

**ANALYSIS OF REACTIVE SCHEDULING PROBLEMS
IN MANUFACTURING SYSTEMS**

**A THESIS
SUBMITTED TO THE DEPARTMENT OF INDUSTRIAL ENGINEERING
AND THE INSTITUTE OF ENGINEERING AND SCIENCES
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE**

By

Murat Bayız

July, 1997

*TS
157.5
.B39
1997*

ANALYSIS OF REACTIVE SCHEDULING PROBLEMS
IN MANUFACTURING SYSTEMS

A THESIS

SUBMITTED TO THE DEPARTMENT OF INDUSTRIAL ENGINEERING
AND THE INSTITUTE OF ENGINEERING AND SCIENCES

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

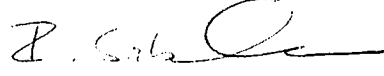
FOR THE DEGREE OF
MASTER OF SCIENCE

By Murat Bayız.
Sanıřlından bağıřlanmıştır
Murat Bayız
July, 1997

TS
157.5
.B39
1937

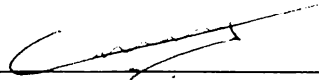
BC38250

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.



Assoc. Prof. İhsan Sabuncuoğlu(Principal Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.



Assoc. Prof. Cemal Dinçer

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.



Assoc. Prof. Nur Evin Özdemirel

Approved for the Institute of Engineering and Sciences:



Prof. Mehmet Baray
Director of Institute of Engineering and Sciences

ABSTRACT

ANALYSIS OF REACTIVE SCHEDULING PROBLEMS IN MANUFACTURING SYSTEMS

Murat Bayız

M.S. in Industrial Engineering

Supervisor: Assoc. Prof. İhsan Sabuncuoğlu

July, 1997

In this study we develop a new scheduling algorithm for the job shop problem. The proposed algorithm is a heuristic method based on the filtered beam search. After extensive analyses on the evaluation functions and search parameters of the beam search, we measure the performance of the algorithm in terms of quality of solutions and CPU times for both the makespan and mean tardiness criteria.

In the second half of the research, we study the reactive scheduling problem. Specifically, we analyze several reactive methods such as no response, periodic response and continuous response under various experimental conditions. The beam search based partial scheduling is also studied in this thesis. The method is analyzed for both deterministic and stochastic environments under several job shop configurations.

Key words: Reactive Scheduling, Beam Search, Job Shop Scheduling

ÖZET

ÜRETİM SİSTEMLERİNDEKİ TEPKİSEL ÇİZELGELEME PROBLEMLERİNİN ANALİZİ

Murat Bayız

Endüstri Mühendisliği Bölümü Yüksek Lisans

Tez Yöneticisi: Doç. İhsan Sabuncuoğlu

Temmuz, 1997

Bu çalışmada atelye tipi üretim sistemleri için yeni bir çizelgeleme algoritması geliştirilmiştir. Bu algoritma süzülmiş ışın taramasına dayalı sezgisel bir yöntemdir. Değerlendirme işlevleri ve tarama parametreleri üzerine yapılan kapsamlı çözümlerden sonra, algoritmanın çözüm kalitesi ve zamanı açılarından başarımı “tüm bitirim süresi” ve “ortalama gecikme” ölçütlerinde ölçülmüştür.

Bu çalışmanın ikinci bölümünde tepkisel çizelgeleme problemi incelenmiştir. Özellikle tepki olmayan, dönemsel tepki ve sürekli tepki gibi yöntemler değişik deney koşulları altında çözümlenmiştir. Bu tezde ışın taramasına dayalı kısmi çizelgeleme yöntemi de araştırılmıştır. Bu yöntem gerekirci ve rassal ortamlarda değişik atelye tipi üretim biçimlerinde çözümlenmiştir.

Anahtar sözcükler: Tepkisel çizelgeleme, Işın taraması, Atelye tipi üretim sistemi çizelgelemesi

To my parents

ACKNOWLEDGEMENT

I would like to express my gratitude to Assoc. Prof. İhsan Sabuncuođlu due to his supervision, suggestions, and understanding throughout the development of his thesis.

I am also indebted to Cemal Dinđer and Nur Evin Özdemirel for showing keen interest to the subject matter and accepting to read and review this thesis.

I cannot fully express my gratitude, love and thanks to Şelale Tüzel for her morale support and encouragement.

I would also like to thank to Alev Kaya, Souheyl Touhami, Fatma Gzara, Feryal Erhun, Hülya Emir, Serkan Özkan, Kemal Kılıç for their help during the preparation of this thesis.

My special thanks also go to my parents, brothers, sisters in law and dear nephews Yağız Efe and Mertcan for their morale support throughout my studies.

And, finally I would like to thank to all my graduate friends for their friendship.

Contents

1	INTRODUCTION	1
2	LITERATURE REVIEW	4
2.1	Observations	14
2.2	Open Research Points and Motivation	17
3	BEAM SEARCH	19
3.1	Introduction	19
3.2	Problem Definition	21
3.3	Beam Search	22
3.3.1	The Proposed Beam Search Based Algorithm	23
3.4	Makespan case	32
3.4.1	Computational results	37
3.5	Mean tardiness case	41
3.5.1	Computational Results	46
3.6	Conclusion	48

<i>CONTENTS</i>	ix
4 REACTIVE SCHEDULING	50
4.1 Introduction	50
4.2 The Proposed Study	52
4.2.1 Scheduling Methods	52
4.2.2 Job Shop Environment	53
4.2.3 Machine Breakdowns	55
4.2.4 Frequency of Scheduling	56
4.3 Computational Results	58
4.3.1 Deterministic Environment	58
4.3.2 Stochastic Environment	62
4.3.3 Partial Scheduling	71
4.3.4 Partial Scheduling in Deterministic Environment	83
4.4 Conclusion	89
5 CONCLUSION	92
A Beam Search	102
B Reactive Scheduling	106

List of Figures

3.1	Representation of beam search tree	24
3.2	Beam search tree for the numerical example	29
3.3	Schedules generated by the beam search algorithm	32
3.4	Percent deviation from optimal solution vs filterwidth	35
3.5	CPU time vs filterwidth	38
3.6	Mean tardiness vs filterwidth analysis	43
3.7	Mean tardiness vs filterwidth analysis	44
4.1	Interactions between scheduling frequency and mean tardiness	70
4.2	CPU time vs scheduling frequency	71
4.3	Interactions between scheduling frequency and makespan values	72
4.4	Mean tardiness & CPU time as a function of partial schedule lengths (Uniform Case)	76
4.5	Mean tardiness & CPU time as a function of partial schedule lengths (Nonuniform Case)	77
4.6	Changes in mean tardiness as a function partial schedule length in uniform and nonuniform environments	79

4.7	Interactions between makespan & CPU time and partial schedule length (Uniform Case)	80
4.8	Interactions between makespan & CPU time and partial schedule length (Nonuniform Case)	81
4.9	Mean Tardiness vs partial schedule length in 90% and 80% efficiency levels, Uniform Case	83
4.10	Makespan vs partial schedule length in 90% and 80% efficiency levels, Uniform Case	84
4.11	Interactions between mean tardiness & CPU time and partial schedule length in deterministic environment	87
4.12	Interactions between makespan & CPU time and partial schedule length in deterministic environment	88

List of Tables

2.1	Classification of the papers	16
3.1	Job information	27
3.2	Descriptions of priority rules used in makespan analysis	33
3.3	Searching Methods used in the Makespan Case	34
3.4	Results of test problems for the Makespan Analysis	40
3.5	Comparison of scheduling algorithms	41
3.6	Description of priority used in tardiness analysis	42
3.7	Search methods used in the first part of the experiments	45
3.8	Search methods used in the second part of the experiments	46
3.9	Results of test problems for the Mean Tardiness Analysis	47
4.1	Sizes of the shop analyzed	54
4.2	Performances of algorithms in deterministic environment.	59
4.3	Performances of algorithms in deterministic environment.	62
4.4	Performances of algorithms in stochastic environment	65

4.5 Performances of algorithms in stochastic environment. 66

4.6 Performances of algorithms in stochastic environment 67

4.7 Performances of algorithms in stochastic environment. 68

A.1 Percent deviation from optimal solution vs filterwidth 103

A.2 Mean tardiness vs filterwidth analysis 104

A.3 Mean tardiness vs filterwidth analysis 105

B.1 Interactions between mean tardiness and scheduling frequency 107

B.2 Interactions between mean tardiness and scheduling frequency 107

B.3 Interactions between CPU time and scheduling frequency 108

B.4 Interactions between makespan and scheduling frequency 109

B.5 Interactions between makespan and scheduling frequency 109

B.6 Interactions between mean tardiness & CPU time and partial schedule
length 110

B.7 Interactions between mean tardiness & CPU time and partial schedule
length 110

B.8 Interactions between mean tardiness & CPU time and partial schedule
length 111

B.9 Interactions between mean tardiness & CPU time and partial schedule
length 111

B.10 Interactions between makespan & CPU time and partial schedule length 112

B.11 Interactions between makespan & CPU time and partial schedule length 112

B.12 Interactions between makespan & CPU time and partial schedule length	113
B.13 Interactions between makespan & CPU time and partial schedule length	113
B.14 Mean tardiness vs partial schedule length in 90% and 80% efficiency levels (Uniform case)	114
B.15 Makespan vs partial schedule length in 90% and 80% efficiency levels (Uniform case)	114
B.16 Interactions between mean tardiness & CPU time and partial schedule length in deterministic environment	115
B.17 Interactions between mean tardiness & CPU time and partial schedule length in deterministic environment	115
B.18 Interactions between mean tardiness & CPU time and partial schedule length in deterministic environment	116
B.19 Interactions between mean tardiness & CPU time and partial schedule length in deterministic environment	116
B.20 Interactions between makespan & CPU time and partial schedule length in deterministic environment	117
B.21 Interactions between makespan & CPU time and partial schedule length in deterministic environment	117
B.22 Interactions between makespan & CPU time and partial schedule length in deterministic environment	118
B.23 Interactions between makespan & CPU time and partial schedule length in deterministic environment	118

Chapter 1

INTRODUCTION

Scheduling is an important part of production system planning. Because, the schedule serves as an overall plan on which many other shop activities are based. In addition, scheduling serves as a mechanism which improves the performance of the manufacturing facility. By properly planning the timing of shop floor activities, performance criteria such as makespan, flow time or mean tardiness can be optimized.

The majority of the published literature on the scheduling problem deals with the task of schedule generation. But, scheduling should be undertaken in two stages: predictive and then reactive. In the first stage a schedule is generated and reactive control is applied on the schedule in the second stage to deal with the unexpected events occurring in the system. Even though reactive control is viewed to be very important for successful implementation of a scheduling system, it has not been extensively studied in the literature.

In the first chapter of this thesis, a review of the related research is provided. At the end of the chapter, the papers are classified according to their problem environments, schedule generation methods and reactive control implementations. As a result of the literature review we determine the open research points in the literature which draws the outline of the subsequent studies throughout the thesis.

To achieve the goal of coordinated planning of system activities in real life, a schedule is prepared in the first stage of the shop floor planning. For that reason, we first develop a schedule generation method in this thesis. Since the job shop system represents a general production setting, it is considered as the scheduling environment in this study. The proposed schedule generation method for the job shop problem bases on the filtered beam search which is an heuristic adaptation of the branch and bound algorithm. In the beam search, as different from branch and bound algorithm, at any level only promising nodes are kept for further branching and the remaining nodes are pruned off permanently.

Even though the beam search has been used to solve a wide variety of optimization problems, its performance is not generally known for the scheduling problems. Because in the existing research work, it is primarily applied to the FMS scheduling problems and compared with some dispatching rules. Hence, its relative performance with respect to the known optimum solution and other new heuristics are not known. Besides, it has not been thoroughly studied as a problem solving strategy with several evaluation functions and search parameters.

In the third chapter of the thesis, we attempt to achieve some of these objectives. First of all, we investigate the effectiveness of various rules as local and global evaluation functions of the beam search applications for both the makespan and the mean tardiness criteria. The previous research indicates that the values of filter and beam width effect the performance of the algorithm. Hence, we also examine the performance of the beam search for various values of filter and beam width and determine their proper values. Furthermore, we test two well known schedule generation schemes (active and nondelay schedule generation schemes) in conjunction to beam search applications to the job shop problems. Finally, after determining proper setting of evaluation functions and parameters for the selected problem environment, we measure the performance of the beam search with respect to optimum solutions and compare it with other well known algorithms for the makespan objective function. This chapter extends the related literature as being the first extensive study of the beam search method in the job shop problem.

The major drawback of precomputed schedule is that, after they are released

for execution, due to the random nature of shop floor conditions, the performance of the schedule degrade. Thus, it is desirable to response the unexpected changes to improve the performance of the schedule. The nature of response depends on the way that schedules are generated or scheduling decisions are made. There are mainly two types of schedule generation methods (*off-line* and *on-line* scheduling) and their way of responding to unexpected events are different. In this study, we analyze reactive scheduling policies of these scheduling techniques in a stochastic environment. In order to perform the analysis, the heuristic algorithm based on the filtered beam search and priority dispatching rules are used as the off-line and on-line schedule generation methods and compared under various experimental conditions.

There are several factors in the real life manufacturing environment, that affect the performance of these scheduling methods. Among these factors system size and load allocation and their effects on the relative performance of the scheduling methods are not throughly analyzed. In the fourth chapter of the thesis, we first analyze the effects of these factors in the deterministic environment. Then we allow machine breakdowns in the system and analyze performances of no response and periodic response reactive scheduling policies. In this analysis, we again examine the effects of system size and load allocation in the simulation experiments. In the next part of the thesis, by utilizing the constructive type of the algorithm we propose a partial scheduling method in the context of periodic response policy. According to this response policy, instead of generating complete schedules, partial ones are generated at the beginning of each period. In this part we evaluate the solution quality and CPU time requirements for various partial schedule lengths again under the same experimental conditions. Finally, the notion of partial scheduling is used in the deterministic environment so as to analyze a time based decomposition of the static job shop scheduling problem. In this chapter, all the analyses are performed for both the mean tardiness and makespan criteria.

The thesis ends with the conclusion chapter in which the results of the analyses are summarized and open research directions are pointed out.

Chapter 2

LITERATURE REVIEW

The majority of the published literature on the scheduling problem deals with the task of schedule generation. But as Szelke and Kerr (1994) state scheduling should be undertaken in two stages: predictive and then reactive. Thus reactive control is also important for successful implementation of scheduling decisions. However, in the literature only a few studies have considered reactive scheduling and control as a part of scheduling problem.

Holloway and Nelson (1974) develop a multi-pass procedure for job shop scheduling problem. This procedure attempts to minimize the extend of the precedence violations in the schedule while treating the machine capacities and due dates as fixed constrains. Their heuristic procedure generates delay schedules by using expected processing times. But they test performance of the heuristic in static job shop with random process times. In this study they compare performance of this multi-pass heuristic with some well known dispatching rules and conclude that depending on the problem and criterion, relatively high performance may result even when actual process times are highly variable. In this study, the authors generate an off-line schedule at the beginning of the scheduling horizon and do not make any reactive control. In that sense this paper does not tackle reactive scheduling but, as being the first study which deals with the process time variation in job shop scheduling problem, it is important in the literature. Later they implement this procedure in dynamic job shop environment (1977). They generate schedules

periodically by two implementation methods. The first method confines attention to operations that are available and does not permit enforced idle time in the schedule. The second one is more involved and integrated the scheduling of old jobs and the jobs arrive after review times. The principle conclusion of this research is that periodic generation of schedules, implemented by using second method, merits consideration as an effective scheduling procedure in dynamic job shop environment.

Yamamoto and Nof (1985) study schedule revisions under unexpected machine breakdowns in static environment. They propose a three-phase scheduling/rescheduling scheme: (1) *Planning Phase*: part-mix assignments are made, initial schedule and machine loading table are generated. (2) *Control Phase*: Processes are controlled according to machine loading order instructions. (3) *Rescheduling Phase*: Rescheduling, revising loading table and resorting to control phase functions are performed. They use a scheduling algorithm which is based on branch and bound and generates active schedules. The same scheduling algorithm is used in the rescheduling phase as in the planning phase. In their proposed method, rescheduling phase is called whenever a machine breakdown occurs. The experiments show that scheduling/rescheduling method outperforms the other two alternatives, and its performance is even better in relatively more complex environments. Therefore, the authors point out that the schedule must be revised at some points in time when the current progress of the system deviates from the schedule over a specified limit. But as they state, determination of this limit and frequency of scheduling is important research issues that needs further investigation.

Farn and Muhlemann (1979) study the performance of some heuristics in both static and then dynamic single machine scheduling problem with sequence dependent changeover times. They come up with the observations that the heuristic for the static problem is not necessarily the best heuristic in the dynamic situation. They periodically generate new schedules in dynamic environment. They also observe that the performance of heuristics deteriorates as the length of scheduling period increases. Later Muhlemann, et al. (1982) investigate the dynamic and uncertainty aspects of job shop problem. Process time variation and machine breakdown are considered as causes of uncertainty in the problem. They propose periodic scheduling scheme to deal with dynamically arriving jobs. At each scheduling point, a static

schedule for the available jobs is generated by a dispatching rule. They examine the performance of the shop with respect to various measures of performance, using several dispatch rules under a range of different frequencies of scheduling. The performance of the rules are evaluated for four different scheduling conditions resulted of low and high combinations of uncertainty and workload levels. As anticipated, performance generally deteriorates when scheduling period increases, because frequent revision of schedules make the schedule more up-to-date and the schedule follows the system dynamics at right times. The authors also observe that the performance of the rules worsens as the level of uncertainty and the work load level increases. The experiments show that SPT rule is the best in the overall performance when rescheduling period is less frequent but for more frequent rescheduling truncated SPT and a composite dispatch rule is found superior.

Ovacik and Uzsoy (1994) study rolling horizon heuristics to minimize maximum lateness on a single machine in the presence of sequence dependent set-up times. They investigate this simple problem because it occurs as a subproblem of more complex job shop scheduling problems. The proposed rolling horizon procedures (RHP) address the dynamic scheduling problem by solving a series of smaller subproblems optimally by means of branch and bound method. The size and the number of the subproblems is determined by algorithm parameters, such as maximum size of the subproblems and forecast window length. At each decision point, a subproblem is solved which yields decisions for a certain time period in the future. Only the decisions related to the current decision point are implemented and then they are revised at the next decision point. To evaluate the performance of RHP, they use two different dispatching rules, EDD and EDD combined with local search, as benchmarks. With the appropriate parameter settings, RHP outperforms the dispatch rules. They also point out that the parameters used in the procedure can be appropriately set so that solution quality and CPU times would be traded off according to the application at hand.

Church and Uzsoy (1992) investigate the problem of rescheduling of single machine production system with dynamic job arrivals. In the first part of the paper, the performance of periodic procedure is evaluated both analytically and computationally. Their analytical results formalize the worst case behavior of

periodic rescheduling policies. In periodic rescheduling, schedules are generated at regular intervals and implemented on a rolling horizon basis. At each rescheduling point, static schedules are generated for available jobs by using the EDD rule. Experiments show that in general the performance of periodic rescheduling policies deteriorates as the rescheduling period length increases. But when system utilization is either very low or very high, or due dates are very tight, the length of the scheduling period does not effect the solution quality. Based on the insight obtained through this analysis, in the second part of the paper the authors propose an event driven scheduling procedure to improve the performance without excessive rescheduling. In this procedure, in addition to periodic rescheduling, arrival of a job with a tight due date also causes a need for rescheduling of the system. Their experiments show that the benefits of extra scheduling diminishes rapidly, demonstrating that a well designed event-driven scheduling policy can achieve excellent performance with less computational burden.

Kiran et al. (1991) study a feed-back based heuristic for scheduling manufacturing systems to evaluate tardiness related performance measure. Their proposed heuristic, given an initial schedule, assigns a priority index to each job based on the weighted sum of previous priority index and tardiness status of jobs. Then it generates a new schedule according to the current priority index until finding a schedule with an acceptable performance. In the first part of the paper, they compare the performance of the algorithm with many dispatching rules in a static job shop environment and conclude that their heuristic outperforms the other rules. In the second part they work on the dynamic job shop problem where a new schedule is generated every morning with the daily list of new and existing jobs. After some pilot experiments, they decide to compare performance of their heuristic with COVERT rule for again some tardiness related performance measures. In general, their experiments show that the feed-back heuristic performs well for a variety of tardiness related criteria and the computational burden of the algorithm is reasonable.

Bean et al. (1991) consider the rescheduling of the shop with multiple resources when unexpected events prevent the use of a preplanned schedule. The scheduling strategy discussed in this paper assumes that a preschedule has been constructed

and this preschedule is followed until a disruption occurs. Then they reschedule to match up with the preschedule at some point in the future. In order to match up, they first resequence all jobs on the disrupted machine. If resequencing results excessive tardiness costs, then the procedure proceeds to the multimachine lot assignment rule to redistribute lot-to-machine assignments. This match up approach is compared with the following preplanned schedule without rescheduling, several dynamic dispatching rules and total reschedule approach which does not seek to match up with the preschedule. The results of test problems indicate that the proposed approach is more advantageous.

This match-up approach is applied to a modified flow shop by Akturk and Gorgulu (1993). The approach is also based on revising the preschedule after a machine breakdown occurs in the system. A new schedule is generated so that the state of the system match up the preschedule some time after disruption. In order to do that they propose a reactive hierarchical scheduling strategy which first selects a match-up point for each machine, then reschedules the specified set of jobs. They decompose the rescheduling problem into three parts that are the scheduling of down machine, scheduling of machines in the upward direction of the down machine and scheduling of the ones in the downward direction of down machine, according to the sequence of machines in the modified flow line. If the resulting schedule is not feasible, then the match-up point is changed to enlarge the set of jobs that are rescheduled. The proposed approach is compared with several match-up alternatives, the static pushback strategy and complete rescheduling rules under different experimental settings. Experimental studies show that the the proposed approach is superior in terms of schedule quality and stability as well as computation time.

