layer resource sharing, routing flexibility, limited buffers and precedence constraints in FMS. Using the concept of markings, the evolution of the system can be completely tracked by the reachability graph of the net. Associating the time with places or transitions in a Petri net allows it to describe a system whose functioning is time-dependent. The problem of FMS deadlock has been ignored by most research in scheduling and control based on methods such as mathematical programming, heuristics dispatching and knowledge-based, and control theoretic methods, while Petri nets can provide an explicit and convenient way for considering deadlock situations in FMSs such that a deadlock-free scheduling and control system can be designed. They are easier to represent and understand compared with the algebraic equations and inequalities used in mathematical programming.

The contributions of this work are multifold. First, it develops a methodology for discrete event controller synthesis for a class of flexible manufacturing systems in a timed Petri net framework. The resulting Petri nets have the desired qualitative properties of liveness, boundedness (safeness), and reversibility, which imply freedom from deadlock, no capacity overflow, and cyclic behavior respectively. This precludes the costly mathematical analysis for these properties and reduces on-line computation overhead to avoid deadlocks. The performances and sensitivities of resulting Petri nets are evaluated. Even though there are several studies in this aspect (Krogh and Beck 1986, Koh and DiCesare 1991, Zhou, DiCesare and Desrochers 1992), for the system with multi-layer resource-sharing and different products sets manufactured concurrently, modeling of a Petri net controller with desirable properties becomes extremely difficult based on their methods. Their methods focus on the logical behavior only. The developed method starts with a bottom-up approach and search for the best performance sequence of events and then synthesize the desirable Petri net controllers.

Second, it introduces a hybrid heuristic search algorithm based on Petri nets for deadlock-free scheduling of flexible manufacturing systems. The issues such as deadlocking, routing flexibility, multiple lot sizes, limited buffer sizes and material handling (loading/unloading) are explored. Even though Lee and DiCesare (1994) presented a scheduling method using Petri nets and heuristic search, their proposed heuristic functions do not guarantee to satisfy the admissible condition (Pearl 1984). Moreover, no deadlock issues are discussed in their demonstrated examples because they always put an intermediate place which serves as the role of a buffer with unlimited

capacity between two operations. Recently, Ramaswamy and Joshi (1996) applied integer programming techniques for deadlock-free scheduling of automated manufacturing workstations. Although both material handling and buffer space are explicitly considered in their generated schedules, the proposed deadlock-free scheme is only applicable to problems with m machines and $\lfloor m/2 \rfloor$ buffers. Other characteristics of FMS such as multiple lot sizes, multiple buffers and routing flexibility are not explored in their work.

Third, it proposes a way to employ fuzzy dispatching rules in a Petri net framework for multi-criterion scheduling. Compared with the simulation model (Grabot *et al.* 1994), our model is easier to develop and can be directly implemented into Petri net controllers.

Finally, it shows the effectiveness of developed methods through examples compared with benchmark dispatching rules, integer programming and Lagrangian relaxation approaches.

8.2 Further Research

The present work has its limitations. These limitations can be overcome with further research.

1. Given the limited degrees of freedom for part movement and staging, it is very important to decide when to introduce a new part into the system. Many parts in the system can lead to congestion, while few parts in the system result in under-utilization of equipment. Control theoretic based methods provide an effective way to control the part release problem. The presented Petri net based method does not provide the part release control scheme. This has to be addressed in many

manufacturing applications.

2. One of the major problems in simulating FMS is to describe stochastic behavior, such as failures of machine tools, repair time, variations of processing time. The presented work is based on deterministic timed Petri nets and does not handle the stochastic situations. The work on the evaluation of the sensitivity in this thesis is a good start to this problem.
3. The synthesized discrete event controller is a marked graph, which is not applicable to systems containing routing flexibility, assembly and disassembly processes.
4. Even though the employed heuristic function is admissible, a more effective admissible heuristic function is desired to reduce the search effort. For hybrid search schemes, instead of employing BT on the top and BF on the bottom or vice versa, a more effective way should be employing BT and BF interchangeably based on the current state. This requires a comprehensive analysis of proposed schemes.
5. The research presented in Chapter 7 demonstrates a research direction which may lead to many important contributions. The simple fuzzy dispatching rules should be developed into a more comprehensive fuzzy dispatching rule base. Moreover, the work should contain deadlock avoidance scheme.
6. The number of each type of resources is supposed to be 1. This is not the case in some manufacturing systems. For example, a system may have two or more same type machines. Colored Petri nets have their potential to model this kind of systems.

7. Even though we present an example for scheduling semiconductor manufacturing, some unique features of semiconductor lines are not explored. These features include random entries of parts, reentrant product flows and part disassembly.
8. The FMS examples demonstrated in this work are still confined to the academic research. A more practical FMS should be investigated.