Nof and Grant (1991) develop an adaptive/predictive scheduling and control system that includes five main functions: scheduler, monitor, comparator, resolver and recovery adopter. Two sets of experiments are performed to analyze the feasibility and effectiveness of the system. In the first experiment set, a manufacturing cell with three machines is considered and the source of disruption is model as process time variation. The system is monitored at the time of order completions. If the observed performance deviates form the specified tolerance fence

(10% of expected value of measure) then time shifting of the current schedule is applied and if the deviation is beyond 20% then adaptation by resequencing of uncompleted orders is applied. The experiments show that the proposed system is feasible as well as renders better production control than no recovery policy. In the second set of experiments, a similar environment is simulated but in this case machine breakdown and unexpected job arrival is taken as the source of disruption. Periodic monitoring is performed and in response to such disruptions, rerouting to alternative machine, order splitting and rescheduling recovery procedures is activated. In the experiments only one of type of recovery procedure is applied in order to observe its separate effects. After analyzing the second set of experiments, the authors conclude that resequencing procedure gives better performance than the other recovery policies. Also they mention that even a relatively weak recovery procedure yields better performance than no recovery at all.

Simulation based approaches are also widely used in complex scheduling problems. By means of simulation models, various options for control action available at each state are simulated and best option is chosen for execution. For example, Matsuura et al. (1993) investigate the problem of selection between sequencing and dispatching as a scheduling approach in job shop environment involving machine breakdowns, specification changes and rush jobs. In the sequencing approach, an initial schedule (sequence) is generated by branch and bound for the initial set of jobs and the resulting sequence is maintained regardless of the unexpected events occurring in the system. Schedules are generated by either FCFS or SPT rules in dispatching approach. In the first part of paper, the problem of which approach is more efficient under various manufacturing environments is studied. Series of simulation experiments was conducted to determine how sequencing and dispatching approaches affect the performance measure of manufacturing environment. The simulation results show that while the manufacturing situation remains similar to the original one, it is better to use the sequencing approach. However, if departure from the original situation is significant, it would be better to use the dispatching approach. In the second part of the paper, they propose a new approach that switches from sequencing to dispatching when the first unexpected event occurs to make the best use of sequencing and dispatching

approaches. Experiments show that this combined method outperforms other two approaches.

In another study, Kim and Kim (1994) investigate a simulation based real time scheduling methodology for a flexible manufacturing system. In the methodology there are two major components, a simulation module and a real time control system. The simulation module evaluates various dispatch rules and select the best one for a given criterion. The real time control system periodically monitors the shop floor and checks the system performance. If at the beginning of a period, the difference between the actual performance and value estimated by simulation exceeds a given limit (performance limit) due to accumulation of minor system disturbances, then simulation module is called. A new simulation is performed with the remaining operations and a new rule is selected for this period. In addition to periodic monitoring, the rule selection is also done when there is a major system disturbances. In this study, major disturbances include urgent job arrival and major machine breakdowns, whereas tool breakage and small machine breakdowns are considered as minor disturbances. The authors perform experiments for different levels of monitoring periods and performance limits. The experiments show that better results can be obtained by the scheduling module with the moderate monitoring period the performance limit. They also conclude that, the proposed approach has a relatively short response time for the simulation mechanism to be used in real time scheduling.

Sabuncuoglu and Karabuk (1997) study the scheduling rescheduling problem in a static FMS environment. The authors propose several reactive scheduling policies in response to machine breakdowns and processing time variations. Both off-line and on-line scheduling algorithms are analyzed under various experimental conditions like routing and sequence flexibility, queue capacity and efficiency level. The relationship between scheduling frequency and the other operating conditions are extensively investigated. The performance of the system is measured for mean tardiness and makespan criteria. Their experimental results indicate that it is not always beneficial to reschedule the operations in response to every unexpected event and the periodic response with an appropriate period length is also sufficient to cope with the interruptions. Besides, the authors recommend that scheduling frequency

has significant interactions with routing and sequence flexibility, and the effects of scheduling frequency increases as the level of flexibility reduces. Finally, they state that machine breakdowns have more negative impact on the system performance than processing time variation.

In another simulation based study Kutanoglu and Sabuncuoglu (1994) investigate the performance of four reactive scheduling policies under machine breakdowns. Four studied policies are all rerouting, arrival rerouting, queue rerouting and no rerouting. These policies are tested under various experimental conditions on a dynamic job shop system. Existence of material handling system (MHS) is also considered in the experiments. The weighted tardiness measure is used as the performance criteria. The results show that if the MHS is ignored then the all rerouting scheduling policy is preferred as a reactive policy. If there exists MHS in the model, all rerouting policy is suggested. The author mentions that the performance of reactive scheduling policies depends on utilization and capacity of machines and MHS, duration and frequency of the unexpected events but not on the due date tightness and the system efficiency levels. It is also mentioned that no reaction is not seen as an appropriate strategy for reactive scheduling.

Wu and Wysk (1988, 1989) propose a multi-pass scheduling algorithm that utilize simulation to make better scheduling decision for an FMS. They assume that the short term planning module provide the perfect information about the events of the next scheduling period and objectives for which the alternatives are compared. In their study, alternatives are the priority dispatching rules that can be applied in the scheduling of the jobs. The multipass scheduling system simulates each rule by using the current shop status information. By this way, one simulation run is conducted for a short time period, called simulation window. The rule which yield the best performance measure is then selected and implemented during this period. At the end of the period, the procedure is repeated. The main idea behind this application is that combining different dispatching rules in a dynamic and multipass manner creates a better result than applying a single rule alone for the entire horizon. The experimental results show that if the scheduling period (or simulation window) is accurately determined according to the environmental conditions and objectives then this logic is very useful. Therefore, they indicate that the length of the scheduling

interval is a significant factor for the performance of multipass scheduling algorithm.

According to Wu and Wysk (1989), if the window is too short, the statistics collected will not give a reasonable measure of the system performance. But if the window is too long, the simulated system performance may be less sensitive to switching between dispatching rules at right time and the scheduling mechanism will only provide average and aggregate performance measures in each period, which may lose the advantages of the multi-pass scheduling. Wu and Wysk (1988) combine the simulation mechanism with a knowledge based system. By this way, a manufacturing control system is developed that learns from its historical performance and makes own scheduling and control decisions by simulating the alternating combinations of priority dispatching rules. In this case, the rules that will be evaluated are selected from a larger set of rules by the short term planning module by considering the system conditions using knowledge base. In these studies, changing the scheduling decisions are not in response to the stochastic disruptions, but since they consider the system status and update the current schedule, they are included in the reactive scheduling literature.

Recently Kutanoglu and Sabuncuoglu (1995) also propose an iterative simulation-based scheduling mechanism in dynamic job shop environment. By using this scheduling mechanism some part of the scheduling decisions are made at decision points while the remaining decisions are left to be determined according to the dynamic changes in the system. The authors test effectiveness of the proposed method by using multi-pass rule selection algorithm and lead time iteration algorithm in both deterministic and stochastic environments. In the stochastic environment, machine breakdown and processing time variation are considered as disruptions to the system. In the experiments, they analyze the interactions between the forecasting horizon, scheduling period, look-ahead window, and the unexpected events such as machine breakdowns and processing time variations. The experimental results indicate that the iterative improvement procedures improve the performances of the priority dispatching rules significantly at the expense of some computational time. Also, the determination of the look ahead window is an important factor for the multi-pass rule selection algorithm, while lead time iteration algorithm is relatively robust to the lengths of forecasting horizon and scheduling

periods.

Jain & Foley (1987) investigate the effects of the machine breakdowns in a flexible manufacturing environment. In this study, it is assumed that there is a base schedule at the beginning and the objective is to follow the planned schedule as closely as possible. The unexpected event considered is machine breakdown and two on-line reactive scheduling policies are compared: (1) rerouting the jobs from broken machines to alternative machines and (2) holding the interrupted jobs with high priority until interruption is removed. The experiments conducted on different levels of machine breakdown and utilization. The results show that rerouting always outperform the policy of holding jobs.

Bengu (1994) also proposes a simulation based scheduler that uses the up to date information about the current status of the system and aims to improve the performance of a rule (ATC) with the simulation under dynamic and stochastic production environment. In this study, a typical electronics assembly facility (flowline) which is manufacturing electronic products is simulated with machine breakdowns. The aim of the use of the simulation scheduler in such an environment is to select the best look-ahead parameter value for the ATC rule with iterating the simulations. The experiments show that the value for the look-ahead parameter in ATC affects the performance of the rule, and simulation based scheduler is a very effective way of finding a good value for this parameter.

Reactive scheduling is also attracting the increased interest of researchers in developing available knowledge based and artificial intelligence (AI) techniques in real time shop floor control applications. For example, Dutta(1990) develop a knowledge based (KB) methodology to perform real time production control in FMS environments. His proposed mechanism monitors the environment for disruptions and takes corrective actions. He considers machine failures, dynamic introduction of new jobs and dynamic increases in job priority as shop floor disruptions. For a initially generated schedule which is assumed to initially acceptable values for several objectives and the control mechanism aims to maintain these objectives in the presence of disruptions and as corrective actions, the jobs effected by disruptions, are either rerouted, if they have alternative machines, or preempted according to

the priorities and and system conditions. Experimental results show that the KB mechanism with such corrective actions renders effective and robust production control. There are also studies which deal with AI based scheduling and control methods. Among them, ISIS developed by Fox and Smith (1984) and OPIS proposed by Smith et al. (1990) are the most known applications of AI in reactive scheduling problems.

Many other research on KB reactive scheduling is summarized in Szelke and Kerr (1994) review paper. In this study, the authors first introduce the problem through a summary of definitions, then provide an overview of research results in the domain of KB reactive scheduling problem and some reported industrial applications. Also they highlight some major areas for further research.

2.1 Observations

Various types of problems are analyzed in the reactive scheduling literature. The problems differ from each other according to shop environment, job arrival information, schedule generation method, etc. In order to analyze the papers in a more organized manner, we develop a classification scheme and represent the problems by records with seven attributes (see Table 2.1). We use three main divisions (*environment*, *schedule generation* and *implementation* of reactive policies) which define the characteristics of the problems. In the *environment* division we have shop floor type, job arrival information and source of stochasticity attributes. In job arrival attribute, semi dynamic refers the dynamic scheduling problem with a priori known ready times. Under *schedule generation* division, we specify the method to generate schedules and the objective function of the problem. In the Table 1, there are abbreviations in method attributes. These are the name of the scheduling methods given by authors in the papers. Unless any name is stated we give the general approach for the method. Finally, in the *implementation* section, we define when and how the reactive scheduling policies are employed. In *when* attribute, we specify the times at which system revision decisions are held. Under this heading, *event driven* means that rescheduling is triggered in response to an unexpected event

that alters the current system status. In *periodic* policy rescheduling is performed at the beginning of the periods and in *performance based* policy rescheduling is performed if the performance of the system considerably deviates from the a priori found performance. In *how* attribute, the type of corrective action is given. Here, *full* new schedule means that all the available operations are rescheduled according to current system status. *Partial* means that only a part of the current schedule is updated. Job selection refers to the local scheduling decisions like priority dispatching rules.

From the literature review, we can make following observations for the static problems;

- Differences in the performance of fixed sequencing (no action) policy of the off-line scheduling method and dispatching rules decrease as the number of machine breakdowns increases (Yamamoto and Nof, 1985, Sabuncuoglu and Karabuk 1997). Dispatching rules perform even better than fixed sequencing in some cases (Matsuura et al., 1993)
- Relative performance of event driven scheduling/rescheduling method and dispatching rule seems to decrease as the number of machine breakdowns increases, without strong evidence (Yamamoto and Nof, 1985).
- Scheduling/rescheduling methods produce better performance than fixed scheduling or dispatching methods (Yamamoto and Nof, 1985, Bean et al, 1991).
- There is not significant difference between responding only major events (event driven) and responding major disruption plus periodic response (event driven & periodic) (Kim and Kim, 1994).

For the dynamic problems, we observe the following;

- Off-line schedule generation algorithms perform better than dispatching rules (Ovacik and Uzsoy, 1994, Kiran et al., 1991).
- Marginal improvement in the system performance is insignificant after a certain number of rescheduling (Church and Uzsoy, 1992).

Table 2.1: Classification of the papers

Author	ENVIRONMENT			SCHEDULE GENERATION		IMPLEMENTATION	
	Shop Floor	Job Arrival	Stochasticity	Method	Objective Function	When	How
Yamamoto & Nof 1985	Job Shop	Static	Machine Breakdown	Branch and Bound	Makespan	Event Driven (MB)	Full New Schedule
Church & Uzsoy 1992	Single Machine	Dynamic	No	EDD	L max	Periodic & Event Driven (urgent jobs)	Full New Schedule
Holloway & Nelson 1974	Job Shop	Static	Process Time Variation	HSP	Tardiness Rel. Perform. M.	No	Initial Full Schedule
Holloway & Nelson 1977	Job Shop	Dynamic	Process Time Variation	HSP	Tardiness Rel. Perform. M	Periodic	Full New Schedule
Ovacik & Uzsoy 1994	Single Machine	Dynamic	No	Algorithm based on B&B	L max	After Scheduling λ jobs	Partial
Kiran, Alptekin & Kaplan 1991	FMS	Static-Dynamic	No	Multipass Heuristic (FH)	Tardiness Rel. Perform. M.	None-Periodic	Full New Schedule
Kim & Kim 1994	FMS	Semi-Dynamic	Machine Break. Urgent Job	Dispatch Rules	mean FT & T Combination	Periodic & Event Driven	Full New Schedule
Matsuura, Tsubone & Kanezashi 1993	Job Shop	Semi-Dynamic	Mach. Break., Specif. Change, Rush Jobs	B&B, FCFS, SPT	Makespan	After first disruption	Full & Job Selection
Muhleman, Locket & Farn, 1982	Job Shop	Dynamic	Machine Break., Process Time Variation	Dispatch Rules	FT, MT, PL, CMT	Periodic	Full New Schedule
Farn & Muhleman 1979	Single Machine	Dynamic	No	DR & Heuristics based on TSP	Changeover Time	Periodic	Full New Schedule
Bean et al. 1991	Multiple Resource	Static	Mach. Break., Unavail. Tool	MUSA	Weighted Total Tard.	Event Driven	Repair
Akturk & Gorgulu 1993	Modified flow line	Static	Machine Break.	RHSA	Earliness and Tardiness	Event Driven	Repair
Nof & Grant 1991	Small Cell	Static	Machine Break. & Unexpected order arrival		Several	Performance Based, Periodic	Full new sch., right shift, rerouting to alter. mac
Sabuncuoglu & Karabuk 1997	FMS	Static	Machine Break. & Process time variation	Beam search and dispatch rule	Mean tardiness and makespan	Periodic	Full new schedule
Kutanoglu and Sabuncuoglu, 1994	Job Shop	Dynamic	Machine Breakdown	All reroute, arrival reroute, queue reroute, no reroute	Mean Weighted Tardiness	Event Driven	Dispatching rule
Kutanoglu and Sabuncuoglu, 1995	Job Shop	Dynamic	Machine Break-down, Process time Var.	Iterative Simulation	Mean Weighted Tardiness	Periodic	Dispatch rule selection
Wu and Wysk, 1988	FMS	Dynamic	No	Dispatching Rules	Mean Tardiness, Mean Flow Time	Periodic	Partial Simulation window
Wu and Wysk, 1989	FMS	Dynamic	No	Dispatching Rules	Mean Tardiness, Mean Flow Time	Periodic	Partial of Simulation Window
Jain and Foley, 1987	FMS	Static	Machine Breakdown		Mean Tardiness	Event Driven	Rerouting
Bengu, 1994	Flowline	Dynamic	Machine Breakdown	ATC	Mean Weighted Tardiness	No	Job Selection
Dutta, 1990	FMS	Static	Machine Breakdown, New Jobs, Change in Job Priority	Knowledge Based	Mean Completion Time, Mean Machine Utilization	Event Driven	Rerouting, Preemption, etc.

- Heuristic which produced the best performance for the static case is no longer the best for the dynamic case (Muhleman and Farn, 1979).
- When congestion is low, performance of all dispatching rules are quite similar.
- Under a particular rescheduling period and level of congestion, performance of the rules slightly gets worse as the level of uncertainty increases.
- Under a particular rescheduling period and level of uncertainty, performance of the rules improves as the level of congestion is reduced.
- The heuristics become more sensitive to changes in the rescheduling period as the level of uncertainty increases (Muhleman et al, 1982).

2.2 Open Research Points and Motivation

In the scheduling literature, most of the studies deal with the schedule generation techniques. The revision of schedules in response to unexpected changes takes recently the attention of the researches. However, many of the system features are not considered in the literature. For instance, we do not know how the reactive scheduling methods (no response, periodic response, responding every event and on-line scheduling) are affected by the variability in machine loads and the system complexity in both static and dynamic environments. System complexity can be defined by the of number of jobs, number of machines or existence of bottleneck work center. Moreover, in static environment, in periodic response policy instead of generating complete schedules at each rescheduling point, partial scheduling can be employed to save CPU time.

The effects of system complexity and type of load allocation on the performances of reactive scheduling policies are not studied in the dynamic environment, either. Also, we do not know how the reactive scheduling performances are affected by the system stochasticity level (machine breakdown rate, process time variation level, etc.). Furthermore, in the existing studies, job based information is not used. For example, the effects of arrival rate of jobs and job based definitions of scheduling

period (for instance, % new arrival/available jobs, number of new arrivals) are not evaluated. Besides, due date changes of a job are not considered as a disruption to the system. Finally, we do not know whether all the conclusions drawn from the static case are also valid for the dynamic case.

In this thesis, we concentrate our attention to the static job shop scheduling problem. As the open research points indicate that there is not a comprehensive study that analyze the effects of system complexity and load allocation of machines on the reactive scheduling policies. Therefore in the subsequent chapters we are aiming to develop reactive scheduling policies and test their performances under various system complexity and load allocation levels. By performing this analysis, we would extend the related literature in that area.

Chapter 3

BEAM SEARCH

3.1 Introduction

Beam search is a heuristic method for solving optimization problems. It is an adaptation of branch and bound method in which only some nodes are evaluated. In this search method, at any level only promising nodes are kept for further branching and remaining nodes are pruned off permanently. Since a large part of the search tree is pruned off aggressively to obtain a solution, its running time is polynomial in the size of the problems.

This search technique was first used in artificial intelligence for the speech recognition problem (Lowerre, 1976). There have been a number of applications reported in the literature since then. Fox (1983) uses beam search for solving complex scheduling problems by a system called ISIS. Later, Ow and Morton (1988) study the effects of using different evaluation functions to guide the search and compare the performance of beam search with other heuristics for the single machine early/tardy problem and the flow shop problem. They also propose a variation of this technique called filtered beam search and find optimal settings of the search parameters.

In another study, Chang et al. (1989) use beam search as a part of their FMS

scheduling algorithm called bottleneck-based beam search (BBBS). Results indicate that BBBS outperforms widely used dispatching rules for the makespan criterion. Another beam search application to FMSs is reported by De and Lee (1990) who show that the solution quality of filtered beam search algorithm is better than depth-first type search heuristics in terms of the average maximum lateness and average flowtime measures. The authors also show that beam search is better than breadth first type heuristic in terms of number of nodes created during the search. In another study, Hatzikonstantis and Besant (1992) propose a heuristic called A^* for the job shop problem with the makespan criterion. A^* algorithm is very similar to the beam search method. The only difference is that A^* algorithm is a best-first search based heuristic and aims to find minimum-cost paths in search trees. Their computational tests indicate that this heuristic search algorithm performs better than dispatching rules. Finally, Sabuncuoglu and Karabuk (1996) propose a filtered beam search algorithm for more complex FMS environment in which AGVs are explicitly modeled in addition to the routing and sequence flexibilities. Their computational experiments show that the beam search performs better than the machine and AGV scheduling rules under all experimental conditions for the makespan, mean flow time and mean tardiness criteria. Their results also indicate that the beam search based scheduling algorithm exploits flexibilities inherent in FMS more effectively than other methods. An overview of the beam search and its applications to optimization problems can be found in Morton and Pentico (1993).

Even though beam search has been used to solve a wide variety of optimization problems, its performance is not generally known for scheduling problems. Because, in the existing research work, beam search is primarily applied to the FMS scheduling problem with additional considerations on MHS finite buffer capacities and flexibilities and compared with only some dispatching rules. Hence, its relative performances with respect to the known optimum solution and other recently developed heuristics are not known. Besides, it has not been thoroughly studied as a problem solving strategy with certain evaluation functions and search parameters.

This chapter attempts to achieve some of these objectives. First of all, we measure the performance of beam search (with respect to optimum solutions) and compare it with other well known algorithms. In addition, we investigate

the effectiveness of various rules as local and global functions of the beam search applications. The previous research indicates that the values of filter and beam width affect the performance of the beam search. Hence, we also examine the performance of beam search for various values of filter and beam width and find their proper values for the selected problem environments. Furthermore, we test two well known schedule generation schemes (active and nondelay schedule generation schemes) in conjunction to beam search applications to the job shop problems.

The rest of the chapter is organized as follows. The next section gives definitions of the job shop problem. Then a beam search based algorithm is developed for the problem. This is followed by a discussion on test problems and computational experience with the proposed algorithm. The analysis ends with concluding remarks for this scheduling method.

3.2 Problem Definition

The job shop problem is to determine the start and completion time of operations of a set of jobs on a set of machines, subject to the constraints that each machines can handle at most one job at a time (capacity constraints) and each job has a specified processing order through the machines (precedence constraints). Explaining the problem more specifically, there are a finite set J of jobs and a finite set M of machines. For each job $j \in J$, a permutation $(\sigma_1^j, \dots, \sigma_m^j)$ of the machines (where $m = |M|$) represents the processing order of job j through the machines. Thus, j must be processed first on σ_1^j , then on σ_2^j , etc. Also, for each job j and the machine i , there is a nonnegative integer p_{ji} , the processing time of job j on machine i . Since, this problem is NP-Hard (Garey and Johnson, 1979) and very difficult to solve, early studies on this problem directed at development of effective priority dispatching rules. But later, due to the general deficiencies exhibited by priority dispatching rules, researchers concentrate on more complex techniques. Tabu search (Glover, 1989, 1990), large step optimization (Martin et al, 1989) simulated annealing (Matsua et al, 1988; Aarts et al 1991) and genetic algorithms (Nakano and Yamada, 1991) are the examples of the formalized applications of such

scheduling techniques to the job shop problem. A comprehensive bibliography of these studies for the job shop problem is given by Jain and Meeran (1997). In this chapter, we measure the performance of beam search for the makespan and mean tardiness criteria. Makespan, C_{max} is the duration in which all operations for all jobs are completed. Tardiness is the positive difference between completion time and due date of a job. The objective is to determine starting times for each operation in order to minimize the makespan and mean tardiness while satisfying all the capacity and precedence constraints:

$$C_{max}^* = \min(C_{max}) = \min_{feasibleschedules}(\max(C_i) : \forall i \in J).$$

$$T^* = (1/|J|)\min_{feasibleschedules}(\sum_{i \in J} \max(0, C_i - d_i)).$$

where C_i and d_i are the completion time and due date of job i , respectively.

3.3 Beam Search

Beam Search is like breadth-first search since it progresses level by level without backtracking. But unlike breadth-first search, beam search only moves downward from the best β promising nodes (instead of all nodes) at each level and β is called *beam width*. The other nodes are simply ignored. In order to select the best β nodes, promise of each node is determined. This value can be determined in various ways. One way is to employ an evaluation function which estimates the minimum total costs of the best solution that can be obtained from the partial schedule represented by the node. Such an evaluation function may require as little effort as computing some priority rating or as much as completing the partial schedule by some method. The former method is called *one-step priority evaluation function*, and the latter case is called *total cost evaluation function*. The one-step priority evaluation function has a *local* view, whereas, total cost evaluation employs a projecting mechanism to estimate costs from the current partial solution. Therefore, evaluation is based on a *global* view of the solution. Unfortunately, there is a trade-off between these two approaches: one-step (local) evaluation is quick but may discard good solutions. On the other hand, more thorough evaluation by the global function is more accurate

but computationally more expensive.

A filtering mechanism is also proposed in the literature to reduce the computational burden of beam search. During filtering some nodes are discarded permanently based on their local evaluation function values. Only the remaining nodes are subject to global evaluation. The number of nodes retained for the further evaluation is called *filter width* (α). The roles of evaluation functions and meaning of search parameters are depicted in Figure 3.1.

After filtering, based on the outcome of the global evaluation, one node (*beam node*) is selected among the descendants of each node. Since we have beam width number of nodes in the former level while keeping one descendant, we again have beam width number of nodes in the next level and therefore the search progresses through β parallel beams. Different nodes at the same level represent different partial schedules. If the local evaluation is a function of the partial schedule (as in the case of the lower bound based local evaluation function to minimize makespan), values of the local evaluation function obtained for expanding one node cannot be compared legitimately with the values of the local evaluation functions obtained for expanding another node at the same level. Therefore, nodes in each parallel beams are evaluated separately and only one node is selected for each beam.

3.3.1 The Proposed Beam Search Based Algorithm

In an algorithm like beam search, there are two key components: (1) search tree representation and (2) application of a search methodology. As mentioned earlier, in the search tree, each node corresponds to a partial schedule. A line between two nodes represents a decision to add a job to an existing partial schedule. Consequently, leaf nodes at the end of the tree corresponds to complete schedules. Baker (1974) describes two search tree generation procedures (active and nondelay) schedules for the job shop systems. In the proposed algorithm, these procedures are used to generate branches from a given node.

Second issue in beam search is the determination of search methodology. In the

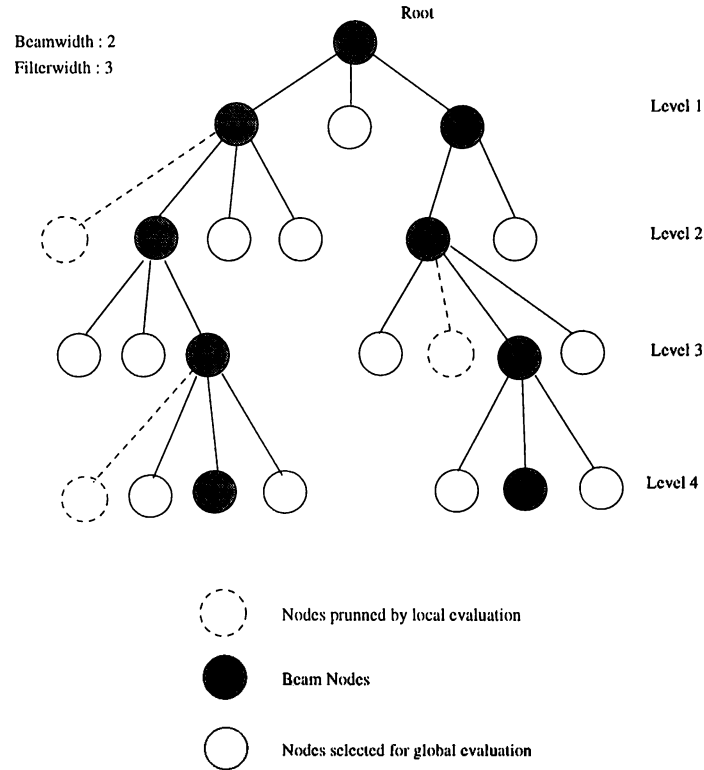


Figure 3.1: Representation of beam search tree

proposed algorithm, the filtered beam search method is used to perform search in the tree. Quality of filtered beam search depends on the quality of local and global evaluation functions as well as beam and filter width parameters. Therefore, a thorough analysis must be carried out to determine the nature of these functions and parameters. In this study, local evaluation is performed by using simple dispatching rules. Global evaluation of a node is determined as the estimation of upper bound value for the solutions that can be generated if that node is added to the partial schedule. In the proposed algorithm, this is performed by generating a complete schedule from a given partial schedule by some dispatching rules and reading the value of objective function. Priority rule in local evaluation function and dispatching rule in global evaluation function are not necessarily the same. In this study, we test several rules for this purpose.

In the proposed algorithm, all the nodes at level 1 are globally evaluated to determine best β number of promising nodes. The selected nodes become the first nodes of the β number of parallel beams. In the subsequent levels, descendants of

the beam nodes are first locally evaluated to find α number of promising nodes and then these nodes are further globally evaluated to select the next beam node. If the number of nodes expanded in the first level is less than specified beam width, then all the nodes are expanded until the number of nodes is greater than beam width in the next level.

To be more specific, procedural form of the beam search based algorithm is given as follows:

Procedure (BEAM-SEARCH)

In the filtered beam search algorithm, we use active or nondelay schedule generation methods developed by Baker (1974, pp 189) in order to generate search tree. At each level of the methods, operations that have already been assigned starting times make up a partial schedule. Given a partial schedule for any job shop problem, a set of schedulable operations can be constructed. Let

- PS_t = a partial schedule containing t scheduled operations
- S_t = the set of schedulable operations at stage t , corresponding to a given PS_t
- σ_t = the earliest time at which operation $j \in S_t$ could be started
- ϕ_t = the earliest time at which operation $j \in S_t$ could be completed.

Active schedule generation subroutine (ACTIVE)

- Step 1 Determine $\phi^* = \min_{j \in S_t} \{\phi_j\}$ and the machine m^* on which ϕ^* could be realized
- Step 2 For each operation $j \in S_t$ that requires machine m^* and for which $\sigma_j < \phi^*$, generate a new node which corresponds to the partial schedule in which operation j is added to PS_t and started at time σ_j .

Nondelay schedule generation subroutine (NONDELAY)

- Step 1 Determine $\sigma^* = \min_{j \in S_t} \{\sigma_j\}$ and the machine m^* on which σ^* could be realized
- Step 2 For each operation $j \in S_t$ that requires machine m^* and for which $\sigma_j = \sigma^*$, generate a new node which corresponds to the partial schedule in which operation j is added to PS_t and started at time σ_j .

We now give the steps of our beam search based algorithm.

Beam Search

Step 0 (Node generation). Generate nodes from the parent node by using the procedure ACTIVE (or NONDELAY) with PS_t as the null partial schedule.

Step 1 (Checking the number of nodes). If the total number of nodes generated is less than beamwidth, then move down to one more level, generate new nodes from each node by using the procedure ACTIVE (or NONDELAY) with PS_t as the partial schedule represented by the node, and go to Step 1. Else, go to Step 2.

Step 2 (Computing global evaluation functions). Compute the global evaluation function values for all the nodes and select the best beamwidth (b) number of nodes (initial beam nodes).

For each initial beam node

Step 3 (Determining beam nodes). While the number of levels is less than the number of operations (denoted as n).

Step 3.1 (Node generation). In the next level, generate new nodes from the beam node according to the procedure ACTIVE (or NONDELAY) with PS_t as the partial schedule represented by the beam node. Let k is the number of nodes generated.

Step 3.2 (Computing local evaluation functions). Compute local evaluation function values for each node.

Step 3.3 (Filtering). Choose the best f' number of nodes according to local evaluation function values (f is filterwidth and $f' = \min(k, f)$).

Step 3.4 (Computing global evaluation functions). Compute global evaluation values of each f' number of selected nodes.

Step 3.5 (Selecting the beam nodes). Select the node with the lowest global evaluation value (i.e., beam node). For the partial schedule represented by the beam

node update the data set as follows:

- (a) Remove operation j from S_t
- (b) Form S_{t+1} by adding the direct successor of operation j to S_t
- (c) Increment t by one

Step 4 (Selecting the solution schedule). Among the beamwidth number of schedules, select the one with the best objective function value.

Numeric Example

Consider the following job shop system with two machines and four jobs and each job has two operations. The processing times and routing information of the jobs is given in Table 3.1 below.

Table 3.1: Job information

JOB	Operation 1		Operation 2	
	Processing Time	Machine	Processing Time	Machine
1	13	1	53	2
2	54	1	42	2
3	1	2	9	1
4	78	1	50	2

In the problem makespan is the performance measure. The filtered beam search with beamwidth and filterwidth of 2 is applied to this problem. Nondelay branching scheme is used to generate search tree. The “most work remaining” (MWR) priority index is taken as the local evaluation function and the MWR dispatching rule is used as the global evaluation function.

The beam search tree of the problem is shown in Figure 3.2 in which GF and LF refer to global and local evaluation function values, respectively. The shaded nodes are the beam nodes and the nodes with crosses are the ones that are pruned off as a result of the global or local evaluation. The labels under the nodes give job number and operation number.

In the algorithm, the first stage, from of Steps 0 to 2, is to determine the initial beam nodes. As seen from Figure 3.2, in Step 0, we generate three nodes (i.e., J1-O1, J2-O2 and J4-O4) from the root by using the procedure NONDELAY. In Step 1, since

number of nodes generated (3 in this case) is greater the beam size of 2, we directly go to Step 2 in which their global evaluation function values (makespan measure of the complete schedule generated by the MWR rule from the partial schedule represented by the node) are computed and the best two nodes are selected on the basis of their global evaluation values. These nodes are J1-O1 & J4-O1. If the number of nodes were smaller than the beam size, we would continue to expand until the number of nodes are greater than the beam size at the next level.

After determining the initial beam nodes, Step 3 of the algorithm is executed to schedule all the remaining operations. Since, Step 3 can be performed for each beam node independently, we only demonstrate the procedure of the proposed algorithm for the left beam.

Step 3 is implemented from level 2 to the end of search three. At each level we generate the new nodes, perform filtering, compute the global evaluation functions and finally select the node which corresponds to the operation added to the current partial schedule. After level 1, the partial schedule represented on the left beam is as follows:

$PS_1 = \{ (J1, O1, 0, 1) \}$ where the array refers to (Job no, Operation no, starting time, machine number).

At level 2

- Step 3.1 only one node (J3-O1) is generated according to the procedure NONDELAY. Hence $k=1$.
- Step 3.2 local evaluation function value of the node is calculated to be 10. Note that in the example we use the MWR priority index as local evaluation function.
- Step 3.3 f' is computed as 1 ($f' = \min(k, f) = \min(1, 2) = 1$) and the node (i.e., J1-O1) is the only alternative for global evaluation.
- Step 3.4 global evaluation function value of this node is 187.
- Step 3.5 again this node is the only alternative to be the beam node (which corresponds to the operation to be added to the partial schedule) and then the current partial schedule becomes.

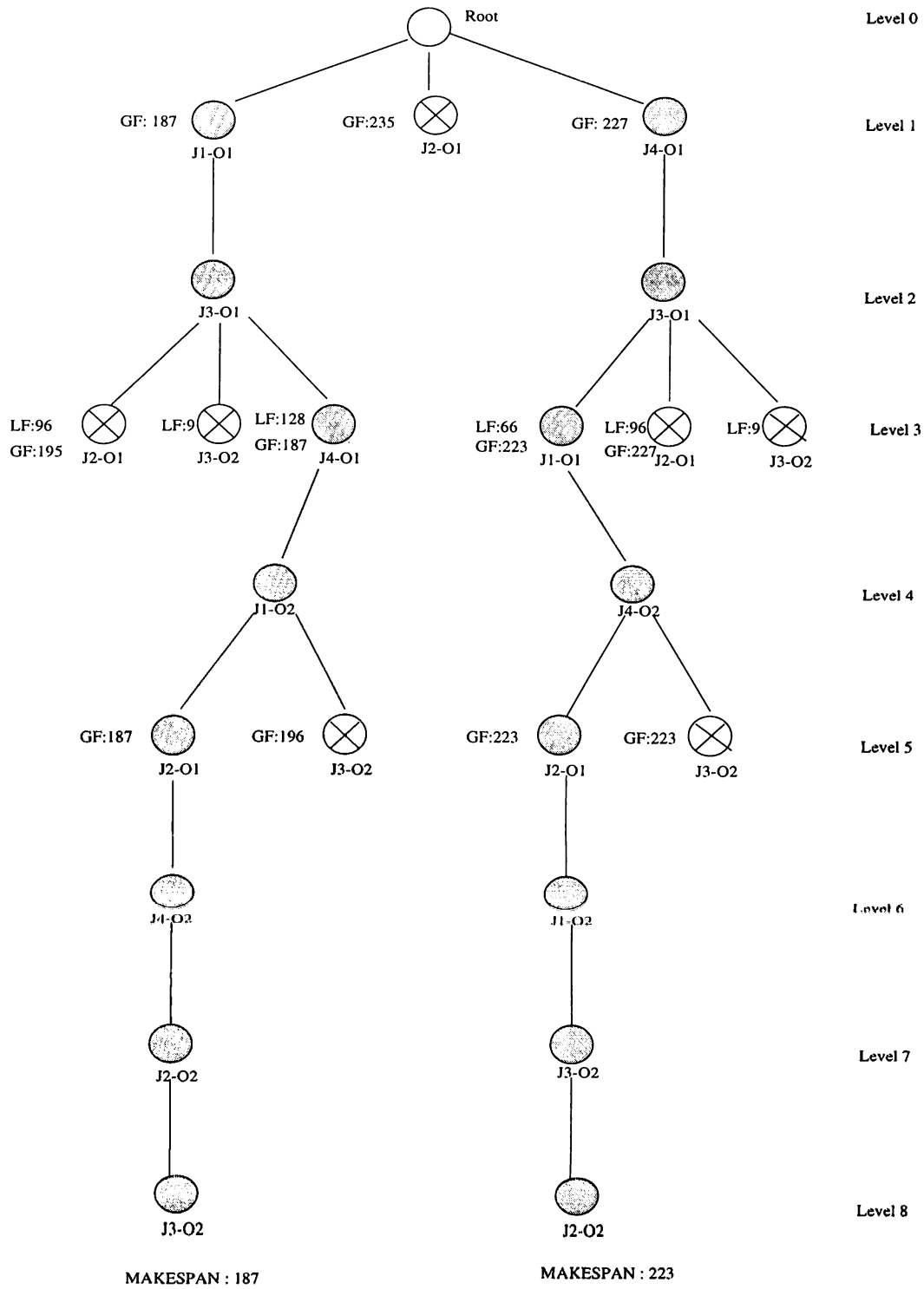


Figure 3.2: Beam search tree for the numerical example

$$PS_2 = \{ (J1, 01, 0, 1), (J3, 01, 0, 2) \}$$

At level 3.

- Step 3.1 three nodes are generated (J2-O1, J3-O2, J4-O1) from the beam node according to the procedure NONDELAY. Hence k is 3
- Step 3.2 the local evaluation function values are calculated for each node as displayed by their LF values in Figure 3.2.
- Step 3.3 Since f' is 2 ($\min(3, 2)$). Two nodes (J2-O1 and J4-O1) are selected according to their local evaluation function values. Hence, the node J3-O2 is eliminated because of the lowest local function values.
- Step 3.4 the selected two nodes are globally evaluated. These are displayed as GF values in Figure 3.2.
- Step 3.5 the node (J4-O1) with the lowest global evaluation value is selected as the beam node and the operation represented by this node is now added to the current partial schedule. Therefore the partial schedule becomes:

$$PS_3 = \{ (J1, 01, 0, 1), (J4, 01, 13, 1), (J3, 01, 0, 2) \}$$

Since there is only one node generated, the execution of steps as level 4 is the same as the ones at level 2. Hence it will not be repeated here. Instead we give the resulted partial schedule at level 4.

$$PS_4 = \{ (J1, 01, 0, 1), (J4, 01, 13, 1), (J3, 01, 0, 2), (J1, 02, 13, 2) \}$$

At level 5,

- Step 3.1 two nodes (J2-O1, J3-O2) are generated according to the procedure NONDELAY. Hence k=2.
- Step 3.2 local evaluation function values are calculated as 96 and 9 for J2-O1 and J3-O2, respectively.
- Step 3.3 since filterwidth is 2, both of them are further evaluated.
- Step 3.4 global evaluation function values are calculated as 187 and 196.

Step 3.5 the node (J2-O1) with the lowest global evaluation function value is selected as the beam node and the operation represented by this node is now added to the current partial schedule as follows:

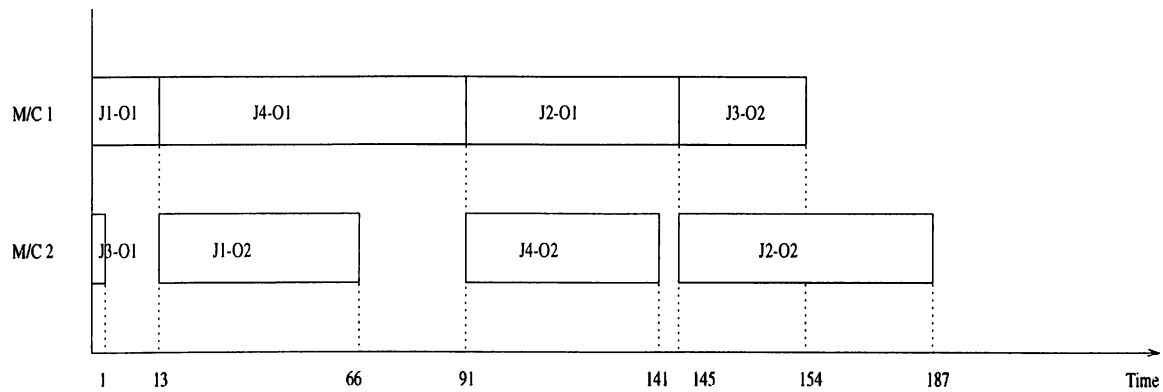
$$PS_5 = \{ (J1, O1, 0, 1), (J4, O1, 13, 1), (J2, O2, 91, 1), (J3, O1, 0, 2), (J1, O2, 13, 2) \}$$

In levels 6, 7 and 8 since again only one node is generated, the steps at these levels are the same as those of level 2. After assigning all the operations, we obtain a complete schedule and the Gantt chart of this resulting schedule is displayed in the Figure 3.3a.

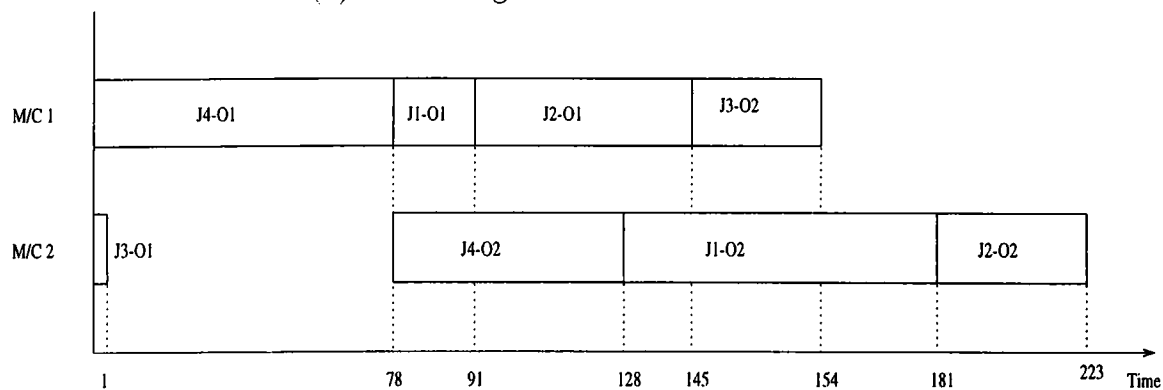
The same procedure is executed for the right beam and the Gantt chart of the resulting schedule is displayed in Figure 3.3b. Note that at the end of the algorithm, we have two complete schedules. In Step 4, we compute makespan of both schedules and select the one with the minimum value. It turns out that, in Figure 3.2, the beam on the left hand side gives the better schedule.