APPENDIX

THE INPUT AND OUTPUT FUNCTIONS OF PETRI NET MODEL OF EXAMPLE 5.4

This appendix contains C statements which generate the input function and output function of the complete Petri net model of Example 5.4

```
numPlaces = 230;      /*number of places in the net*/
numTrans = 180;       /*number of transitions in the net*/
numJobs = 30;         /*number of jobs to be scheduled*/
numMachines = 20;     /*number of resources*/

/*Initialize all entries of input and output matrixes to 0*/
for ( int i = 0; i < numPlaces; i++ )
   for ( int j = 0; j < numTrans; j++ )
      {
       inputArc[i][j] = 0;
       outputArc[i][j] = 0;
       }

/*For arcs existing from places (not including resource places) to transitions, set
corresponding entries in the input matrix to 1*/
for ( i = 0; i < numTrans; i++ )
   inputArc[i][i] = 1;

/*For arcs existing from transitions to places (not including resource places), set
corresponding entries in the output matrix to 1*/
for ( j = 0; j < numTrans; j++ )
   inputArc[j+numJobs][i] = 1;


/*For the arcs connecting to and from shared resource places, set corresponding entries in
the input and output matrixes to 1*/

j = 0;  /*For Job 1*/
inputArc[MS1][j] = 1; inputArc[MS2][j+60] = 1; inputArc[MT9][j+120] = 1;
outputArc[MS1][j+30] = 1; outputArc[MS2][j+90] = 1; outputArc[MT9][j+150] = 1;
j = 1;  /*For Job 2*/
inputArc[MS3][j] = 1; inputArc[MS2][j+60] = 1; inputArc[MT10][j+120] = 1;
outputArc[MS3][j+30] = 1; outputArc[MS2][j+90] = 1; outputArc[MT10][j+150] = 1;
j = 2;  /*For Job 3*/
inputArc[MS4][j] = 1; inputArc[MT8][j+60] = 1; inputArc[MT15][j+120] = 1;
outputArc[MS4][j+30] = 1; outputArc[MT8][j+90] = 1; outputArc[MT15][j+150] = 1;
j = 3;  /*For Job 4*/
inputArc[MS2][j] = 1; inputArc[MT16][j+60] = 1; inputArc[MT14][j+120] = 1;
outputArc[MS2][j+30] = 1; outputArc[MT16][j+90] = 1; outputArc[MT14][j+150] = 1;
j = 4;  /*For Job 5*/
inputArc[MS1][j] = 1; inputArc[MT10][j+60] = 1; inputArc[MT11][j+120] = 1;
```

```
outputArc[MS1][j+30] = 1; outputArc[MT10][j+90] = 1; outputArc[MT11][j+150] = 1;
j = 5;  /*For Job 6*/
inputArc[MS2][j] = 1; inputArc[MT7][j+60] = 1; inputArc[MT13][j+120] = 1;
outputArc[MS2][j+30] = 1; outputArc[MT7][j+90] = 1; outputArc[MT13][j+150] = 1;
j = 6;  /*For Job 7*/
inputArc[MS1][j] = 1; inputArc[MS2][j+60] = 1; inputArc[MT12][j+120] = 1;
outputArc[MS1][j+30] = 1; outputArc[MS2][j+90] = 1; outputArc[MT12][j+150] = 1;
j = 7;  /*For Job 8*/
inputArc[MS2][j] = 1; inputArc[MT8][j+60] = 1; inputArc[MT10][j+120] = 1;
outputArc[MS2][j+30] = 1; outputArc[MT8][j+90] = 1; outputArc[MT10][j+150] = 1;
j = 8;  /*For Job 9*/
inputArc[MT2][j] = 1; inputArc[MT1][j+60] = 1; inputArc[MT3][j+120] = 1;
outputArc[MT2][j+30] = 1; outputArc[MT1][j+90] = 1; outputArc[MT3][j+150] = 1;
j = 9;  /*For Job 10*/
inputArc[MT1][j] = 1; inputArc[MT3][j+60] = 1; inputArc[MT2][j+120] = 1;
outputArc[MT1][j+30] = 1; outputArc[MT3][j+90] = 1; outputArc[MT2][j+150] = 1;
j = 10;  /*For Job 11*/
inputArc[MT4][j] = 1; inputArc[MT1][j+60] = 1; inputArc[MT5][j+120] = 1;
outputArc[MT4][j+30] = 1; outputArc[MT1][j+90] = 1; outputArc[MT5][j+150] = 1;
j = 11;  /*For Job 12*/
inputArc[MT2][j] = 1; inputArc[MT1][j+60] = 1; inputArc[MT3][j+120] = 1;
outputArc[MT2][j+30] = 1; outputArc[MT1][j+90] = 1; outputArc[MT3][j+150] = 1;
j = 12;  /*For Job 13*/