The complexity of the beam search is $O(n^3)$, where n is the number of operations to be scheduled in the job shop problem. The search tree generated by the algorithm has n levels, and at each level only beamwidth (b) number of nodes are kept for further expansions. Hence, the number of subproblems considered in the search is equal to bn . At any level, filterwidth (f) number of nodes are selected by a local evaluation function among the nodes generated from a beam node. For the job shop problem, number of nodes expanded from a root node is much less than total number of operations (n). Hence, the number of local evaluation operations in the search tree is less than bn^2 .

At any branch, only filterwidth number of nodes are selected for global evaluation. Therefore, the number of global evaluation is at most equal to fbn . For a global evaluation operation, at each level at most n number of nodes are locally evaluated and one node is selected for further expansion. These selected nodes are expanded further down to at most n levels. Hence, for one global evaluation function requires at most n^2 number of local evaluation operations. Since there are fbn global evaluation, this part of the search requires at most fbn^3 number of local evaluations.



(a) Schedule generated on the left beam



(b) Schedule generated on the right beam

Figure 3.3: Schedules generated by the beam search algorithm

In summary, $bn^2 + fbn^3$ number of local evaluation operations have to be performed. In the proposed version of the beam search, local evaluation is taken as computation of a priority index. This makes the total computation work is proportional to n^3 . Therefore, the proposed algorithm has $O(n^3)$ complexity.

3.4 Makespan case

The performance of the scheduling algorithm is first measured for the makespan criterion. Effects of different local and global functions and various beam & filter width levels are also evaluated in the experiments. Since the optimal results of some test problems are known in the literature for the makespan criterion, the performance of the algorithm is tested in terms of the percent deviation from optimality. Some of these problems are given in Applegate and Cook (1990). These problem are

generated according to format described in the previous section. Each of the test problems used in this study has 10 machines and 10 jobs. The problems ABZ5 and ABZ6 are from Adams et al. (1988); the problems LA16 through LA20 are from Lawrence (1984); the problems ORB3, ORB4 and ORB5 stated again in Applegate and Cook (1990).

Since the proposed algorithm is a heuristic, it is also compared with well known dispatching rules, such as MWR, MTWK & LPT. These rules are implemented via the nondelay scheduling scheme proposed by Baker (1984).

Before the analyses, we have tested performances of MWR, LPT, MTWK, SPT dispatching rules and note that MWR outperforms the others. Therefore MWR is used as the nondelay dispatching rule in the global evaluation function to complete the partial schedule from the nodes. For the local evaluation function, however, both MWR and a lower bound proposed by Baker (1974) are considered in the algorithm. The complete definition of these rules and the lower bound are given in Table 3.2 where S_t is the set of unscheduled operations at level t ; σ_j is the earliest time at which operation j could be started; R_j is the unscheduled processing for the job corresponding to operation j ; f_k is the latest completion of an operation on machine k ; M_k is the unscheduled processing that will require machine k .

Table 3.2: Descriptions of priority rules used in makespan analysis

Rule	Description
MWR (Most Work Remaining)	$MWR_{ij} = \sum_{q=j}^{m_i} p_{iq}$
MTWK (Most Total Work)	$MTWK_{ij} = \sum_{q=1}^{m_i} p_{iq}$
LPT (Longest Processing Time)	$LPT_{ij} = p_{ij}$
LB (Lower Bound)	$LB_{ij} = \max(\max_{j \in S_t}(\sigma_j + R_j), \max_{1 \leq k \leq m}(f_k + M_k))$

In addition, two different schedule generation (or search tree representation) schemes are tested in this study. As discussed in Baker (1974), the search trees are expanded by either the active or nondelay schedule generation procedures. In an active schedule, no operation is started earlier without delaying some other operations whereas in a nondelay schedule, no machine remains idle when there exist a schedulable operation. A combination of two evaluation functions and two schedule generation schemes results in four versions of the proposed beam search algorithm (see Table 3.3).

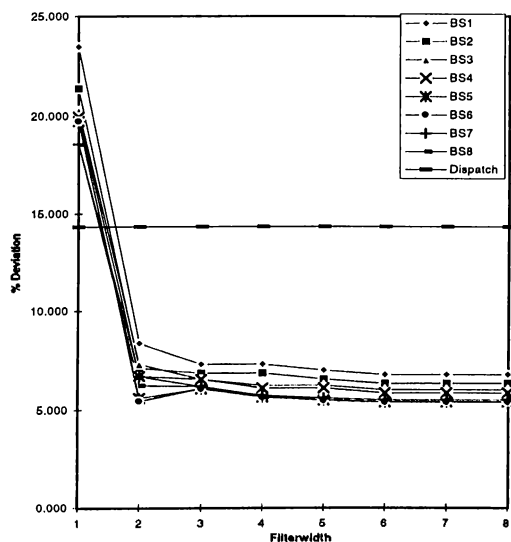
Table 3.3: Searching Methods used in the Makespan Case

Search Tree Representation	Local Evaluation Rule	Global Evaluation Function
Active	MWR	Nondelay schedule generation using MWR
Nondelay	MWR	Nondelay schedule generation using MWR
Active	LB	Nondelay schedule generation using MWR
Nondelay	LB	Nondelay schedule generation using MWR

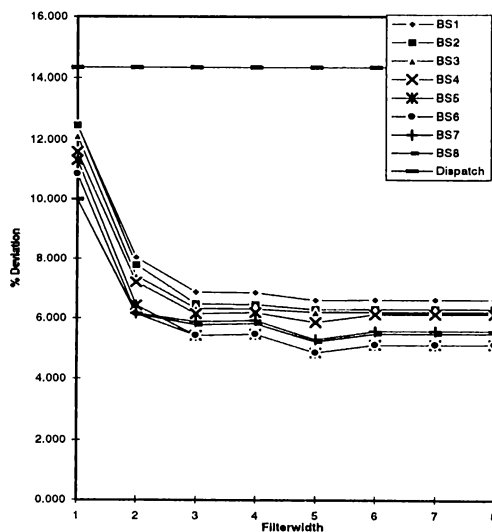
The results of the experiments are depicted in Figure 3.4 for different beam and filter width values. The vertical axis shows the average (over 10 problems) deviation from optimality. Each curve represents the results of a particular beam width (i.e., BS1, BS2, etc). The horizontal axis measures filter width. The horizontal lines in the graphs are the performances of the nondelay MWR dispatching heuristic.

In general, the performance of the algorithm changes for different search tree representation schemes, local evaluation functions and beam and filter width values (Figure 3.4). Therefore, each of these factors is tested separately in order to use the best version of filtered beam search method in the later stages of the research. In the L-shaped graphs, one can observe some erratic behaviors. These behaviors are mainly due to the imperfectness of the local and global evaluation functions. In general, local evaluations are cheaper but also less accurate. Thus, too small filter width makes it more likely that errors in the local estimate prevents good nodes from being passed to global evaluation. Therefore, if we increase the filter width, more nodes enter the second stage, and nodes erroneously valued by the local estimate will have a second chance. If the global estimate happens to be more accurate, then the node will be saved as appropriate in the second stage. However, both estimates could be imperfect. Hence, the larger filter width in this case forces a poor node, according to the local evaluation function, to pass to the second stage of the evaluation. The global method may then erroneously save these bad nodes. When this happens, the performance of the search may deteriorates as the filter width increases while keeping the beam width constant. Since either estimate is not very accurate the above behavior is observed.

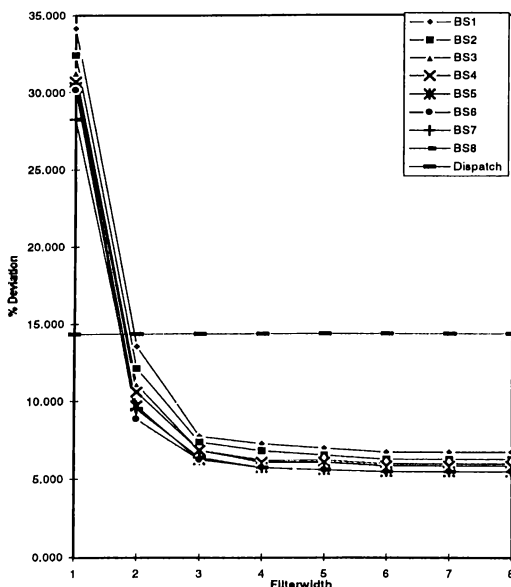
We also note that, if there are fewer nodes expanded than the size of beam width at the level 1, all the nodes are further expanded in the next level until the number



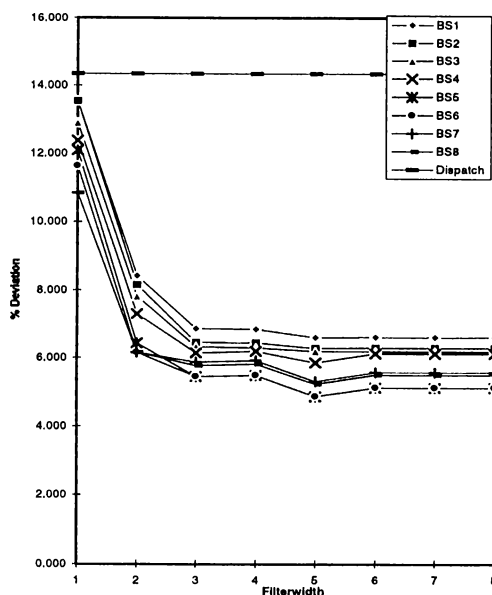
(a) Active Branching
Local Evaluation: MWR
Global: Nondelay Dispatch (MWR)



(b) Nondelay Branching
Local Evaluation: MWR
Global: Nondelay Dispatch (MWR)



(c) Active Branching
Local Evaluation: LB
Global: Nondelay Dispatch (MWR)



(d) Nondelay Branching
Local Evaluation: MWR
Global: Nondelay Dispatch (MWR)

Figure 3.4: Percent deviation from optimal solution vs filterwidth

of nodes are greater than the beam width. Then all the nodes are globally evaluated to select beam nodes. If we increase the beam width, we may have more such nodes to evaluate by the global function. Similarly, if the global estimate is imperfect, global function may then mistakenly select inferior nodes as the beam nodes. In this case, the performance of the search heuristics may deteriorate as the beam width increases if filter width is kept constant. We observe such a behavior in the nondelay schedule generation scheme. But the amount of deterioration on the schedule is negligible.

Active vs nondelay Schedules

In the analysis, both active and nondelay branching methods are used to generate search trees. Our computational experiments indicate that overall performance of the nondelay branching scheme is better than the active branching approach for small filter widths. However, as the filter width increases beyond a certain limit, the active scheme starts performing slightly better than the nondelay scheme. We also observe that the best result found by the active scheme is worse than the one found by the nondelay scheme. Even though this finding may seem to be counter intuitive, it is consistent with the generally held view that the nondelay scheduling scheme produces better results than the active scheduling scheme when used with dispatching rules for the makespan criterion (Baker, 1974).

Local evaluation functions

We measure the performance of two local evaluation functions: the lower bound discussed in Baker (1974) and the MWR rule. A lower bound value of a partial schedule is computed as the maximum of a job based bound value and machine based bound value. The job based bound value is equal to MWR plus earliest possible start time (Table 3.2). This is generally greater than a machine based value. Since evaluation function based on the lower bound resembles the MWR rule, and the results of MWR and lower bound are not expected to differ too much. However, as depicted in Figure 3.4, the MWR local function yields better results than the lower bound estimate. The details of these results are given in Table A.1 in Appendix A.

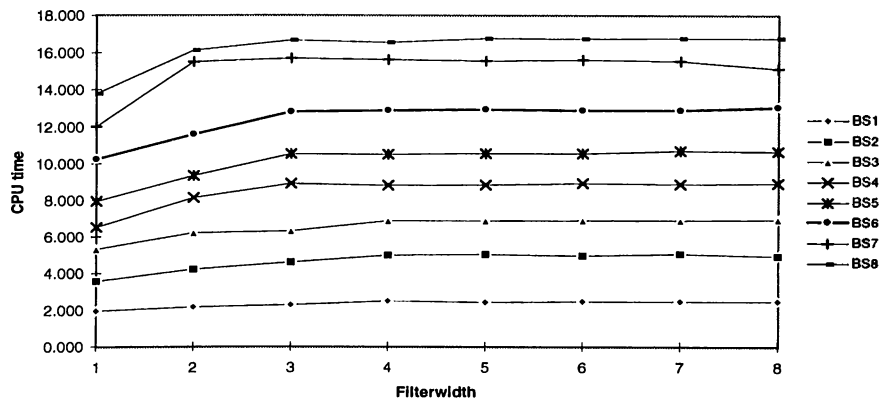
Filter and beam width

In the classical job shop problem, there is not much flexibility as in the case of FMS. Hence, relatively fewer number of nodes are expanded from the root node in the search tree and all such nodes are locally evaluated for large filter widths. Consequently, the performance of the algorithm does not change significantly as the filter width increases beyond a certain limit. From our experiments, it appears that the smallest value of the filter width that gives reasonably good performances for a given beam width is approximately five. Our experiments also show that increasing the beam width improves the solution quality. This is because we continuously expand levels of the tree until we have more number of nodes than the beam size. As a result, for large beam sizes, we have larger pool of nodes to evaluate and hence better solution possibilities. But one should also take into account computation time requirements. As seen from Figure 3.5, the higher the beam width, the higher the CPU time needed to execute the algorithm. It seems that the best beam size is five when considering both the CPU times and the quality of solutions. These parameter settings are used in the latter stages of experiments.

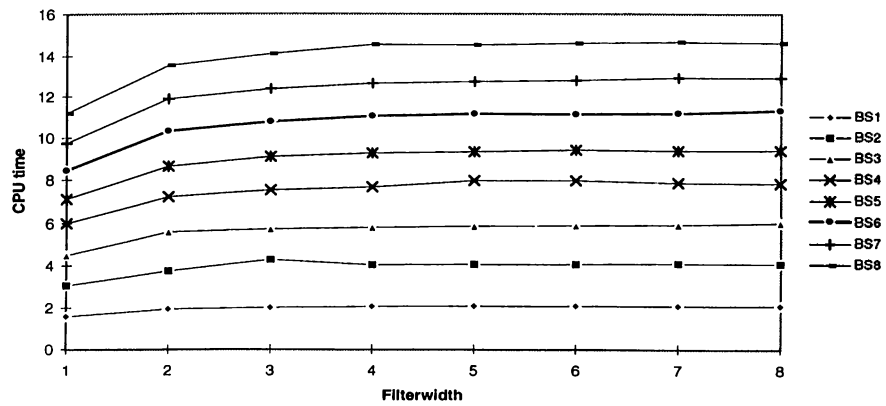
3.4.1 Computational results

In order to measure the makespan performance of beam search algorithm in the job shop environment, we used well known benchmark problems reported in the literature. These are: 40 problems generated by Lawrence (1984), 2 problems used by Adams et al. (1988), and 5 problems mentioned in Applegate and Cook (1990). The famous FT10 problem is also included in the experiments. Both the proposed algorithm and the rules are run on Sparc Station Classic with one 60 MHz micro SPARC 8-CPU and 1 GB memory. The codes are written in the programming language C.

The computational results are given in Table 3.4. Each cell in that table represents the average deviation from optimality and the computation times in CPU seconds for the corresponding problem instance. Since solution times of the dispatching rules are very small (e.g. less than 10^{-2} seconds), the CPU times of the



(a) CPU time vs Filterwidth for Makespan Objective



(b) CPU time vs Filterwidth for Average Tardiness Objective

Figure 3.5: CPU time vs filterwidth

rules are not included in this table.

As expected, the performance of the algorithm in terms of the average percent deviation from optimality is much better than the rules. The average deviation of the algorithm is % 4.26 for all the test problems, while it is 16, 29, and 13 percents for SPT, LWK, MWK, respectively. These rules solve only three problem instances to optimality, but the proposed algorithm solves 16 out of 48 instances, including the most difficult problems FT10, LA36,...,LA40. For the instances which we do not know optimal solutions, the percent deviations from optimality cannot be calculated and thus these cell are shown as '*’.

Even though beam search is a branch and bound based algorithm, the CPU times is not very high. It appears that the number of jobs (rather than machines) is the most determining factor for its computational time requirement.

We also note that the performance of the algorithm changes for different problem types. It seems that the algorithm performs very well if the number of jobs is greater than the number of machines which we call rectangular instances. It also appears that the square type instances (number of machines equals to number of jobs) are hard instances for the beam search algorithm.

We also compare the beam search algorithm with other heuristic methods whose performances are reported for some selected problems. In a recent study, Aarts et al. (1994) propose multi-start iterative improvement (MSII), threshold accepting (TA), simulated annealing (SA) and genetic local search (GLS) algorithms for job shop problems. The authors use two neighborhood structures (1 & 2) in their algorithms. They apply these solution methods to 43 problem instances, among which 40 of them are also common in our test set. The resulting performances of their algorithms and our beam search based algorithm (referred to as BS) are given in Table 3.5.

In terms of average % deviation from optimality, the solution quality of BS is only better than the multi-start iterative improvement methods (MSII1, MSII2) and close to threshold accepting method (TA1). In terms of the maximum % deviation and number of optimally solved instances, the performance of the beam search based algorithm is still better than MSII1, MSII2 and TA1 and competes with the well

Table 3.4: Results of test problems for the Makespan Analysis

Problem	Optimum	Algorithm b=5, f=5			SPT		LPT		MWR	
		Solution	% Dev.	CPU	Solution	%Dev.	Solution	% Dev.	Solution	%Dev.
10*5										
LA01	666	666	0	2.5	751	12.8	933	40.01	735	10.4
LA02	655	704	7.48	2.9	821	25.3	830	26.7	817	24.7
LA03	597	650	8.88	3	672	12.6	822	37.7	696	16.6
LA04	590	620	5.09	2.8	711	20.5	833	41.2	758	28.5
LA05	593	593	0	3.2	610	2.9	766	29.2	593	0
Averages			4.29			14.82		34.98		16.04
15*5										
LA06	926	926	0	13.6	1200	29.6	1067	15.2	926	0
LA07	890	890	0	12.2	1034	16.2	1136	27.6	970	9
LA08	863	863	0	14.7	942	9.2	1176	36.3	957	10.9
LA09	951	951	0	12.1	1045	9.9	1334	40.3	1015	6.7
LA10	958	958	0	14.2	1049	9.5	1312	37	966	0.8
Averages			0			14.88		31.28		5.48
20*5										
LA11	1222	1222	0	45.4	1473	20.5	1525	24.8	1268	3.8
LA12	1039	1039	0	39.8	1203	15.8	1305	25.6	1137	9.4
LA13	1150	1150	0	44.9	1275	10.9	1354	17.7	1166	1.4
LA14	1292	1292	0	43.3	1427	10.4	1725	33.5	1292	0
LA15	1207	1207	0	41.6	1339	10.9	1648	36.5	1343	11.3
Averages			0			13.7		27.62		5.18
10*10										
LA16	945	988	4.55	10.7	1156	22.3	1347	42.5	1054	11.5
LA17	784	827	5.49	9.6	924	17.9	1203	53.4	846	7.9
LA18	848	881	3.89	10.2	981	15.7	1154	36.1	970	14.4
LA19	842	882	4.75	8	940	11.6	986	17.1	1013	20.3
LA20	902	948	5.1	8.8	1000	10.9	1232	36.6	964	6.9
FT10	930	1016	9.2	74	1074	15.5	1324	42.4	1108	19.1
ABZ5	1234	1288	4.4	10.8	1352	9.6	1735	40.6	1369	10.9
ABZ6	943	980	3.9	14.6	1097	16.3	1110	17	987	4.7
ORB1	1059	1174	10.85	14.5	1478	39.6	1398	32	1359	28.3
ORB2	888	926	4.27	10.9	1175	32.3	1170	31.8	1047	17.9
ORB3	1005	1087	7.54	12.9	1179	17.3	1389	38.2	1247	24
ORB4	1005	1036	3.09	11.9	1236	23	1432	42.5	1172	16.6
ORB5	887	968	9.13	11.2	1152	29.9	1175	32.5	1173	32.2
Averages			5.86			20.14		35.58		16.53
15*10										
LA21	1040-1053 ⁺	1154	*	44	1324	*	1518	*	1264	*
LA22	927	985	6.26	44.3	1180	27.3	1589	71.4	1079	16.4
LA23	1032	1051	1.84	39.8	1162	12.6	1374	33.1	1185	14.8
LA24	935	992	6.1	39.3	1203	28.7	1214	29.8	1101	17.8
LA25	977	1073	9.83	43.1	1449	48.3	1487	52.2	1166	19.3
Averages			6.01			29.23		46.63		17.08
20*10										
LA26	1218	1269	4.19	136.1	1498	23	1606	31.9	1435	17.8
LA27	1235-1269 ⁺	1361	*	129.8	1784	*	1728	*	1442	*
LA28	1216	1373	12.91	137.2	1610	32.4	1750	43.9	1487	22.3
LA29	1120-1195 ⁺	1252	*	140.7	1556	*	1665	*	1337	*
LA30	1355	1435	5.9	144.6	1792	32.3	2067	52.5	1534	13.2
Averages			7.67			29.23		42.77		17.77
30*10										
LA31	1784	1784	0	810.3	1951	9.4	2322	30.2	1931	8.2
LA32	1850	1850	0	806	2165	17	2341	26.5	1875	1.4
LA33	1719	1719	0	818.9	1901	10.6	2125	23.6	1875	9.1
LA34	1721	1780	3.42	823.5	2070	20.3	2223	29.2	1935	12.4
LA35	1888	1888	0	684.2	2118	12.2	2316	22.7	2118	12.2
Averages			0.68			13.89		26.43		8.66

⁺ refers to the problems that we do not know their optimal makespan values

Table 1.4: (Cont'd) Results of test problems for the Makespan Analysis

Problem	Optimum	Algorithm b=5, f=5			SPT		LPT		MWR	
		Solution	% Dev.	CPU	Solution	%Dev.	Solution	% Dev.	Solution	%Dev.
15*15										
LA36	1268	1401	10.47	98.7	1681	32.6	1908	50.5	1521	20
LA37	1397	1503	7.59	99.2	1693	21.2	1884	34.9	1643	17.6
LA38	1171-1184	1297	*	93.7	1509	*	1686	*	1477	*
LA39	1233	1369	11.03	95.8	1447	17.4	1894	53.6	1443	17
LA40	1222	1347	10.23	100	1495	22.3	1661	35.9	1475	20.7
Averages			9.83			23.36		43.72		18.82
Average of Means			4.26			15.99		28.62		12.16

known searching schemes (simulating annealing and genetic local search methods)

Bear in mind that BS is a constructive algorithm whereas others are iterative procedures whose computation times can be indeed very long. In Aarts et al. (1994) the average of running times reported for each problem are much larger than the CPU time requirements of BS. Even for small sized problems, the differences between CPU times are significant, (i.e., the beam search method is approximately 10 and 6 times faster than the other methods for 10 jobs 10 machines and 15 jobs 15 machines problems, respectively).

Table 3.5: Comparison of scheduling algorithms

	MSII1	MSII2	TA1	SA1	SA2	GLS1	GLS2	BS
Avr. % Dev.	11.73	8.96	3.73	1.52	1.51	2.22	1.92	4.47
Max. % Dev.	32.61	27.36	12.98	10.13	9.45	15.20	12.50	11.79
# of optimum	4	7	15	19	18	16	16	16

3.5 Mean tardiness case

The performance of the beam search based algorithm is also measured in terms of the mean tardiness. Again, we first examine evaluation functions and find proper settings of the beam search parameters. Since optimum solutions of the job shop problems are not generally known in the tardiness case, we only compare the performance of the algorithm with dispatching rules.

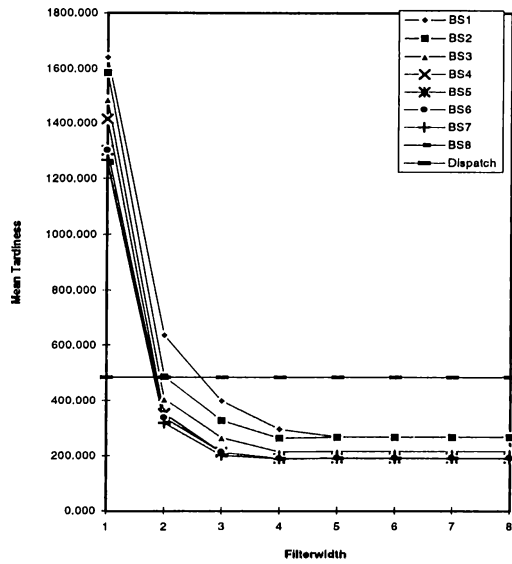
During computational experiments, we use the modified version of the problem data used in the makespan case. Specifically, we append the due date information to the data sets using the TWK due date assignment method (Baker, 1984). Based on

pilot runs, the proportionality constant (tardiness factor) of TWK is set to 1.5 for the tight due date case (corresponds to 50 percent tardy jobs) and 2 for the loose due date case (corresponds to 6 percent tardy jobs). These two tardiness levels are very close to each other because the due dates are assigned in proportional to total processing time, instead of average processing time. We also know that in low utilization rates (such as the case in our model with 62% utilization), the closeness of tardiness factors is expected as indicated by Baker (1984, pp.1099). In this analysis we use the tardiness factor of 1.5 to determine the due dates of the jobs.

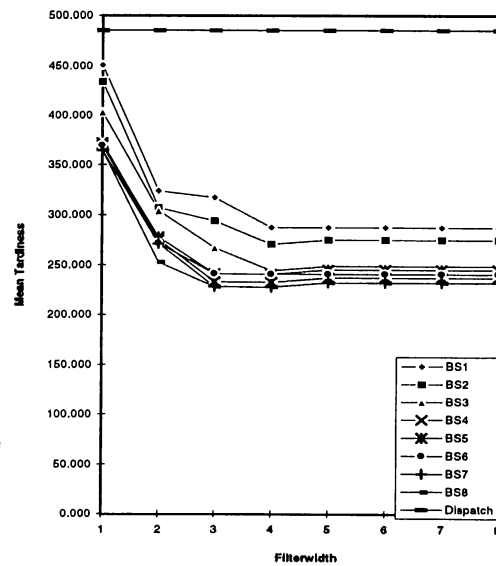
After determining the due date settings for the jobs, the performances of nondelay dispatching rules are measured in 10 test problems to find the appropriate local and global functions for the beam search algorithm. Five rules (SPT, EDD, LWR, MDD and MODD) are used in the experiments (see Table 3.6 for the complete description). Results indicate that the solution quality of EDD, MDD and SPT are comparable with each other and they are all better than the other rules with the average tardiness values of 517.9, 540.3 and 485.6, respectively. Similar to the makespan case, we determine most suitable branching scheme, evaluation functions and search parameters. Our computational experiments are carried out in two stages: we first investigate the branching scheme and then local evaluation functions and search parameters. Descriptions of these search methods are given in Table 3.7.

Table 3.6: Description of priority used in tardiness analysis

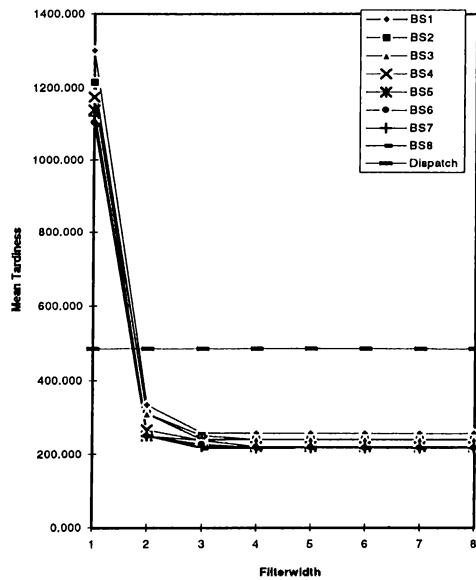
RULE	DESCRIPTION
SPT (Shortest Processing Time)	$SPT_{ij} = p_{ij}$
LWK (Least Work Remaining)	$LWR_{ij} = \sum_{q=j}^{m_i} p_{iq}$
EDD (Earliest Due Date)	$EDD_{ij} = duedate_i$
MOD (Modified Operational Due Date)	$MODD_{ij} = (duedate_i / \sum_{q=1}^{m_i} p_{iq}) \sum_{q=1}^j p_{iq}$
MD (Modified Due Date)	$MDD_{ij} = \max(duedate_i, t + \sum_{q=j}^{m_i} p_{iq})$



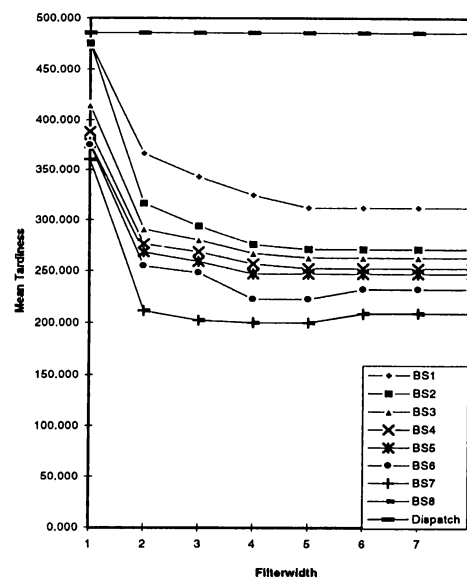
(a) Active Branching
Local: SPT
Global: Nondelay Dispatch(SPT)



(b) Nondelay Branching
Local: SPT
Global: Nondelay Dispatch(SPT)

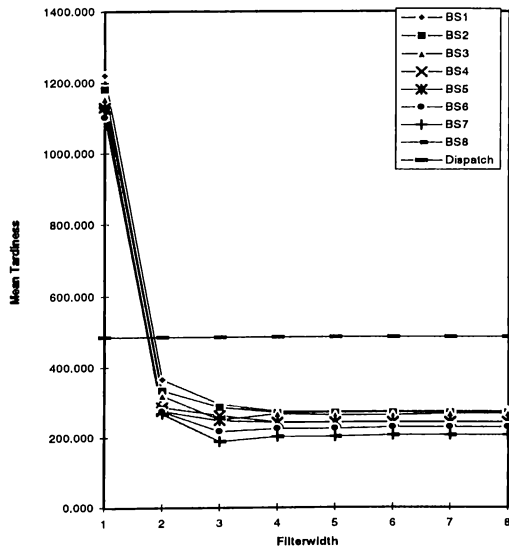


(c) Active Branching
Local: EDD
Global: Nondelay Dispatch(EDD)

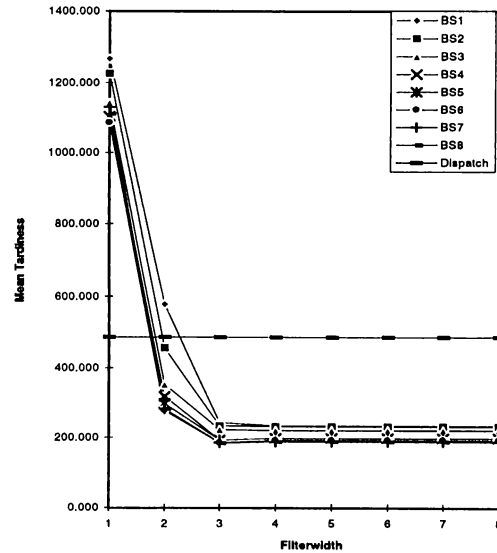


(d) Nondelay Branching
Local: EDD
Global: Nondelay Dispatch(EDD)

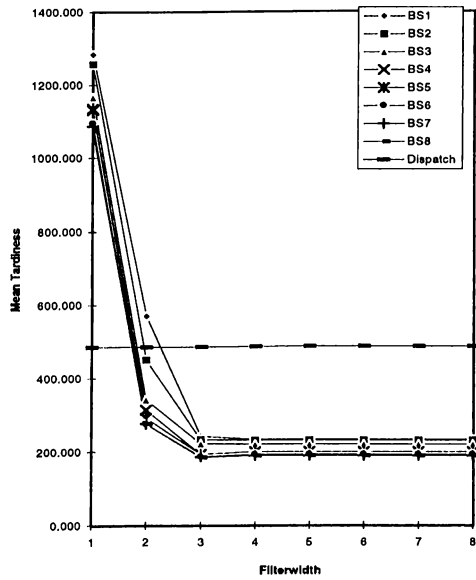
Figure 3.6: Mean tardiness vs filterwidth analysis



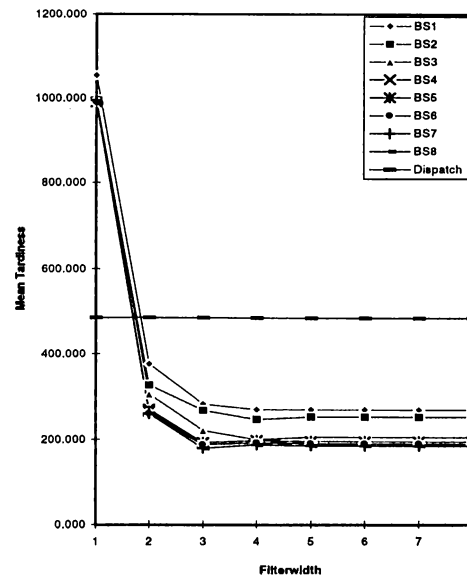
(a) Active Branching
Local: MDD
Global: Nondelay Dispatch(MDD)



(b) Active Branching
Local: MDD
Global: Nondelay Dispatch(SPT)



(c) Active Branching
Local: EDD
Global: Nondelay Dispatch(SPT)



(d) Active Branching
Local: MOD
Global: Nondelay Dispatch(SPT)

Figure 3.7: Mean tardiness vs filterwidth analysis

Active vs nondelay schedules

The results indicate that the performance of nondelay schedules is better than the active schedules for only small filter widths (Figure 3.6). However, after the filter width value of 3, the performance of the active generation method becomes better than the nondelay method (see Table A.2 in Appendix for the details of the experiments). This behavior is observed due to the fact that the number of active schedules generated by the beam search algorithm is greater than the number of nondelay schedules and hence, there is more chance to obtain better results by searching active schedules. For that reason, in contrast to the makespan case, the active scheduling scheme is used in later stages of this research.

Table 3.7: Search methods used in the first part of the experiments

Search Tree Representation	Local Evaluation Rule	Global Evaluation Function
Active	SPT	Nondelay SPT Dispatch Heuristic
Nondelay	SPT	Nondelay SPT Dispatch Heuristic
Active	EDD	Nondelay EDD Dispatch Heuristic
Nondelay	EDD	Nondelay EDD Dispatch Heuristic
Active	MDD	Nondelay MDD Dispatch Heuristic

Local and global evaluation functions

In the experiments, as a part of the active schedule generation scheme, MODD, SPT and EDD rules are used in both local and global evaluation functions. Results shows that SPT is better than the other rules. This result is consistent with results of the earlier studies reported by Kiran and Smith (1984) that SPT is one of the best priority rules in terms of all due date related measures.

However, due date based rules were expected to perform well for the mean tardiness criterion. Hence, the second set of experiments is carried out with EDD, MDD and MODD as the local evaluation functions. Due to better performance of the beam search with evaluation functions of SPT rule in the previous analysis, global estimation is again performed by this rule (see Table 3.8). The experimental results indicate that the due date based rules perform better than SPT (Figure 3.7). As given in Table A.3, due date based rules find promising nodes easily for small

filter widths. For that reason, their performances are considerably better than SPT. However, for the large beam and filter widths, they only perform slightly better than SPT. Overall, MODD displays the best performance, and hence is selected as the local evaluation function.

Filter and beam widths

As previously discussed in the makespan case, only a few nodes are expanded from a beam node in the job shop problem. Hence, increasing filter width does not improve the solution quality. It seems that the appropriate value is 5 (see also Figure 3.6 & 3.7). For the beam width parameter, the test results show that the beam width of 5 is also a proper value for the tardiness case when considering the CPU times and the quality of solutions (Figure 3.5). As a result, we decide to use active schedule generation method with local evaluation function of the MODD rule and global evaluation function of the SPT rule.

Table 3.8: Search methods used in the second part of the experiments

Search Tree Representation	Local Evaluation Rule	Global Evaluation Function
Active	MDD	Nondelay SPT Dispatch Heuristic
Active	EDD	Nondelay SPT Dispatch Heuristic
Active	MODD	Nondelay SPT Dispatch Heuristic

3.5.1 Computational Results

After determining the proper evaluation functions and parameter settings, the resulting beam search algorithm is applied to the 48 test problems described earlier in the paper. The detailed results of the algorithm and five dispatching rules are given in Table 3.9. Since we do not know optimal solutions in the tardiness case, we only compare the algorithm with the rules. Note that the percent deviations of dispatching rules from the solution of the proposed algorithm are also reported in that table.

The results indicate that, the proposed algorithm outperforms the rules for all problem instances. Among the dispatching rules, there is not a clear winner in

Table 3.9: Results of test problems for the Mean Tardiness Analysis

Prob.	Algor. b=5, f=5		EDD	LWR	MDD	MODD	SPT	EDD	LWR	MDD	MODD	SPT
	Sol.	CPU	Sol.	Sol.	Sol.	Sol.	Sol.	%Dev.	%Dev.	%Dev.	%Dev.	%Dev.
10*5												
LA01	97.0	3.4	118.2	118.2	118.2	147.9	135.6	21.8	21.8	21.8	52.4	39.7
LA02	62.5	3.9	103.9	105.8	107.5	132.4	89.0	66.2	69.2	72.0	111.8	42.4
LA03	70.0	3.4	98.7	97.0	97.0	132.4	85.9	41.0	38.5	38.5	89.1	22.7
LA04	86.8	8.7	133.3	133.8	135.8	151.4	130.8	53.5	54.1	56.4	74.4	50.7
LA05	95.8	3.6	106.9	112.2	106.1	144.5	118.0	11.60	17.1	10.7	50.8	23.1
Averages								38.9	40.2	39.9	75.7	35.7
15*5												
LA06	207.8	12.9	233.7	227.2	217.9	350.5	255.6	12.4	9.3	4.8	68.6	23.0
LA07	195.6	15.7	238.9	241.0	238.5	297.6	214.5	22.1	23.2	21.9	52.1	9.6
LA08	197.3	13.8	220.3	240.3	240.3	310.8	242.9	11.6	21.7	21.7	57.5	23.1
LA09	227.0	14.5	284.0	265.0	232.2	341.3	284.6	25.1	16.7	2.1	50.3	25.4
LA10	218.2	13.1	237.9	228.8	225.2	374.2	251.1	8.9	4.8	3.1	71.4	15.0
Averages								16.0	15.2	10.7	60.0	19.2
20*5												
LA11	343.7	36.2	365.5	353.6	341.0	564.4	389.8	6.3	2.8	-0.7	64.2	13.4
LA12	302.3	38.5	305.7	286.7	295.7	466.3	334.3	1.1	5.1	-2.1	54.2	10.6
LA13	329.8	39.3	352.9	335.7	340.5	515.1	371.0	7.0	1.8	3.2	56.1	12.5
LA14	398.6	38.5	403.3	370.4	377.5	566.0	411.0	1.1	-7.1	-5.3	41.9	3.1
LA15	386.1	37.2	396.6	419.7	412.9	566.6	429.1	2.7	8.7	6.9	46.7	11.1
Averages								3.6	0.2	0.4	52.7	10.2
10*10												
LA16	25.5	13.3	64.0	73.0	64.0	30.5	48.4	150.9	186.2	150.9	19.6	89.8
LA17	20.5	13.2	39.3	80.4	39.0	76.1	54.0	91.7	292.2	91.7	271.2	163.4
LA18	6.6	13.0	50.3	55.0	44.2	47.0	18.2	662.1	733.3	569.7	612.1	175.7
LA19	11.3	12.4	27.2	39.8	34.7	50.9	51.7	140.7	252.2	207.1	350.4	357.5
LA20	9.5	12.6	40.0	74.0	40.0	28.1	38.2	321.0	678.9	321.0	195.7	302.1
FT10	56.7	14.2	106.0	117.0	106.0	148.0	95.5	86.9	106.3	86.9	161.0	68.4
ABZ5	18.4	13.5	57.2	171.6	57.2	31.9	41.5	210.9	832.6	210.8	73.3	125.5
ABZ6	0.0	12.3	11.0	17.9	11.0	10.2	4.7	*	*	*	*	*
ORB1	113.2	15.6	194.7	157.1	205.4	199.8	241.0	72.0	38.8	81.4	76.5	112.9
ORB2	23.2	13.7	76.0	61.1	80.5	77.8	50.1	227.6	163.3	246.9	235.3	115.9
ORB3	105.8	15.5	135.1	156.3	136.2	298.9	184.6	27.7	47.7	28.7	182.5	74.5
ORB4	39.6	14.9	74.2	175.9	91.7	155.6	91.3	87.4	344.2	131.6	292.9	130.5
ORB5	40.3	13.6	78.7	97.4	78.7	96.5	87.5	95.3	141.7	95.3	139.4	117.1
Averages								181.2	318.1	185.2	217.5	152.8
15*10												
LA21	90.8	55.4	150.0	175.1	156.7	207.5	175.7	65.0	92.6	72.4	128.3	93.3
LA22	114.5	54.7	241.5	235.2	210.7	225.6	172.3	111.0	105.5	84.1	97.1	50.5
LA23	96.2	52.7	130.3	159.9	147.9	177.4	143.1	35.4	66.0	53.6	84.3	48.7
LA24	95.6	53.9	129.8	131.1	121.7	173.0	131.0	35.8	37.2	27.3	80.9	37.0
LA25	106.7	53.6	176.9	151.7	163.2	187.5	162.9	65.8	42.1	52.9	75.6	52.6
Averages								62.6	68.7	58.0	93.3	56.4
20*10												
LA26	225.6	156.0	345.0	241.1	288.3	344.4	250.1	52.9	6.9	27.8	52.6	10.8
LA27	226.3	148.4	303.8	292.4	286.1	358.0	271.3	34.3	29.2	26.4	58.2	18.9
LA28	212.4	155.9	301.4	330.8	227.3	410.2	342.3	41.9	55.7	30.5	93.1	61.1
LA29	216.1	150.7	270.2	276.2	266.9	443.4	269.9	25.0	27.8	23.5	105.1	24.9
LA30	233.8	155.2	396.2	372.9	382.9	425.4	337.6	69.4	59.5	63.7	81.9	44.4
Averages								44.7	35.8	34.4	78.2	32.2
30*10												
LA31	348.3	915.4	566.6	566.4	564.2	783.3	625.9	62.7	62.6	61.9	124.8	79.7
LA32	364.8	942.3	565.8	563.8	571.4	813.5	652.9	55.1	54.5	56.6	123.0	78.9
LA33	378.1	940.1	565.4	501.3	584.9	755.2	540.0	49.5	32.6	54.7	99.7	42.8
LA34	333.3	900.7	582.4	549.4	537.2	776.4	591.9	74.7	64.8	61.1	132.9	77.5
LA35	335.3	866.5	605.6	567.1	587.2	815.1	603.4	80.6	69.1	75.1	143.0	79.9
Averages								64.5	56.7	61.9	124.7	71.8

Table 1.9: (Cont'd) Results of test problems for the Mean Tardiness Analysis.