inputArc[MS1][j] = 1; inputArc[MT4][j+60] = 1; inputArc[MT8][j+120] = 1;
outputArc[MS1][j+30] = 1; outputArc[MT4][j+90] = 1; outputArc[MT8][j+150] = 1;
j = 13;  /*For Job 14*/
inputArc[MT2][j] = 1; inputArc[MT1][j+60] = 1; inputArc[MT4][j+120] = 1;
outputArc[MT2][j+30] = 1; outputArc[MT1][j+90] = 1; outputArc[MT4][j+150] = 1;
j = 14;  /*For Job 15*/
inputArc[MS2][j] = 1; inputArc[MS3][j+60] = 1; inputArc[MS4][j+120] = 1;
outputArc[MS2][j+30] = 1; outputArc[MS3][j+90] = 1; outputArc[MS4][j+150] = 1;
j = 15;  /*For Job 16*/
inputArc[MT4][j] = 1; inputArc[MT3][j+60] = 1; inputArc[MT1][j+120] = 1;
outputArc[MT4][j+30] = 1; outputArc[MT3][j+90] = 1; outputArc[MT1][j+150] = 1;
j = 16;  /*For Job 17*/
inputArc[MT1][j] = 1; inputArc[MT5][j+60] = 1; inputArc[MT4][j+120] = 1;
outputArc[MT1][j+30] = 1; outputArc[MT5][j+90] = 1; outputArc[MT4][j+150] = 1;
j = 17;  /*For Job 18*/
inputArc[MT5][j] = 1; inputArc[MT4][j+60] = 1; inputArc[MT3][j+120] = 1;
outputArc[MT5][j+30] = 1; outputArc[MT4][j+90] = 1; outputArc[MT3][j+150] = 1;
j = 18;  /*For Job 19*/
inputArc[MT9][j] = 1; inputArc[MT8][j+60] = 1; inputArc[MT6][j+120] = 1;
outputArc[MT9][j+30] = 20; outputArc[MT8][j+90] = 1; outputArc[MT6][j+150] = 1;
j = 19;  /*For Job 1*/
inputArc[MT7][j] = 1; inputArc[MT2][j+60] = 1; inputArc[MT4][j+120] = 1;
outputArc[MT7][j+30] = 1; outputArc[MT2][j+90] = 1; outputArc[MT4][j+150] = 1;
j = 20;  /*For Job 21*/
inputArc[MT5][j] = 1; inputArc[MT6][j+60] = 1; inputArc[MT9][j+120] = 1;
outputArc[MT5][j+30] = 1; outputArc[MT6][j+90] = 1; outputArc[MT9][j+150] = 1;
j = 21;  /*For Job 22*/
inputArc[MT4][j] = 1; inputArc[MT7][j+60] = 1; inputArc[MT6][j+120] = 1;
outputArc[MT4][j+30] = 1; outputArc[MT7][j+90] = 1; outputArc[MT6][j+150] = 1;
j = 22;  /*For Job 23*/
inputArc[MT2][j] = 1; inputArc[MT1][j+60] = 1; inputArc[MT5][j+120] = 1;
```

```
outputArc[MT2][j+30] = 1; outputArc[MT1][j+90] = 1; outputArc[MT5][j+150] = 1;
j = 23;  /*For Job 24*/
inputArc[MT3][j] = 1; inputArc[MT4][j+60] = 1; inputArc[MT6][j+120] = 1;
outputArc[MT3][j+30] = 1; outputArc[MT4][j+90] = 1; outputArc[MT6][j+150] = 1;
j = 24;  /*For Job 25*/
inputArc[MT6][j] = 1; inputArc[MT1][j+60] = 1; inputArc[MT5][j+120] = 1;
outputArc[MT6][j+30] = 1; outputArc[MT1][j+90] = 1; outputArc[MT5][j+150] = 1;
j = 25;  /*For Job 26*/
inputArc[MT10][j] = 1; inputArc[MT13][j+60] = 1; inputArc[MT7][j+120] = 1;
outputArc[MT10][j+30] = 1; outputArc[MT13][j+90] = 1; outputArc[MT7][j+150] = 1;
j = 26;  /*For Job 27*/
inputArc[MT15][j] = 1; inputArc[MT16][j+60] = 1; inputArc[MT9][j+120] = 1;
outputArc[MT15][j+30] = 1; outputArc[MT16][j+90] = 1; outputArc[MT9][j+150] = 1;
j = 27;  /*For Job 28*/
inputArc[MT12][j] = 1; inputArc[MT8][j+60] = 1; inputArc[MT5][j+120] = 1;
outputArc[MT12][j+30] = 1; outputArc[MT8][j+90] = 1; outputArc[MT5][j+150] = 1;
j = 28;  /*For Job 29*/
inputArc[MT8][j]= 1; inputArc[MT9][j+60] = 1; inputArc[MT2][j+120] = 1;
outputArc[MT8][j+30] = 1; outputArc[MT9][j+90] = 1; outputArc[MT2][j+150] = 1;
j = 29;  /*For Job 30*/
inputArc[MT13][j] = 1; inputArc[MT11][j+60] = 1; inputArc[MT12][j+120] = 1;
outputArc[MT13][j+30] = 1; outputArc[MT11][j+90] = 1; outputArc[MT12][j+150] = 1;
```

REFERENCES

T. Agerwala and Y. Choed-Amphai, "A synthesis rule for concurrent systems," in *Proceedings of 15th Design Automation Conf.*, Las Vegas, NV, pp. 305-311, 1978.

R. G. Askin and C. R. Standridge, *Modeling and Analysis of Manufacturing Systems*, John Wiley & Sons, Inc., NY, 1993.

F. Baccelli and Z. Liu, "Comparison properties of stochastic decision free Petri nets," *IEEE Trans. on Automatic Control*, vol. 37, no. 12, pp. 1905-1920, 1992.

K. R. Baker, *Introduction to Sequencing and Scheduling*, John Wiley & Sons, New York, 1974.

C. Basnet and J. Mize, "Scheduling and control of flexible manufacturing systems: a critical review," *Int. J. of Computer Integrated Manufacturing*, vol. 7, no. 6, pp. 340-355, 1994.

Z. Banaszak and B. Krogh, "Deadlock avoidance in flexible manufacturing systems with concurrently competing process flows," *IEEE Trans. on Robotics and Automation*, vol. 6, no. 6, pp. 724-734, 1990.

O. Berman and O. Maimon, "Cooperation among flexible manufacturing systems," *IEEE J. of Robotics and Automation*, vol. 2, no. 1, pp. 24-30, 1986.

J. H. Blackstone, D. T. Phillips and G. L. Hogg, "A state-of-the-art survey of dispatching rules for manufacturing job shop operations," *Int. J. of Production Research*, vol. 20, no. 1, pp. 27-45, 1982.

J. Blazewicz, H. Eiselt, G. Finke, G. Laporte and J. Weglarz, "Scheduling tasks and vehicles in a flexible manufacturing system," *Int. J. of Flexible Manufacturing Systems*, 4, pp. 5-16, 1991.

J. Campos, G. Chiola, J. M. Colom and M. Silva, "Properties and performance bounds for timed marked graphs," *IEEE Trans. on Circuits and Systems*, vol. 39, no.5, pp. 386-401, 1992.

J. Carlier and E. Pinson, "An algorithm for solving the job-shop problem," *Management Science*, vol. 35, no. 2, pp. 164-176, 1989.

J. K. Chaar, D. Teichroew and R. A. Volz, "Developing manufacturing control software: a survey and critique," *Int. J. of Flexible Manufacturing Systems*, 5, pp. 53-58, 1993.

O. Charalambous and K. Hindi "A knowledge based job-shop scheduling system with controlled backtracking," *Computer and Industrial Engineering*, vol. 24, no. 3, pp. 391-400, 1993.

T. R. Chen, *Scheduling for IC Sort and Test Facilities via Lagrangian Relaxation*, Ph.D. Dissertation, University of Carlifornia at Davis, CA, 1994.

Q. Chen and J. Y. Luh, "Operational scheduling using truncated Petri net technique," in *Proceedings of IEEE Workshop on Emerging Technologies and Factory Automation*, Melbourne, Australia, pp. 230-235, Aug. 1992.

T. Chen, X. Qi and F. Tu, "A bicriteria scheduling problem with earliness and tardiness penalties," in *Proceedings of the 33nd Conf. on Decision and Control*, Lake Buena Vista, FL, pp. 1577-1582, Dec. 1994.

H. Cho, T. K. Kumaran and R. A. Wysk, "Graph-theoretic deadlock detection and resolution for flexible manufacturing systems," *IEEE Trans. on Robotics and Automation*, vol. 11, no. 3, pp. 413-421, 1995.

L. Custodio, J. Sentieiro and C. Bispo, "Production planning and scheduling using a fuzzy decision system," *IEEE Trans. on Robotics and Automation*, vol. 10, no. 2, pp. 160-168, 1994.

C. S. Czerwinski and P. B. Luh, "Scheduling Products with bills of materials using an improved Lagrangian relaxation technique," *IEEE Trans. on Robotics and Automation*, vol. 10, no. 2, pp. 99-111, 1994.

R. David and H. Alla, *Petri Nets and Grafcet*, Prentice Hall International (UK) Ltd, 1992.

R. David and H. Alla, "Petri nets for modeling of dynamic systems - a survey," *Automatica*, vol. 30, no. 2, pp. 175-202, 1994.

F. DiCesare and A. A. Desrochers, "Modeling, control, and performance analysis of automated manufacturing systems using Petri nets," *Control and Dynamic Systems*, C. T. Leondes (Ed.), vol. 47, pp. 121-172, Academic Press, MA, 1991.