Prob.	Algor. b=5, f=5		EDD	LWR	MDD	MODD	SPT	EDD	LWR	MDD	MODD	SPT
	Sol.	CPU	Sol.	Sol.	Sol.	Sol.	Sol.	%Dev.	%Dev.	%Dev.	%Dev.	%Dev.
15*15												
LA36	33.53	123.03	113.93	115.20	113.9	142.2	140.7	239.7	243.5	239.7	324.1	319.7
LA37	26.8	125.4	48.0	109.4	54.1	172.1	118.6	79.3	308.2	101.9	542.2	342.8
LA38	27.1	126.3	128.6	136.6	130.0	87.4	53.1	374.2	403.5	379.1	222.1	95.8
LA39	24.5	121.9	87.3	114.3	87.3	93.3	76.1	256.0	366.0	256.0	280.4	210.3
LA40	43.6	125.5	92.1	123.2	106.4	126.2	88.9	110.9	182.1	143.8	188.9	103.6
Averages								212.0	300.7	224.1	311.6	214.7
Sol. : Solution								%Dev. : Percent deviation from algorithm's solution				

the experiments. Their relative performances vary for different problem types (i.e., square or rectangular). We also note that the mean tardiness values are low for the square type instances (i.e., number of jobs equals to number of machines). For that reason, differences due to small numbers results in high percent deviation of the rules for these problem instances.

3.6 Conclusion

In this chapter, we used the beam search to solve the classic job shop scheduling problem for the makespan and mean tardiness criteria. We also examined active and nondelay schedule generation schemes, different priority rules for both local and global evaluation functions and various values of beam and filter width parameters. According to our computational experience, we identified the proper settings of these parameters which can also used in future applications of this method. Our computational experiments also indicate that beam search is very good heuristic for the job shop problems.

As compared to other algorithms, the speed and the performance of a beam search based algorithm are manipulated by changing search parameters and evaluation functions. In addition, with beam search, it is also quite possible to generate partial schedules, since as the schedules are built progressively from the first operation to the last one in a forward direction. This is an important property of this search technique, because in a stochastic and dynamic environment where unexpected events can easily upset schedules, this feature of the beam search can be well utilized to generate partial schedules in a rolling horizon scheme. We should also note that,

coding of the algorithm is very simple and hence can be applied by practitioners.

One drawback of the beam search algorithm is imperfect assessing of the promise of nodes. As a result of this, the nodes that can lead to good solutions, are sometimes erroneously discarded. For the makespan problems, however, strong lower bounds are available in the literature. Hence, these lower bounds can be used as a part of evaluation functions to improve the performance of the proposed beam search algorithm in future studies.

Chapter 4

REACTIVE SCHEDULING

4.1 Introduction

Scheduling is an integral part of production systems planning for two important reasons. First, the schedule serves as an overall plan on which many other shop floor activities are based. Second, scheduling serves as a mechanism which optimize performance of the manufacturing facility. By properly planning the timing of shop-floor activities, performance criteria such as makespan, flow time, or mean tardiness can be optimized.

To achieve the goal of coordinated planning of system activities in real life a schedule is prepared in advance. However, a major drawback of precomputed schedules is that, after they are released for execution, the performance of the schedule degrades due to the random nature of shop floor conditions. Thus, it is desirable to respond to the unexpected changes to improve the performance of the schedule. The nature of response depends on the way schedules are generated or scheduling decisions are made.

There are mainly two types of schedule generation methods stated in the literature: *off-line scheduling* in which all available jobs are scheduled all at once for the entire horizon and *on-line scheduling* in which scheduling decisions are made

one at a time and when it is needed according to the changing system conditions. According to the on-line scheduling approach, the schedule is not determined all at once, but is constructed over time. Thus, by its nature, any disruption can automatically be handled as these local scheduling decisions are made. In the off-line approach, a priori generated predictive schedules should be revised because they are generally invalidated as a result of occurrences of stochastic disturbances in the system. The revisions can be done in various forms ranging from a repairment of the current schedule to generation of a new complete schedule for the remaining time period.

Among on-line scheduling methods, priority dispatching constitutes an important class. Therefore, in this study we use this scheme to generate on-line schedules. Generally speaking, schedules are easily generated by using priority rules, but solution quality is sacrificed due to the myopic nature of these decisions. A filtered beam search algorithm developed in the previous chapter is used as the off-line scheduling method. This method searches greater space and yields better solutions at a cost of computation time if the scheduling environment is static. However, the real life is dynamic and stochastic in nature. Hence, it is not clear if one dominates the other in actual shop floor conditions.

The performances of these two schedule generation schemes depend on the shop floor conditions. There are various factors affecting the system conditions and consequently characteristic of the scheduling problem. For instance, a classification can be made according to the job arrival information, in which case two types of scheduling problems are identified in the literature: *static problem* in which all jobs are assumed to be available simultaneously, whereas in *dynamic problem* jobs are expected to arrive at different random times, i.e., the set of jobs to be scheduled changes over time.

In addition, size of the system affects solvability of the problem. A shop with large number of machines and jobs can be viewed as more complex than its smaller counterpart. The scheduling problems of the complex systems are also expected to be more difficult in terms of computation times. However, it is not known how the relative performance of off & on-line methods is affected by the system size.

In addition, the load allocation in the system can also affect the performance of scheduling schemes. In real life, processing speed of machines are different and loads are unevenly allocated to the machines (i.e., there may be bottleneck work centers in the shop). Intuitively speaking, scheduling of uniformly loaded shops are expected to yield better results than their nonuniform counterparts. However, how different these two schedule generation methods perform in such systems is again an open question. Moreover, random events and interruptions that occur in the environment add stochasticity to the scheduling problem. Corrective actions need to be taken to deal with these unexpected changes for off-line scheduling scheme, thus scheduling problem become more complex in the stochastic environments.

The purpose of this chapter is first to analyze the effects of load allocation and complexity of system on the relative performances of on-line and off-line scheduling approaches in deterministic job shop environment. Secondly, frequency of rescheduling of the off-line method is analyzed in a job shop where random machine breakdowns are allowed to occur. The effect of machine breakdowns is examined again for different complexity levels and load allocations. Finally, we analyze partial scheduling, and its trade off between computational time requirements and solution quality in both deterministic and stochastic systems.

4.2 The Proposed Study

In this section, we first describe on-line and off-line scheduling approaches. Then we explain the job shop system and the environmental conditions including machine breakdown model and frequency of rescheduling.

4.2.1 Scheduling Methods

The off-line scheduling method used in this study is a heuristic algorithm which is based on the filtered beam search. In the previous chapter we have given a comprehensive analysis for the filtered beam search technique. Based on the

previous results, for the mean tardiness criterion, we use active schedule generation method with local evaluation function of MODD priority rule and global evaluation function of the SPT dispatching heuristic. For the makespan criterion, however, nondelay schedule generation method with local evaluation function of MWR priority rule and global evaluation function of the MWR dispatching heuristic are used. Recall that the values of beam and filter width search parameters were previously determined to be 5 for both performance criteria.

In the previous chapter, we have also evaluated five rules as on-line priority dispatching rules for the mean tardiness criterion and found that SPT performs better than the other rules. For the makespan criterion, however, we have noted that MWR outperforms the other rules. In our study, we use these rules in a classic nondelay priority rule scheme as stated in Kanet and Zhou (1993). According to this scheme, dispatching decisions are given, (1) whenever an operation becomes available for processing at a machine, if the machine is free, then engage the machine with the operation; otherwise, place the operation in the operation queue of the machine, and (2) whenever a machine becomes free, choose an operation from its queue according to the priority rule and engage the machine with the chosen operation.

4.2.2 Job Shop Environment

A classical job shop system is used in this study. The machines in the shop are capable of performing several types of operations but process only one at a time. Here, we assume the static case, in other words the jobs are available for processing at time zero. Job j consists of q_j number of operations (drawn from discrete uniform distribution between 5 and 15) in series and each operation is assigned to only one machine so that machines have equal probability to be visited. Number of operations are drawn from a discrete uniform distribution between 5 and 15. Besides, we have a priori known processing times, p_{ij} , (drawn from a discrete uniform distribution between 20 and 80) for operation i of job j . In the shop, preemption is not allowed and set-up times are included in the processing times. Number of machines and jobs existing in the shop determines the size of the problem. We consider the problem

with higher number of jobs or machines as more complex than the problem with smaller size. Since we will analyze the potential effects of problem complexity on the performance of the algorithms, we use four different sizes of the problem as described in Table 1.

Table 4.1: Sizes of the shop analyzed

Case	Number of jobs	Number of Machines
1	9	6
2	18	12
3	12	6
4	24	12

In order to obtain Case 2, the number of machines in Case 1 is increased with a factor of two. In both of the cases we want to have the same average work load per machine to preserve the consistency between the problems, because the mean tardiness and makespan objective criteria are the function of job completion times which is effected by the work load of the machines. Hence, in the above cases in which we increase the number of machines, number of jobs is appropriately increased so as to have equal average loads. We use the following expression to compute the average load per machine:

$$\frac{\text{Expected work load}}{\text{machine}} = \frac{E[\text{Process time}] * E[\text{Number of Operations}] * \text{Number of jobs}}{\text{Number of machines}}$$

In the larger shops, the job characteristics should be the same so that expected number of operations and expected processing time are kept unchanged. As the number of machines doubled, we increase the number of jobs by a factor of two in order to have the same average work load per machine.

This system complexity analysis is performed at two levels of job sizes, Cases 3 and 4 are generated (by utilizing the same argument), to observe the effects of having higher number of jobs in the same system.

The problem instances are generated so as to allocate the same expected work load to each machine. In other words, work load is uniformly allocated to the

machines. However, as stated previously, the second purpose of this study is to examine the effects of nonuniform work load. In order to have nonuniform work allocation, processing times of the previously generated problems are modified. By that way, we preserve the consistency between the problems. Keeping the total work load unchanged, in the shops with six workplaces, processing times of each machine are multiplied with the coefficient of 0.7, 0.8, 0.9, 1.1, 1.2, 1.3, respectively. In the shops with 12 machines, processing times on *two* machines are multiplied with the coefficients of 0.7, 0.8, etc. By this perturbation, speed of some machines are decreased to form bottleneck work centers and consequently to unbalance the load allocation in the system.

4.2.3 Machine Breakdowns

The most important source of randomness that interrupts the system operation in many manufacturing systems is the machine breakdowns or unscheduled down times. Therefore, we assume that the machines are subject to random breakdowns in our study. Busy times approach is used to model machine breakdowns (Law and Kelton, 1991). This method allows the machine to breakdown when it is busy. A random up time is generated from a busy time distribution and the machine operates until its total accumulated busy time reaches the end of this up-time. At the time of a failure, a repair time is generated and the machine is kept down during this period. After that, an up time will again be generated from the busy time distribution.

In the absence of real data, Law and Kelton recommend that busy time distribution is most likely to be Gamma Distribution with a shape parameter (α_b) of 0.7 and scale parameter (β_b) specified according to the experimental conditions. They also state that Gamma distribution with a shape parameter (α_d) of 1.4 is appropriate for the distribution of down times. In this proposed method, level of machine breakdown is measured by efficiency level (e) which gives the long run ratio of machine busy time to total busy and down time. The relationship among scale parameters, mean down time and efficiency is also formulated to specify the complete machine breakdown model. By this way, duration of each breakdown drawn from,

$$Gamma(\alpha_d = 1.4, \beta_d = \frac{\mu_d}{1.4})$$

and total busy time between two successive failures is drawn from,

$$Gamma(\alpha_b = 0.7, \beta_b = \mu_d * \frac{e}{0.7(1 - e)})$$

Here μ_d shows the mean duration of breakdown. In our experiments, we use 90% efficiency with 360 minutes of mean busy time and 40 minutes of mean downtime.

4.2.4 Frequency of Scheduling

In response to machine breakdowns or other interruptions in the system, we can either take no action and use the fixed sequence through the entire horizon (let the system recovers from the situation by itself) or reschedule the facility at every machine breakdown (continuous rescheduling). The former approach has a disadvantage that breakdowns alter the system status where alternative schedules might yield better solutions. The disadvantage of the latter approach is that in a large facility there are many events occur and too frequent schedule revision can increase the system nervousness. Also, computational requirements of the latter approach can be quite excessive.

Between these two extremes policies, periodic rescheduling is also studied in this chapter. In this approach, system is continuously monitored but the necessary actions are taken periodically by considering the unscheduled operations and the current system status. First issue in periodic scheduling is to determine an appropriate period length. *Fixed time* or *variable time interval* approaches can be used for this purpose. In this study, we use variable time interval method. According to this approach (Sabuncuoglu and Karabuk, 1997), the system is monitored at each time increment. If the cumulative processing time realized on all machines in the system reaches to a multiple of the specified length of the period, then rescheduling is triggered at this point. In the fixed time interval method, however, the period

length is solely determined by the absolute time. The variable time interval approach has some advantages over the fixed interval approach. First, since we use busy time method to determine machine breakdowns, probability of breakdowns is the same in each scheduling period. Also, this method divides the entire scheduling horizon into equal intervals in terms of processing times so that amount of schedule executed is the same in each interval. By this way, we can measure the level of responsiveness of the rescheduling without being affected by the system load or any other scheduling factor.

The following expression is used to calculate the rescheduling period.

$$t_s = TP/f$$

where

t_s : amount of processing times between two consecutive rescheduling points.

TP : total processing time of all jobs to be scheduled.

f : number of rescheduling points in a given scheduling horizon (frequency of scheduling)

We use ten levels of frequency of scheduling in our experiments. These are 0, 2, 4, 6, 8, 10, 12, 14, 16, 1000. Here, 0 corresponds to no rescheduling (fixed sequencing) case, in which, a schedule is generated at the beginning of the horizon and the sequence determined in this preschedule is used through the scheduling horizon regardless of any future event. If a machine breakdown event occurs, then the unexecuted operations on this machine are simply right shifted for the duration of down time. Another extreme level, 1000, represents the continuous rescheduling. In this case, rescheduling frequency is so high that reschedule is triggered at any event that alters the system status. Between these two extreme cases, eight levels of periodic scheduling are analyzed. For instance, level 4 results in the schedule to be revised approximately four times during the makespan of the schedule. However, if there is not any machine breakdown during the period, schedule is not revised. That is, the current schedule is still used up to the next scheduling period.

4.3 Computational Results

In this section the algorithms (on-line and off-line) are applied to the scheduling problems of different sizes. For each system size, uniform and nonuniform loading are also considered. In the experiments, ten randomly generated problems are used at each system condition. The averages of the results based on ten replications are presented in the tables. The mean tardiness and makespan criteria are used to evaluate the performance of the algorithms. In the first part of the section, the analysis is performed in the deterministic environment, then the effects of machine breakdowns and frequency of scheduling are examined in the stochastic environment. Also, the role of partial scheduling is discussed at the end of this section in both stochastic and deterministic environments

4.3.1 Deterministic Environment

In this first part, we assume that the machines in the system are always available. Simulation experiments are conducted to examine the effects of system size and load allocation on the performance of the algorithms in this deterministic environment. The analysis is performed for both the mean tardiness and makespan criteria as discussed in the subsequent sections. The results of experiments are presented in Table 4.2 and Table 4.3 for the mean tardiness and makespan respectively. In these tables, % Deviation 1 stands for the difference between the beam search algorithm and the dispatching rule solutions. Likewise, % Deviation 2 is the deviation between the solutions of each scheduling method in uniform and nonuniform systems. Paired t-test is used to determine the significance of the differences. In the tables, ‘*’ refers that respective term is statistically significant with a confidence interval of 95 %.

Mean Tardiness Analysis

Simulation experiments are first performed for the mean tardiness criteria. The results of the experiments are given in Table 4.2. As seen in the table, all the differences are statistically significant according to the paired t-test.

In the mean tardiness case, the first observation is that the beam search method always performs better than the dispatching rule (SPT) under all the experimental conditions. This observation confirms the results of the previous chapter.

We also note that in the large systems, mean tardiness values are higher than the small ones (i.e., In Table 2, the off-line algorithm yields the mean tardiness of 76.32 and 90.76 for the 9 jobs 6 machines and 18 jobs 12 machines, respectively). We did not expect this result because we determine the number of jobs so as to have equal work load per machine. However, we measure that, in the large systems, utilization of machines is generally lower than the small ones (0.677 for 9 jobs 6 machines shop, 0.629 for 18 jobs 12 machines shop) and consequently in the large systems, completion times of jobs are slightly longer than those of the small systems.

Table 4.2: Performances of algorithms in deterministic environment.

for mean tardiness for criteria				
	Algorithm	Dispatch	% Deviation1	Absolute Dif.
9 jobs 6 machines				
uniform	76.32	123.70	62.08*	47.38*
nonuniform	127.55	187.68	47.15*	60.13*
% Deviation2	67.13*	51.72*		
18 jobs 12 machines				
uniform	90.76	132.58	46.09*	41.83*
nonuniform	119.38	166.93	39.83*	47.55*
% Deviation2	31.53*	25.91*		
12 jobs 6 machines				
uniform	171.32	231.37	35.05*	60.05*
nonuniform	220.38	291.68	32.35*	71.29*
% Deviation2	28.64*	26.07*		
24 jobs 12 machines				
uniform	212.61	279.29	31.36*	66.68*
nonuniform	245.35	317.78	29.52*	72.43*
% Deviation2	15.40*	13.78*		

Additionally we observe that, the difference between the performances of the beam search algorithm and the dispatching rule varies for different sizes of the shops. It seems that the system complexity does not considerably affect the relative performance of on and off line schedule generation methods. Because as the number of machines increases, (9 jobs 6 machine vs 18 jobs 12 machines) we do not observe a regular trend in terms of increases or decreases in the *absolute differences*. However,

percent deviations decrease as the system size gets larger, since the large scale of mean tardiness values result small percent deviations.

In contrast, as the number of jobs in the system increases, (9 jobs 6 machines vs 12 jobs 6 machines) we observe an increasing trend in the absolute differences. In other words the proposed algorithm performs better than the dispatching rule in the crowded systems. This is due to the fact that in the system with more jobs, there are larger number of alternative schedules to search through by the off-line algorithm.

Another important observation is that in the system with nonuniform load allocation, the mean tardiness value is higher than the uniform case. This observation makes sense, because in nonuniform facilities, there are bottleneck machines that delay the completion time of the jobs. Moreover, the performance of the algorithm is effected more severely than the dispatching rule in the nonuniform environment (Table 2. % Deviation2 values). The degradation is less significant in larger shops (67.13% for 9 jobs 6 machines, 15.40% 24 jobs 12 machines).

In addition, the absolute difference between the algorithm and the rule gets larger for all the shop size combinations in the nonuniformly loaded shops. In the pilot runs we noticed that, if the variability of processing times is high, then the off-line algorithm performs better than the dispatching rules. In order to obtain an unbalanced system, we change (increase or decrease) the processing times by multiplying certain coefficients, and consequently increase the variability in the processing times. Based on the observation, it is expected that in the nonuniform shop floor conditions the proposed algorithm performs relatively better than the rules.

We also measure the CPU time requirements of the algorithms. As expected, the on-line algorithm (i.e., dispatching rule) is very fast, because it generates schedules by only evaluating the priority function values of operations. For instance, CPU times are on the average 10^{-2} seconds for 9 jobs 6 machines system and $6 * 10^{-2}$ seconds for 24 jobs 12 machines system. As these numbers show from the smallest to largest system, the CPU time requirements only change with a factor of 6. On the other hand, the CPU time requirements of the off-line algorithm, vary significantly

according to the system size. Average CPU times are 12.2, 31.5, 98.7, 260.6 sec. for 9 jobs 6 machines, 12 jobs 6 machines, 18 jobs 12 machines, 24 jobs 12 machines, respectively. According to these statistics, we can state that as the number of jobs is increased with a factor of $4/3$, required CPU times increases approximately 2.5 times. This observation is consistent with the $O(n^3)$ complexity of the beam search algorithm. As seen from the results, the CPU time requirements of off and on-line algorithms are not comparable ($6 * 10^{-2}$ seconds for dispatching rule and 260.6 seconds for the beam search method). However, percent difference between the solutions of algorithms is 31.36% (see Table 4.2) in the largest system.

Makespan Analysis

The same experiments are also performed for the makespan criteria and the results of this experiment are given in Table 4.3. As seen in the table, all the difference terms are statistically significant with the 95 % confidence interval, according to the paired t-test.

The results of the makespan case are very similar to the those of the mean tardiness. However, in the mean tardiness case we have observed that performance of the algorithm is affected more than the dispatching rule in the nonuniform environment whereas in the makespan case we note that the deterioration of the solutions due to nonuniform load allocation is almost equal for both the algorithm and the dispatching rule (see % Deviation 2 values in Table 4.3). Moreover in the mean tardiness case we have also observed that the absolute difference between the solutions of the algorithm and the rule gets larger in the nonuniformly loaded shops for all the shop size combinations. In the makespan case this observation is also valid for three shop size combinations, but in the system with 12 jobs and 6 machines the absolute differences between the solutions of scheduling methods in uniform and nonuniform environments is almost same (see Absolute Difference values in Table 4.3). This is simply due to the randomness involved in the problem instances.