Z. Doulgeri, G. D'alessandro and N. Magaletti "A hierarchical knowledge-based scheduling and control for FMSs," *Int. J. of Computer Integrated Manufacturing*, vol. 6, no. 3, pp. 191-200, 1993.

R. A. Dudek, S. S. Panwalkar and M. L. Smith, "The lessons of flowshop scheduling research," Operations Research, vol. 40, no. 1, pp.86-98, 1992.

O. Dunkler, C. M. Mitchell, T. Govindaraj, and J. C. Ammons, "The effectiveness of supervisory control strategies in scheduling flexible manufacturing systems," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 18, no. 2, pp. 223-237, 1988.

E. Falkenauer and S. Bouffouix, "A genetic algorithm for job shop," in *Proceedings of 1991 IEEE Int. Conf. on Robotics and Automation*, Sacramento, CA, pp. 824-829, April, 1991.

L. Ferrarini, "An incremental approach to logic controller design with Petri nets," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 23, no. 3, pp. 461-473, 1992.

L. Ferrarini, M. Narduzzi and M. Tassan-Solet, "A new approach to modular liveness analysis conceived for large logical controller's design," *IEEE Trans. on Robotics and Automation*, vol. 10, no. 2, pp. 169-184, 1994.

S. France, *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop*. New York: Wiley, NY, 1982.

P. Freedman, "Time, Petri Nets, and Robotics," *IEEE Trans. on Robotics and Automation*, vol. 7, no. 4, pp. 417-433, 1991.

B. Grabot and L. Geneste, "Dispatching rules in scheduling: a fuzzy approach," *Int. J. of Production Research*, vol. 32, no. 4, pp. 903-915, 1994.

I. Hatono, K. Yamagata and H. Tamura, "Modeling and on-line scheduling of flexible manufacturing systems using stochastic Petri nets," *IEEE Trans. on Software Engineering*, vol. 17, no. 2, pp. 126-132, 1991.

H. P. Hillion and J. M. Proth, "Performance evaluation of job-shop systems using timed event-graphs," *IEEE Trans. on Automatic Control*, vol. 34, no. 1, pp. 3-9, 1989.

D. Y. Hsieh and S. C. Chang, "Dispatching-driven deadlock avoidance controller synthesis for flexible manufacturing systems," *IEEE Trans. on Robotics and Automation*, vol. 10, no. 2, pp. 196-209, 1994.

IEC, Technical Committee 65: Industrial Process Measurement and Control, Subcommittee 65A, Working Group 6 (1990). Part 3: *Programming Languages*, March, 1990.

N. Ishii and J. Talavage, "A mixed dispatching rule approach in FMS scheduling," *Int. J. of Flexible Manufacturing Systems*, vol. 6, no. 1, pp. 69-87, 1994.

M. A. Jafari, "Performance modeling of a flexible manufacturing cell with two workstations and a single material handling device," in *Proceedings of 1987 IEEE Int. Conf. on Robotics and Automation*, pp. 866-871, 1987.

M. A. Jafari, "An architecture for a shop-floor controller using colored Petri nets," *Int. J. of Flexible Manufacturing Systems*, vol. 4, no. 1, pp. 159-181, 1992.

M. D. Jeng and F. DiCesare, "A review of synthesis techniques for Petri nets with applications to automated manufacturing systems," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 23, no. 1, pp. 301-312, 1993.

M. N. Karsiti, J. B. Cruz and J. H. Mulligan, "Simulation studies of multilevel dynamic job shop scheduling using heuristic dispatching rules," *J. of Manufacturing Systems*, vol. 11, no. 5, pp. 346-357, 1992.

M. H. Kim and Y. D. Kim, "Simulation-based real-time scheduling in a flexible manufacturing system," *J. of Manufacturing Systems*, vol. 13, no. 2, pp. 85-93, 1994.

J. Kimemia and S. Gershwin, "An algorithm for the computer control of a flexible manufacturing system," *IIE Transactions*, vol. 15, no. 4, pp. 353-362, 1983.

G. J. Klir and T. A. Folger, *Fuzzy Sets, Uncertainty, and Information*, Prentice Hall, NJ, 1991.

I. Koh and F. DiCesare, "Modular transformation methods for generalized Petri nets and their application to automated manufacturing systems," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 21, no. 6, pp. 1512-1521, 1991.

B. H. Krogh and C. L. Beck, "Synthesis of place/transition nets for simulation and control of manufacturng systems," in *Proceedings of IFIP Symposium on Large Systems*, Zurich, pp. 1-6, Aug. 1986.

C. Y. Lee, R. Uzsoy and L. A. Martin-Vega, "Efficient algorithms for scheduling semiconductor burn-in operations," *Operations Reserach*, vol. 40, no. 4., pp. 764-795, 1992.

D. Y. Lee and F. DiCesare, "Scheduling flexible manufacturing systems with the consideration of setup times," in *Proceedings of the 32nd Conference on Decision and Control*, San Antonio, TX, pp.3264-3269, Aug. 1993.

D. Y. Lee and F. DiCesare, "Scheduling FMS using Petri nets and heuristic search," *IEEE Trans. on Robotics and Automation*, vol. 10, no. 2, pp. 123-132, 1994.

D. Y. Lee and F. DiCesare, "Integrated scheduling of flexible manufacturing systems employing automated guided vehicles," *IEEE Trans. on Industrial Electronics*, vol. 41, no. 6, pp. 602-610, 1994.

C. Lee, "Fuzzy logic in control systems: fuzzy logic controller, part I, II" *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 20, no. 2, pp. 404-435, 1990.

V. J. Leon and S. D. Wu, "Characteristics of computerized scheduling and control of manufacturing systems," *Computer Control of Flexible Manufacturing Systems*, S. Joshi and G. Smith (Ed.), Chapman & Hall, UK, pp.63-73, 1994.

J. J. Lesage and J. M. Roussel, "Hierarchical approach to grafcet using forcing order," *RAIRO, Automatic Control Production Systems*, vol. 27, no. 1, pp. 25-38, 1993.

F. L. Lewis, H. H. Huang and S. Jagannathan, "A system approach to discrete event controller design for manufacturing systems control," in *Proceedings of 1993 American Control Conf.*, San Francisco, CA, pp. 1525-1531, 1993.

S. Li, T. Takamori and S. Tadokoro, "Scheduling and re-scheduling of AGVs for flexible and agile manufacturing," *Petri Nets in Flexible and Agile Automation*, M. C. Zhou (Ed.), pp. 189-205, Kluwer Academic Publications, Boston, MA, 1995.

J. Lin and D. Ionescu, "Optimization of controller design for discrete event systems in a temporal logic framework," in *Proceedings of the 1992 American Control Conference*, Chicago, IL, pp. 2819-2823, June 1992.

P. B. Luh and D. J. Hoitomt, "Scheduling of manufacturing systems using the Lagrangian relaxation technique," *IEEE Trans. on Automatic Control*, vol. 38, no. 7, 1993, pp. 1066-1079, 1993.

O. Z. Maimon and S. B. Gershwin, "Dynamic scheduling and routing for flexible manufacturing systems that have unreliable machines," *Operations Research*, vol. 36, no. 2, pp. 279-292, 1988.

M. Montazeri and L. N. Van Wassenhove, "Analysis of scheduling rules for an FMS," *Int. J. of Production Research*, vol. 28, no. 4, pp. 785-802, 1990.

S. Morioka and T. Yamada, "Performance evaluation of marked graph by linear programming," *Int. J. of Systems Science*, vol. 22, no. 9, pp. 1541-1552, 1991.

T. Murata, "Petri nets: properties, analysis and applications," *Proceedings of The IEEE*, vol. 77, no. 4, pp. 541-579, 1989.

T. Murata, N. Komoda, K. Matsumoto and K. Haruna, "A Petri Net-Based Controller for Flexible and Maintainable Sequence Control and Its Applications in Factory Automation," *IEEE Trans. on Industrial Electronics*, vol. 33, no. 1, pp. 1-8, 1986.

Y. Narahari and N. Viswanadham, "A Petri net approach to the modeling and analysis of flexible manufacturing systems," *Ann. Operations Resaerch*, vol. 3, pp. 449-472, 1985.

N. Nilsson, *Principles of Artificial Intelligence*, Palo Alto, CA, 1980.

K. M. Passino and P. J. Antsaklis, "On the optimal control of discrete event systems," in *Proceedings of the 28th Conf. on Decision and Control*, Tampa, Florida, pp. 2713-2718, Dec. 1989.

J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Reading, MA,: Addison-wesley, 1984.

J. L. Peterson, *Petri Net Theory and Modeling of Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1981.

J. M. Proth and I. Minis, "Planning and scheduling based on Petri nets," *Petri Nets in Flexible and Agile Automation*, M. C. Zhou (Ed.), pp. 109-148, Kluwer Academic Publications, Boston, MA, 1995.

C. V. Ramamoorthy and G. S. Ho, "Performance evaluation of asynchronous concurrent systems using Petri nets," *IEEE Trans. on Software Engineering*, vol. 6, no. 5, pp. 440-449, 1980.

S. E. Ramaswamy and S. B. Joshi, "Deadlock-free schedules for automated manufacturing workstations," *IEEE Trans. on Robotics and Automation*, vol. 12, no. 3, pp. 391-400, 1996.

C. Ramchandani, *Analysis of Asynchronous Concurrent Systems by Timed Petri Nets*, Ph.D. Disseration, MIT, MA, September 1973.