In the makespan case the percent differences between the algorithm and the dispatching rule solutions are small (% Deviation1 in Table 4.3) compared to the percentages in the mean tardiness case (% Deviation2 in Table 4.2). Because the characteristic of the makespan measure is different from the mean tardiness measure.

It is based on the completion time of only one job and completion time of the other jobs are not taken into account. In the job shop problem without any flexibility, there are limited number of nondelay schedules. Thus, schedule of the job that gives the maximum completion time does not too much vary among the alternative schedules, and hence the algorithm and the dispatching rule produce similar schedules with close makespan values. Consequently the differences between the algorithm and dispatching rule solutions are not too large. In addition, the scale of the makespan values are considerably higher than the scale of the mean tardiness values and even larger absolute differences give low percent differences for the makespan case.

Table 4.3: Performances of algorithms in deterministic environment.
for makespan criteria

	Algorithm	Dispatch	% Deviation1	Absolute Dif.
9 jobs 6 machines				
uniform	1048.2	1109.2	5.82*	61.00*
nonuniform	1203.5	1284.6	6.74*	81.10*
% Deviation2	14.82*	15.81*		
18 jobs 12 machines				
uniform	1114.3	1171.8	5.16*	57.50*
nonuniform	1225.7	1310.7	6.93*	85.00*
% Deviation2	10.00*	11.85*		
12 jobs 6 machines				
uniform	1237.9	1315.4	6.26*	77.50*
nonuniform	1361.8	1438.7	5.65*	76.90*
% Deviation2	10.01*	9.37*		
24 jobs 12 machines				
uniform	1437.4	1514.2	5.34*	76.80*
nonuniform	1590.2	1679.3	5.60*	89.10*
% Deviation2	10.63*	10.90*		

4.3.2 Stochastic Environment

In this section, we now assume that the machines are subject to random breakdowns. The breakdowns occur according to the busy time approach as described in the previous section. In the simulation experiments for the machine breakdowns, we use 90% efficiency with 360 time unit of mean busy time and 40 time units of mean downtime. In the subsequent analysis, we first employ *no response* policy for off-line

scheduling method, and study the effects of machine breakdowns on the relative performance of off-line and on-line algorithms for various system complexity and uniformity combinations. Then we use the *periodic response* policy and measure the effects of frequency of rescheduling extensively.

No response Policy

In this section we analyze the deterioration on the solution quality of the algorithms due to machine breakdowns. Recall that the on-line algorithms takes machine breakdowns into account while constructing the schedules. In this section we use no response option for the off-line algorithm. As mentioned before, in this response policy, a schedule is generated at the beginning and the sequence determined in this schedule is used through out the scheduling horizon regardless of any future event. In the experiments we again use various problem sizes for uniformly and uniformly loaded systems. The performance of the schedules are measured for both the mean tardiness and makespan criteria as discussed in the subsequent sections. Paired t-test is used to determine the significance of the differences. In the tables (Tables 4.4 through 4.7), ‘*’ refers that respective term is statistically significant with a confidence interval of 95 %.

Mean Tardiness Analysis

The results of the simulation experiments (the mean tardiness values of off and on-line algorithms for both deterministic and stochastic environments) are given in Table 4.4. In this table, % Deviation 1 differences between solutions with respect to deterministic and stochastic environments. Similarly, % Deviation 2 corresponds to the deviation between the algorithm and the dispatching rule solutions. As seen from the table, all the difference terms are statistically significant according the paired t-test.

The most important observation is that performance of the on-line scheduling algorithm degrades less than that of the off-line algorithm in the stochastic environment (% Deviation 2 values are smaller for dispatching rules in all the problem sets). The same observation was also made in Yamamoto and Nof (1985)

and Sabuncuoglu and Karabuk (1997). In the above studies, the authors pointed out that as the number of disruptions increases, the performance of the algorithm with the fixed sequencing method and the on-line method in the form of dispatching rule get closer to each other.

The number of breakdowns in the shop with 12 machines is approximately two times larger than the one with 6 machines (there are on the average 7.2, 16, 10 and 18.9 machine breakdowns in the shops with 9 jobs 6 machines, 18 jobs 12 machines, 12 jobs 6 machines and 24 jobs and 12 machines, respectively). In a system with more breakdowns, the solution quality of the algorithm is expected to deteriorate more significantly. However, note that absolute differences between deterministic and stochastic cases slightly increase in the larger shops. This is mainly due to the slack in the schedules. In the systems with 12 machines, there are 2 times more slacks than the shop with 6 machines. Therefore, amount of deterioration is approximately the same in the both systems (42.58 for 9 jobs 6 machines, 48.19 for 18 jobs 12 machines).

However, we observe that, as there are more jobs in the system, absolute differences between the solution of scheduling methods in the deterministic and stochastic systems become more significant (42.58 for 9 jobs 6 machines, 71.20 for 12 jobs 6 machines). Because, as the number of jobs increases, the system become more crowded and the average utilization of the machines gets higher (0.677 for 9 jobs, 0.728 for 12 jobs). Thus, machine breakdowns negatively affect the performance of schedules considerably in larger systems. When we compare the percent differences between deterministic and stochastic cases, we observe that differences get smaller as the system size gets larger. Because, the scale of the mean tardiness is large for more complex systems, and even larger absolute differences yield low percent differences.

In the nonuniform systems, we also notice the performance changes in the stochastic environment in comparison to the deterministic environment. As given in Table 4.5, the same pattern is observed as in the uniform systems. The absolute differences between deterministic and stochastic cases are close to each other in both small and large systems. However, as the number of jobs increases, the system

Table 4.4: Performances of algorithms in stochastic environment for mean tardiness criteria (Uniform Case)

	Deterministic	Stochastic	% Deviation1	Absolute Dif.
9 jobs 6 machines				
Algorithm	76.32	118.90	55.79*	42.58*
Dispatch	123.70	157.22	27.10*	33.52*
% Deviation2	62.08*	32.23*		
18 jobs 12 machines				
Algorithm	90.76	138.94	52.10*	48.19*
Dispatch	132.58	168.12	26.80*	35.54*
% Deviation2	46.09*	21.00*		
12 jobs 6 machines				
Algorithm	171.32	242.52	41.56*	71.20*
Dispatch	231.37	280.14	21.08*	48.78*
% Deviation2	35.05*	15.51*		
24 jobs 12 machines				
Algorithm	212.61	286.78	34.89*	74.17*
Dispatch	279.29	335.35	20.07*	56.07*
% Deviation2	31.36*	16.94*		

performance is affected by the machine breakdowns more significantly like in the uniform cases.

However, we note that performances of the schedules are not affected as much as in the uniformly loaded systems. Because, in the nonuniform systems, the utilization of the machines are low due to the reason of bottleneck machines extend the scheduling horizon. In other words, there is excessive slack in the system, and machines breakdowns are absorbed by this slack and consequently they do not seriously degrade the scheduling performance.

Makespan Analysis

We perform the same simulation experiments also for the makespan criteria. The results are given in Table 4.6 and Table 4.7 for uniform and nonuniform environments, respectively. As seen in the table, the difference between the off-line and on-line algorithms in the stochastic environment for the 12 jobs 6 machines and 24 jobs and 12 machines systems are not statistically significant. This result of the test confirms our observations in the mean tardiness analysis. Recall that,

Table 4.5: Performances of algorithms in stochastic environment.
for mean tardiness criteria (Nonuniform Case).

	Deterministic	Stochastic	% Deviation1	Absolute Dif.
9 jobs 6 machines				
Algorithm	127.55	169.97	33.26*	42.42*
Dispatch	187.68	221.29	17.91*	33.61*
% Deviation2	47.15*	30.20*		
18 jobs 12 machines				
Algorithm	119.38	172.08	44.15*	52.70*
Dispatch	166.93	204.20	22.33*	37.27*
% Deviation2	39.83*	18.66*		
12 jobs 6 machines				
Algorithm	220.38	288.26	30.80*	67.88*
Dispatch	291.68	339.30	16.33*	47.63*
% Deviation2	32.35*	17.71*		
24 jobs 12 machines				
Algorithm	245.35	312.98	27.57*	67.64*
Dispatch	317.78	370.53	16.60*	52.75*
% Deviation2	29.52*	18.39*		

the performance of the off-line algorithm worsens more than the on-line algorithm in the stochastic environment and the deterioration is even more significant in the systems with more jobs. Except for these two terms, the rest of the difference terms are statistically significant according to paired T test.

In the makespan case, the patterns of the results are very similar to the mean tardiness case. There is only one erratic result in this analysis. According to the results of the mean tardiness case, we are expecting that the deterioration on the solutions, due to machine breakdown, in the nonuniform environment is less than that of the uniform environment. However, in the makespan case, we note that the solution of algorithm, for the system with 9 jobs and 6 machines, deteriorates more in the nonuniform environment. Other than this result, all the observations and discussions made in the mean tardiness are also valid for the makespan case.

Table 4.6: Performances of algorithms in stochastic environment for makespan criteria (Uniform Case)

	Deterministic	Stochastic	% Deviation1	Absolute Dif.
9 jobs 6 machines				
Algorithm	1048.2	1119.2	6.77*	71.00*
Dispatch	1109.2	1180.7	6.45*	71.50*
% Deviation2	5.82*	5.49*		
18 jobs 12 machines				
Algorithm	1114.3	1218.8	8.84*	104.50*
Dispatch	1171.8	1253.9	7.01*	82.10*
% Deviation2	5.16*	3.39*		
12 jobs 6 machines				
Algorithm	1237.9	1372.3	10.86*	134.40*
Dispatch	1315.4	1400.5	6.47*	85.10*
% Deviation2	6.26*	2.05		
24 jobs 12 machines				
Algorithm	1437.4	1562.8	8.72*	125.40*
Dispatch	1514.2	1611.1	6.4*	96.90*
% Deviation2	5.34*	3.09		

Periodic Response Policy

The on-line scheduling algorithm in the form of a dispatching policy takes machine breakdowns into consideration as they occur. However, the system should be revised for the off-line algorithm in order to recover from the negative effects of the interruptions. In the periodic response policy, these revisions are made periodically and all the unexecuted operations are rescheduled at the beginning of the next period. In our simulation experiments, we examine various levels of scheduling frequency (i.e., scheduling period) for different problem sizes for uniformly and nonuniformly loaded systems using the mean tardiness and the makespan performance measures as discussed in the following sections.

Mean Tardiness Analysis

The simulation experiments are first performed for the mean tardiness criterion and the results of the experiments for the uniformly loaded environment are given in Figures 4.1a & 4.1b (numerical results are also given Tables B.1 in the appendix).

Table 4.7: Performances of algorithms in stochastic environment.
for makespan criteria (Nonuniform Case).

	Deterministic	Stochastic	% Deviation1	Absolute Dif.
9 jobs 6 machines				
Algorithm	1203.5	1292.4	7.39*	88.90*
Dispatch	1284.6	1350.4	5.12*	65.80*
% Deviation2	6.74*	4.49*		
18 jobs 12 machines				
Algorithm	1225.7	1317.3	7.47*	91.60*
Dispatch	1310.7	1374.3	4.85*	63.60*
% Deviation2	6.93*	4.33*		
12 jobs 6 machines				
Algorithm	1361.8	1472.1	8.10*	110.30*
Dispatch	1438.7	1521.7	5.77*	83.00*
% Deviation2	5.65*	3.37*		
24 jobs 12 machines				
Algorithm	1590.2	1714.2	7.80*	124.00*
Dispatch	1679.3	1740.4	3.64*	61.10*
% Deviation2	5.60*	1.53		

The graphs in the Figure display the behavior of the mean tardiness as a function of scheduling frequency.

The first observation drawn from the results is that, the performance of the off-line algorithm improves as the frequency of scheduling increases. However, as seen from Figure 4.1b, the levels of improvement are not always large. Because, the static job shop system used in this study does not have any routing or sequence flexibilities, and consequently the set of active schedules is not very large. In such a system, the beam search based algorithm produces very similar sequences at each rescheduling point and the performance of the algorithm does not improve considerably. Even at some frequency levels, the mean tardiness value is slightly worse than that of less frequent. We observe such behavior due to the randomness involved in the test problems. Therefore, the solution quality is not guaranteed to be always better for the higher values of scheduling frequency.

In addition, as depicted in Figure 4.1b, the segmented line for the small system is flatter than the larger system. Because in large systems, we have more number of

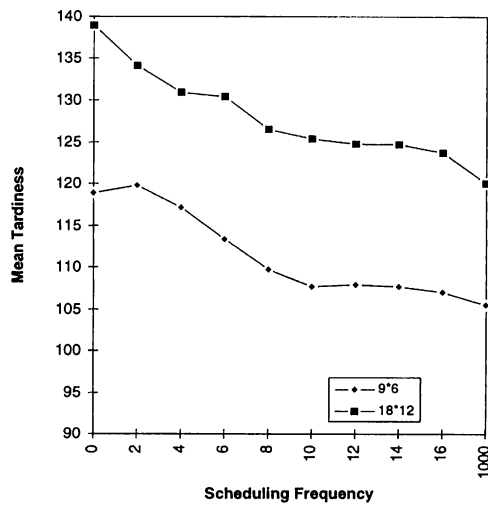
breakdowns in the scheduling horizon and in the rescheduling periods. Therefore, in these more disrupted systems, rescheduling helps more to recover from these unexpected events. This observation is less clear in Figure 4.1a, but still the level of improvement is slightly better for the large system (12.7 % for 9 jobs 6 machines case, 15.8 for 18 jobs 12 machines case).

The effects of rescheduling is more erratic in the nonuniform systems. The results are displayed in Figures 4.1c & 4.1d (the numerical results are also given in Table B.2 in the appendix). The unexpected ups and downs of mean tardiness values in these systems may be the result of high level of variability in processing times. As we compare the graphs in Figure 4.1, we see that the effect of rescheduling is less significant in nonuniformly loaded systems. Because, as mentioned before, the excessive amount of slack in nonuniform environment absorbs the negative effects of machine breakdowns. Therefore, the effect of machine breakdowns is less disruptive in such systems.

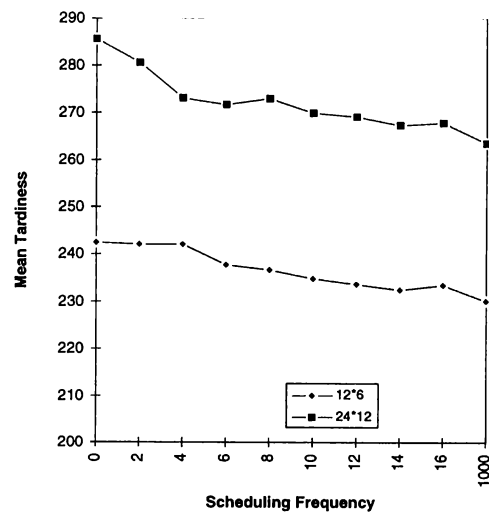
In addition, as seen in Figure 4.1a & 4.1b for the low scheduling frequency levels, the improvement in solutions is clearer. However, after a certain point, increasing frequency does not significantly improve the solution quality. On the other hand, as seen from Figure 4.2, the CPU time requirements get considerably larger as the frequency of scheduling increases (CPU times values are also given in Table B.3 in the appendix). Frequent rescheduling also increases the system nervousness. Thus, after trading off the CPU time and improvement gained, we propose not to reschedule the system too frequently.

Makespan Analysis

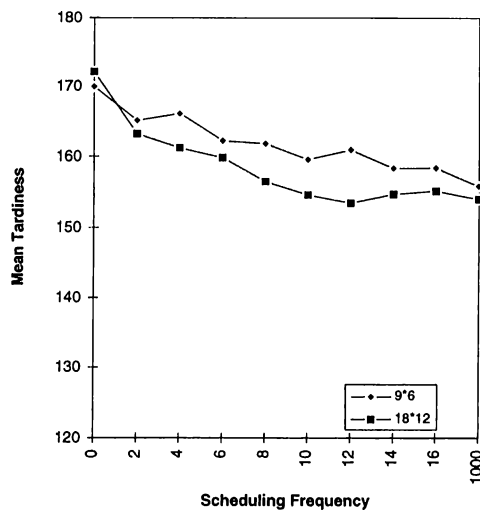
The same set of simulation experiments are also performed for the makespan criterion. The changes of the makespan as a function of scheduling frequency for the uniformly loaded environment are depicted in Figure 4.3a & 4.3b and (the results are also given in Table B.4 in the appendix). As seen in the figures, the makespan do not significantly improves as the scheduling frequency increases. Because, the number of alternative schedules generated by nondelay scheme are not too high for the job shop problem analyzed in this study. Thus at each scheduling point, very similar schedules are determined with close makespan measures.



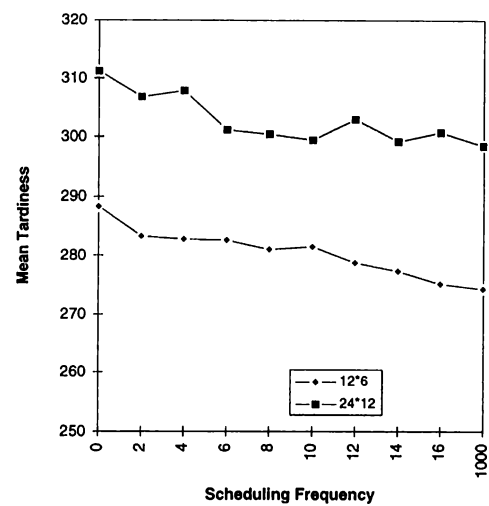
a) Scheduling Frequency vs Mean Tardiness Uniform Case



b) Scheduling Frequency vs Mean Tardiness Uniform Case



c) Scheduling Frequency vs Mean Tardiness Nonuniform Case



d) Scheduling Frequency vs Mean Tardiness Nonuniform Case

Figure 4.1: Interactions between scheduling frequency and mean tardiness

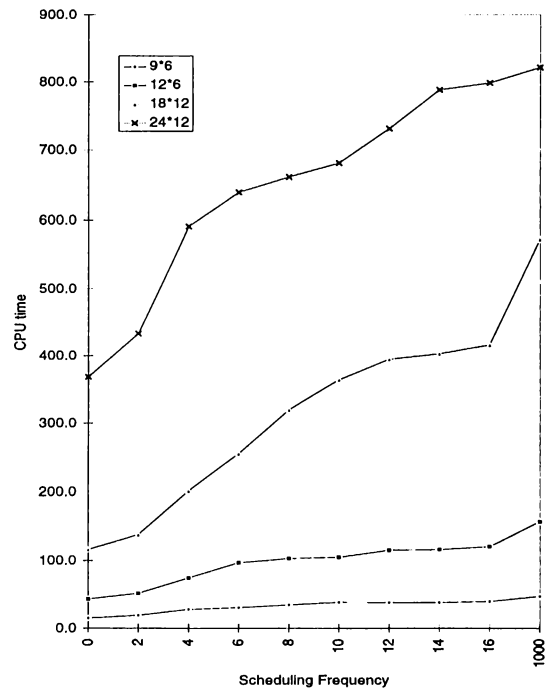
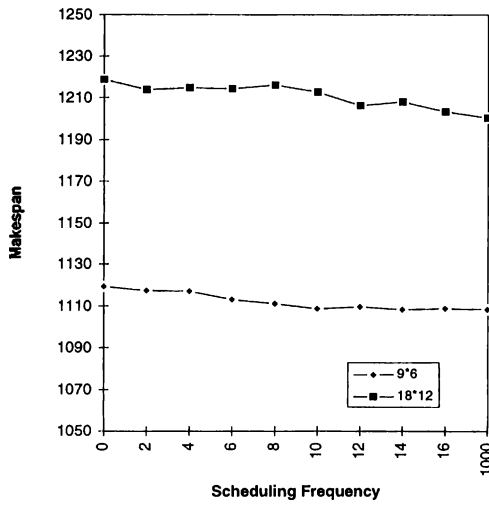


Figure 4.2: CPU time vs scheduling frequency

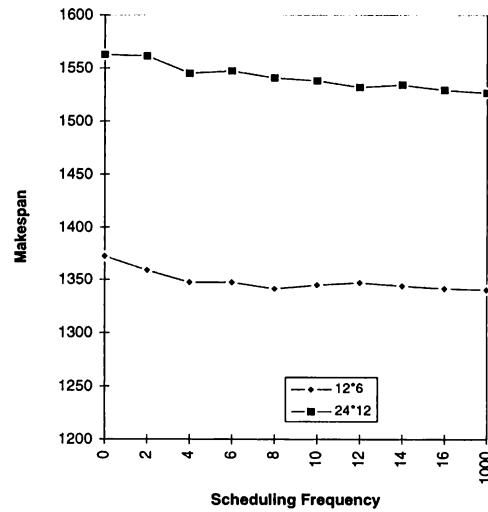
In the makespan case, except for the insignificant improvement of objective function in response to scheduling frequency, other observations are consistent with the mean tardiness analysis. Here we again observe that the changes in makespan values for small systems are relatively less than those of large systems and the effects of rescheduling is also more erratic in the nonuniform system (Figure 4.3c & 4.3d).

4.3.3 Partial Scheduling

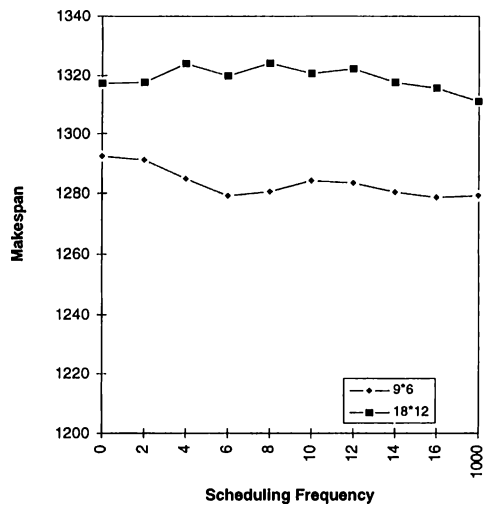
Beam search is a constructive type heuristic algorithm. Operations are scheduled sequentially in forward direction by using the ideas of branch and bound method. This feature of beam search algorithms allows us to generate partial schedules. We define *partial schedule* as the one that does not schedule all the available operations but rather a subset of schedulable operations in the system.