F. Rodammer and J. K. White, "A recent survey of production scheduling," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 18, no. 6, pp. 841-851, 1988.

R. V. Rogers and K. P. White, "Algebraic, mathematical programming, and network models of the deterministic job-shop scheduling problem," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 21, no. 3, pp. 693-697, 1991.

B. Sayat and P. Ladet, "Control specification of a production system using Grafcet and Petri nets," *RAIRO, Automatic Control Production Systems*, vol. 27, no. 1, pp. 53-64, 1993.

T. Sen and S. K. Gupta, "A branch-and-bound procedure to solve a bicriterion scheduling problem," *IIE Trans*, vol. 15, no. 1, pp. 84-87, 1983.

R. Sengupta and S. Lafortune, "Optimal control of a class of discrete event systems," *IFAC Symposium on Distributed Intelligence Systems*, Arlington, VA, pp. 25-30, Aug. 1991.

M. J. Shaw, "Knowledge-based scheduling in flexible manufacturing systems: An integration of pattern-directed inference and heuristic search," *Int. J. of Production Research*, vol. 26, no. 5, pp. 821-844, 1988.

L. Shen, Q. Chen and J. Luh, "Truncation of Petri net models for simplifying computation of optimum scheduling problems," *Computers in Industry*, 20, pp. 25-43, 1992.

H. Shih and T. Sekiguchi, "A timed Petri net and beam search based on-line FMS scheduling system with routing flexibility," in *Proceedings of the 1991 IEEE Int. Conf. on Robotics and Automation*, Sacramento, CA, pp. 2548-2553, April 1991.

T. Sun, C. Cheng and L. Fu, "A Petri net based approach to modeling and scheduling for an FMS and a case Study," *IEEE Trans. on Industrial Electronics*, vol. 41, no. 6, pp. 593-601, 1994.

T. Sawik, "Modeling and scheduling of a flexible manufacturing system," *European J. of Operations Research*, vol. 45, no. 1, pp. 177-190, 1990.

Y. Tsukamoto, "An approach to fuzzy reasoning method," in *Advances in Fuzzy Set Theory and Applications*, M. M. Gupata, R. K. Ragade, and R. R. Yager, Eds., Amsterdam: North-Holland, 1979.

R. Uzsoy, C. Y. Lee and L. A. Martin-Vega, "A review of production planning and scheduling models in the semiconductor industry part I: system characteristics, performance evaluation and production planning," *IIE Trans. on Scheduling and Logistics*, vol. 24, no. 4, pp. 47-60, 1992.

R. Uzsoy, C. Y. Lee and L. A. Martin-Vega, "A review of production planning and scheduling models in the semiconductor industry part II: shop-floor control," *IIE Trans. on Scheduling and Logistics*, vol. 26, no. 5, pp. 44-55, 1994.

R. Uzsoy, L. A. Martin-Vega, C. Y. Lee and P. A. Leonard, "Production scheduling algorithms for a semiconductor test facility," *IEEE Trans. on Semiconductor Manufacturing*, vol. 4, pp. 271-280, 1991.

R. Valette, "Analysis of Petri nets by stepwise refinements," *J. of Computation and System Science*, vol. 18, pp. 35-46, 1979.

P. Van Laarhoven, E. Aarts and J. K. Lenstra, "Job-shop scheduling by simulated annealing," *Operations Research*, vol. 40, no. 1, pp. 113-125, 1992.

K. Venkatesh, M. C. Zhou and R. J. Caudill, R, "Comparing ladder logic diagrams and Petri nets for sequence controller design through a discrete manufacturing system," *IEEE Trans. on Industrial Electronics*, vol. 41, no. 6, pp. 611-619, 1994.

N. Viswanadham, Y. Narahari and T. Johnson, "Deadlock prevention and deadlock avoidance in flexible manufacturing systems using Petri net models," *IEEE Trans. on Robotics and Automation*, vol. 6, no. 6, pp. 713-723, 1990.

T. Watanabe, H. Tokumaru, Y. Nakajima and Y. Hashimoto, "Job-shop scheduling using fuzzy inference to take profit into account," in *Proceedings of Japan-U.S.A. Syposium on Flexible Automation*, San Franciso, CA, pp. 423-427, July 1992.

R. G. Willson and B. H. Krogh, "Petri net tools for the specification and analysis of discrete controllers," *IEEE Trans. on Software Engineering*, vol. 16, no. 1, pp. 39-50, 1990.

S. Wu and R. A. Wysk, "Multi-pass expert control system - a control/scheduling structure for flexible manufacturing cells," *J. of Manufacturing Systems*, 7, pp. 107-120, 1988.

S. Wu and R. A. Wysk, "An application of discrete-event simulation to on-line control and scheduling in flexible manufacturing," *Int. J. Of Production Resaerch*, 27, pp. 1603-1623, 1989.

R. A. Wysk, N. Yang and S. Joshi, "Resolution of deadlocks in flexible manufacturing systems: avoidance and recovery approaches," *Journal of Manufacturing Systems*, vol. 13, no. 2, pp. 128-138, 1994.

X. L. Xie, "Superposition properties and performance bounds of stochastic timed-event graphs," *IEEE Trans. on Automatic Control*, vol. 39, no. 7, pp.1376-1386, 1994.

K. Xing, B. Hu and H. Chen, "Deadlock avoidance policy for flexible manufacturing systems," *Petri Nets in Flexible and Agile Automation*, M. C. Zhou (Ed.), pp. 239-263, Kluwer Academic Publications, Boston, MA, 1995.

H. H. Xiong and M. C. Zhou, "Computer communication networks for automated manufacturing," in *Proceedings of 1994 Int. Conf. on Electronics and Information Technology*, Beijing, China, pp. 178-183, Aug. 1994.

H. H. Xiong, M. C. Zhou and R. J. Caudill, "A hybrid heuristic search algorithm for scheduling flexible manufacturing systems," in *Proceedings of 1996 IEEE Int. Conf. on Robotics and Automation*, Minneapolis, MN, pp. 2793-2797, Apr. 1996.

H. H. Xiong, M. C. Zhou and R. J. Caudill, "Design of optimal sequence controller for a flexible manufacturing system," to appear in *Proceedings of 1996 IEEE Int. Conf. on Systems, Man, and Cybernetics*, Beijing, China, Oct. 1996.

H. H. Xiong, M. C. Zhou and C. N. Manikopoulos, "Modeling and performance analysis of medical service systems using Petri nets," in *Proceedings of 1994 IEEE Int. Conf. on Systems, Man, and Cybernetics*, San Antonio, TX, pp. 2339-2342, Oct. 1994.

H. H. Xiong, M. C. Zhou and C. N. Manikopoulos, "Scheduling flexible manufacturing systems based on timed Petri nets and fuzzy dispatching rules," in *Proceedings of 1995 IEEE Symposium on Emerging Technologies and Factory Automation*, Paris, France, pp. 309-315, Oct. 1995.

D. Zhang, "Planning using timed Pr/T Nets," in *Proceedings of Japan-U.S.A. Symp. on Flexible Automation*, San Francisco, CA, pp. 1179-1184, July 1992.

D. N. Zhou, V. Cherkassky, T. R. Baldwin and D. E. Olson, "A neural network approach to job-shop scheduling," *IEEE Trans. on Neural Networks*, vol. 2, no. 1, pp. 175-179, 1991.

M. C. Zhou, H. Chiu and H. H. Xiong, "Petri net scheduling of FMS using branch and bound method," *Proc. of 1995 IEEE Int. Conf. on Industrial Electronics, Control, and Instrumentation*, Orlando, FL, pp. 211-216, Nov. 1995.

M. C. Zhou and F. DiCesare, "Adaptive design of Petri net controllers for error recovery in automated manufacturing systems," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 19, no. 5, pp. 963-973, 1989.

M. C. Zhou and F. DiCesare, "Parallel and sequential mutual exclusions for Petri net modeling of manufacturing systems with shared resources," *IEEE Trans. on Robotics and Automation*, vol. 7, no. 4, pp. 515-527, 1991.

M. C. Zhou and F. DiCesare, *Petri Net Synthesis for Discrete Event Control of Manufacturing Systems*, Kluwer Academic Publications, Boston, MA, 1993.

M. C. Zhou, F. DiCesare and A. Desrochers, "A hybrid methodology for synthesis of Petri nets for manufacturing systems," *IEEE Trans. on Robotics and Automation*, vol. 8, no. 3, pp. 350-361, 1992.

M. C. Zhou, F. DiCesare and D. Rudolph, "Design and implementation of a Petri net supervisor for a flexible manufacturing systems," *Automatica*, vol. 28, no. 6, pp. 1999-2008, 1992.

M. C. Zhou, K. McDermott and P. A. Patel, "Petri net synthesis and analysis of an FMS cell," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 23, no. 2, pp. 523-531, 1993.

M. C. Zhou, H. H. Xiong and C. N. Manikopoulos, "Performance models for communication networks in manufacturing environment," in *Proceedings of the Fourth Int. Conf. on Computer Integrated Manufacturing and Automation Technology*, Troy, NY, pp. 417-422, Oct. 1994.

R. Zurawski, "Systematic construction of functional abstractions of Petri net models of flexible manufacturing systems," *IEEE Trans. on Industrial Electronics*, vol. 41, no. 6, pp. 584-592, 1994.

R. Zurawski and M. C. Zhou, "Petri nets and industrial applications: a tutorial," *IEEE Trans. on Industrial Electronics*, vol. 41, no. 6, pp. 567-583, 1994.