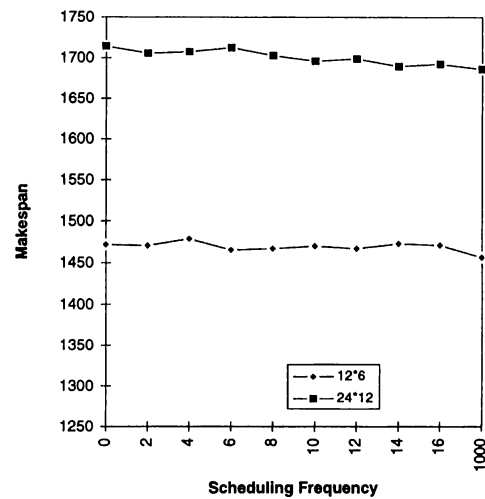
a) Scheduling Frequency vs Makespan Uniform Case



b) Scheduling Frequency vs Makespan Uniform Case



c) Scheduling Frequency vs Makespan Nonuniform Case



d) Scheduling Frequency vs Makespan Nonuniform Case

Figure 4.3: Interactions between scheduling frequency and makespan values

The length of the partial schedule is important, since it affects the solution quality and CPU time requirements. This length can be measured by either in terms of clock time or number of operations. But we prefer the latter approach in our study in order to be consistent with the definition of the period length. According to the number of operations approach, a partial schedule of half length means that half of the operations are scheduled at a time.

Previously, we have not encountered any study on the partial scheduling in the job shop scheduling literature. In our study, we implement the partial scheduling as a part of the periodic response policy, which is discussed in the previous section. Before the implementation of the partial scheduling, we determine the partial schedule length. At this stage, scheduling frequency should be taken into account, because the partial schedule length should at least be enough to cover the period length (i.e., total processing time of the scheduled operations should at least be equal to the period length). This partial scheduling with the periodic response system is implemented as a rolling horizon control scheme. At each scheduling point, a partial schedule with certain length is generated and used until the next decision point at which a new partial schedule is again generated according to current system status. During a period, if all the scheduled operations are executed and the next scheduling point is not reached (i.e., the length of the schedule is not enough to cover the period), the scheduling scheme is triggered to generate a new partial schedule at this point in time. The resulting partial scheduling with in the periodic response policy will be simply called as partial scheduling for the rest of the thesis.

To perform the simulation experiments, we created the same system conditions (system size and uniform vs nonuniform) used before in the periodic response policy. In addition, we analyze the effects of partial scheduling for two levels of scheduling frequency. We choose one level for low frequency (4) and one level for high frequency (14). Recall that for the high frequency level, the period lengths are shorter than those of low frequency. Therefore, we can test more partial schedule length alternatives for the scheduling frequency level of 14 than the level 4. Because at the scheduling frequency of level 14, even short partial schedule lengths will be enough to cover the periods. Therefore, we use $1/10$, $1/8$, $1/6$, $1/4$, $1/2$, 1 as the partial schedule lengths for the scheduling frequency level of 14. Since the period

lengths are long at the scheduling frequency of level 4 we only use $1/3$, $1/2$, 1 partial schedule lengths for that level. Here, $1/3$ means that at each scheduling point, $1/3$ of the total number of operations are scheduled and needless to say, 1 refers to generating complete schedules.

Mean Tardiness Analysis

The effects of partial scheduling are first measured for the mean tardiness performance criterion. The results of the analysis for the uniform case is shown in Figure 4.4 (the results are also tabulated in Table B.6 and Table B.7 in the appendix). These graphs show the changes in mean tardiness and CPU times as the function of partial schedule length for both the low and high frequency levels.

The first observation from these graphs is that as the length of partial schedule increases the quality of the schedules improves regardless of scheduling frequency. Because in the short lengths myopic decisions are taken by the partial schedules and this negatively affects the quality of schedules. Besides, during the implementation of partial schedules, additional idle times need to be inserted in the schedules. Because, we generate the partial schedules by using a subset of all operations. These scheduled operations may not be homogeneously assigned to the machines, for instance there may be 10 operations allocated on one machine whereas only 5 operations allocated on the other. During the implementation of a partial schedule, if a machine does not have any more operations to schedule, we should insert an idle time until the next scheduling point. This inserted idle time worsens the performance of the partial scheduling method especially for small scheduling lengths.

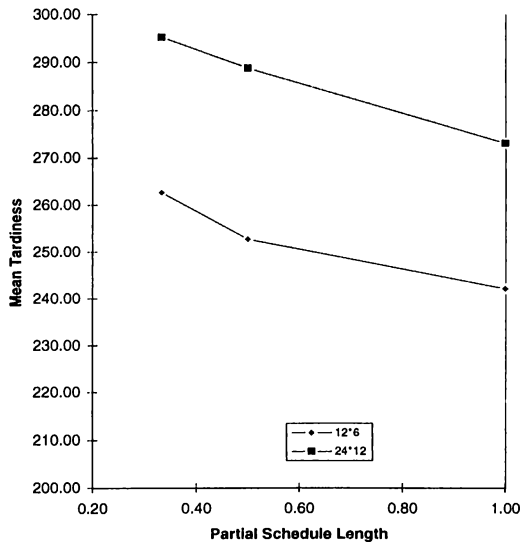
From the results, we also note that the mean tardiness does not linearly change with the partial schedule length. We have analyzed the effects of partial scheduling for two scheduling levels. However, the behavior of the mean tardiness is more informative at the scheduling frequency level of 14 (see Figure 4.c). Because at this level, we have evaluated more partial schedule length alternatives. From Figure 4.4c, it can be noted that for short partial lengths (i.e., the lengths of $1/10$ and $1/8$), the amount of deterioration in the schedule quality is negligible. Because the number of operations in the partial schedules are close to each other for the lengths of $1/10$ and $1/8$. Thus, the proposed partial scheduling system produce similar schedules with

comparable performances. However, as the partial schedule length increases (i.e., the lengths of $1/6$ and $1/4$), there are significant improvements in the mean tardiness values. The reason for that is the reduction of inserted idle times due to the increase in partial schedule length. Finally, we observe that marginal improvement in the mean tardiness become smaller for long partial schedule lengths (i.e., the lengths of $1/2$ and 1). Because for the long partial schedule lengths, we have small amount of inserted idle times which slightly worsens the performance of the schedules. In summary, we can conclude that mean tardiness change significantly for moderate level of partial schedule lengths whereas changes are insignificant for short and very large partial schedule lengths.

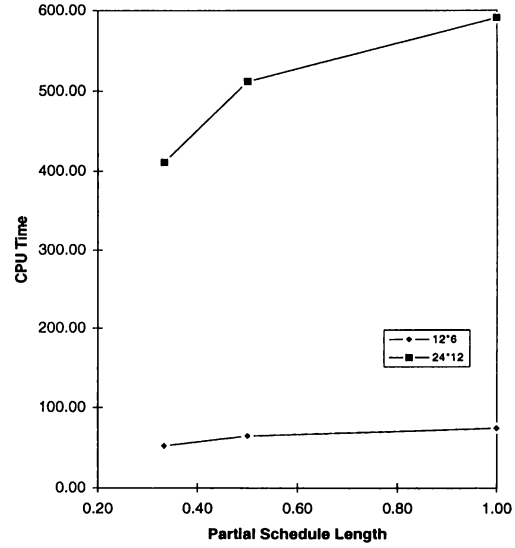
In addition, we note that the system complexity does not effect the performance of the partial scheduling. We arrive at this conclusion because the mean tardiness lines are almost parallel for small and large systems as seen in Figure 4.4a & 4.4c.

As for the CPU times, we observe that the CPU times increases as the partial schedule lengths increases (see Figure 4.4b and 4.4d). This is expected because we eliminate extra work to generate complete schedules at every scheduling period. In these graphs, CPU times for small systems are seen as if the changes are insignificant, however this is not the case. This is simply due to the high scale of the time axis.

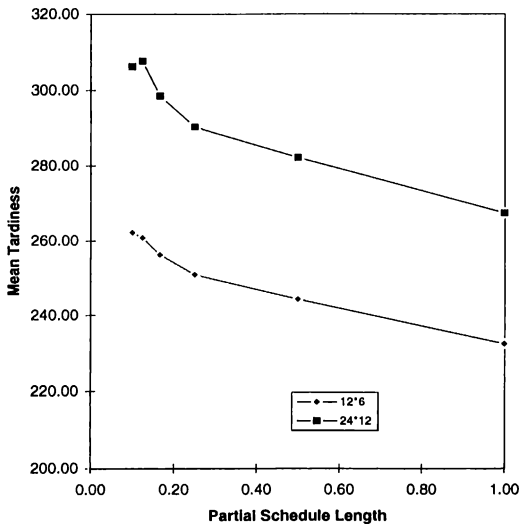
The patterns of segmented lines (Figure 4.4b & 4.4d) for the CPU times are consistent with the mean tardiness. For instance, the changes in the CPU times is also negligible for the small partial schedule lengths. Similar to the mean tardiness values, we observe significant increases in the CPU times for the moderate lengths of partial schedules. Finally, marginal increases in the CPU times are again small for the long partial schedules. Because in the beam search algorithms, generating the first half of the search tree requires more computation time than the generating the other half of the tree since global evaluations takes less time at the higher levels of the search.



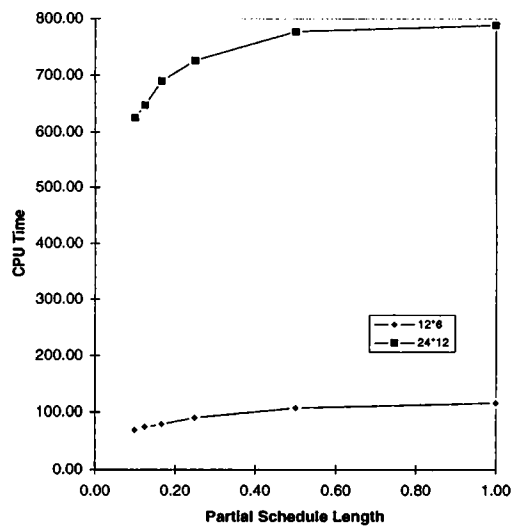
a) Frequency level of 4
Mean tardiness vs partial schedule length



b) Frequency level of 4
CPU time vs partial schedule length

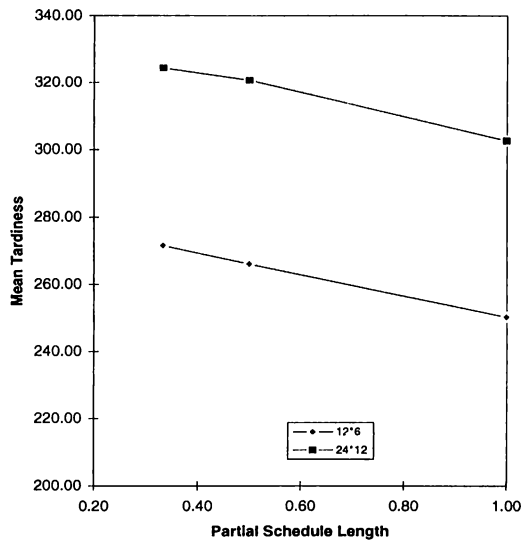


c) Frequency level of 14
Mean tardiness vs partial schedule length

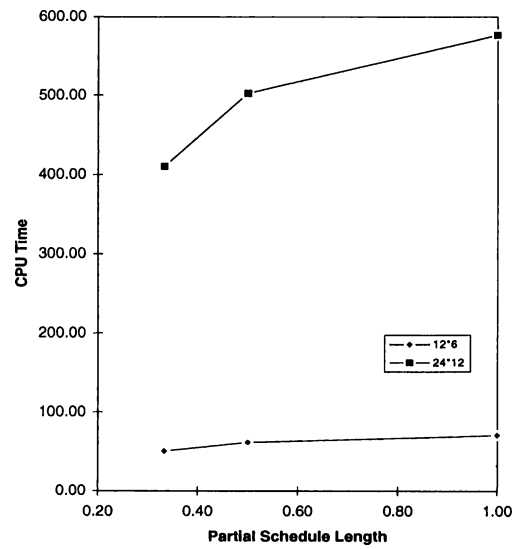


d) Frequency level of 14
CPU time vs partial schedule length

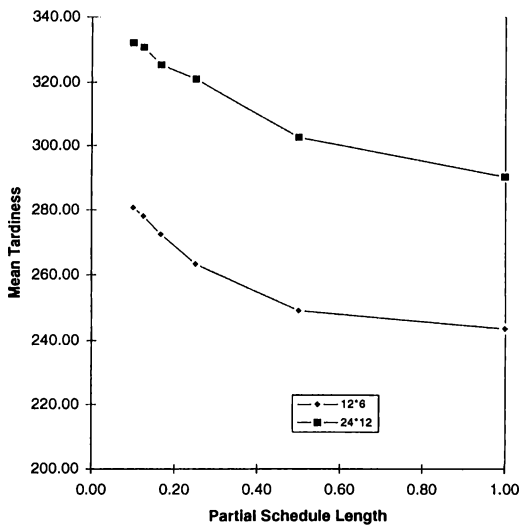
Figure 4.4: Mean tardiness & CPU time as a function of partial schedule lengths (Uniform Case)



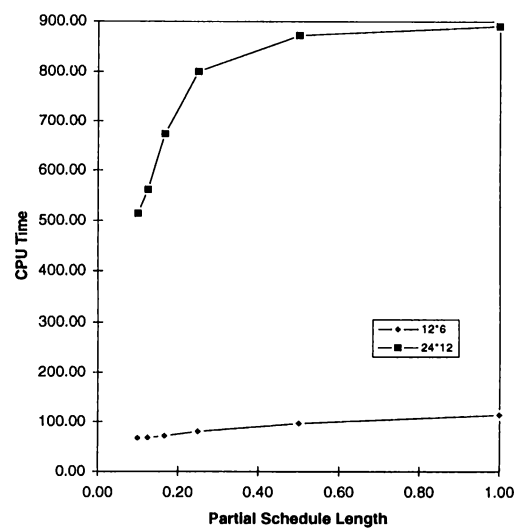
a) Frequency level of 4
Mean tardiness vs partial schedule length



b) Frequency level of 4
CPU time vs partial schedule length



c) Frequency level of 14
Mean tardiness vs partial schedule length



d) Frequency level of 14
CPU time vs partial schedule length

Figure 4.5: Mean tardiness & CPU time as a function of partial schedule lengths (Nonuniform Case)

By applying partial scheduling, the solution quality is sacrificed on the average % 10.6 and % 11.2 for 12 jobs 6 machines system and 24 jobs 12 machines system, respectively. However, we gain from the CPU times on the average % 35.2 and % 31.3 for small and large systems, respectively at the frequency levels of 4 and 14. In terms of percent difference, the gain of CPU times seems to be more significant than the degradation of solution quality. Here, user of the scheduling system should trade off the losses and gains to determine the most suitable policy for the current status.

The same analysis is performed in the nonuniformly loaded systems. The results of the analysis are depicted in Figure 4.5 (the numerical results are also given in Table B.8 & Table B.9 in the appendix). We compare the graphs in Figure 4.4 and 4.5, and note that the behavior of the segmented lines is very similar. Moreover, in order to compare the changes in the mean tardiness in the uniform and nonuniform cases, we plot the results of both cases in the same graph (Figure 4.6). As seen in the figure, the pattern of the segmented lines are the same in the both cases except that the mean tardiness values are larger in the nonuniformly loaded systems. As a result, we say that the effects of partial scheduling are the same in both uniformly and nonuniformly loaded systems.

Makespan Analysis

The effects of partial scheduling are also analyzed for the makespan performance measure. The simulation experiments are conducted under the experimental conditions specified earlier. The results (both the makespan and CPU times) of the analysis are shown in Figure 4.7 & 4.8 for both uniformly and nonuniformly loaded systems (the results are also tabulated in Table B.10 through B.13 in the appendix).

Our first observation is that as compared to the tardiness case, the deterioration in makespan is not significant for the small length of the partial schedule. Because, makespan performance measure is not considerably affected by the length of the partial schedule. Since, the additional inserted idle time due to partial scheduling has less probability to delay the completion time of the job that determines the makespan of the schedule. Another reason for this behavior can be the small amount

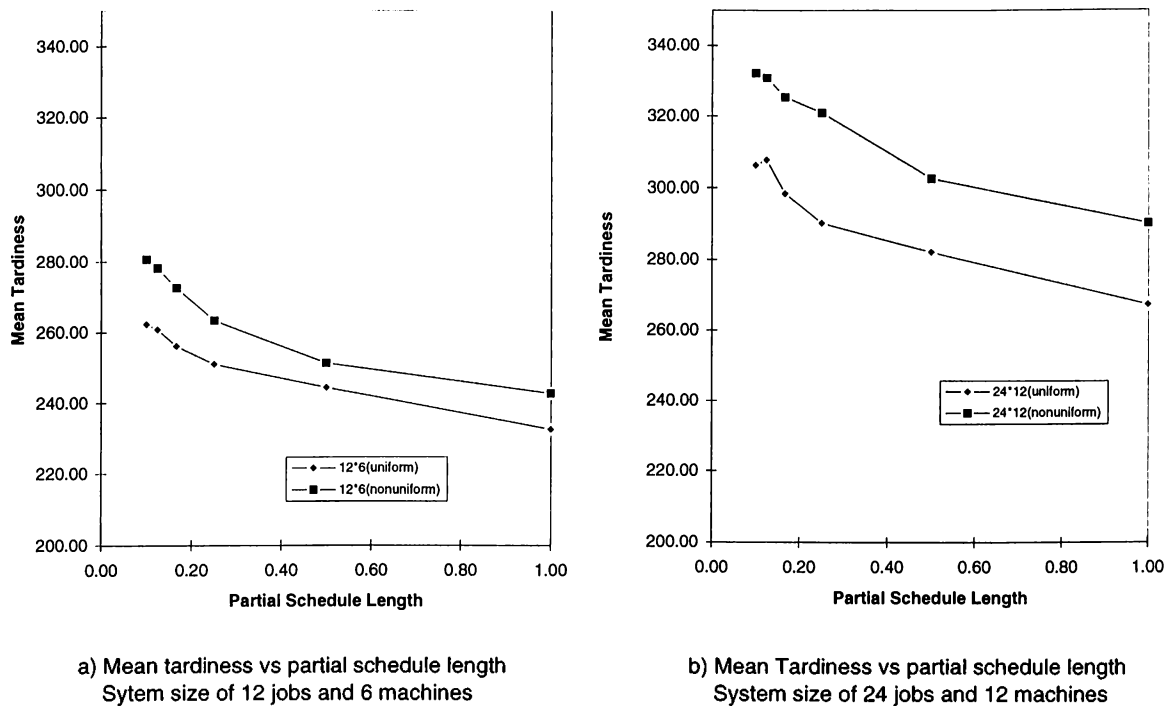
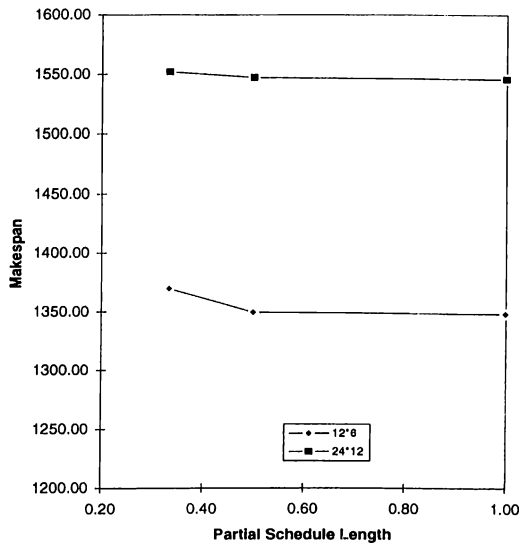


Figure 4.6: Changes in mean tardiness as a function partial schedule length in uniform and nonuniform environments

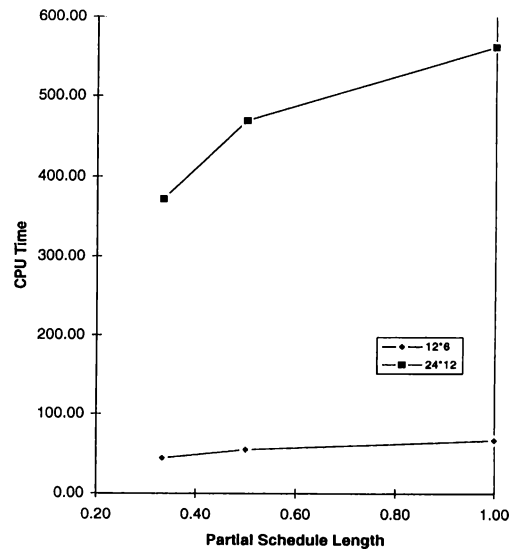
of inserted idle time resulted by the nondelay schedule generation scheme. Because, this scheme tries to first assign the operation with the minimum starting time. Therefore, the distribution of operations among the machines are homogeneous in the partial schedules, which reduces the extra need for the idle times. In addition, the scale of the makespan axis is too large so that changes in this value does not seem to be clear.

In the makespan case, other observations (related to the effects of the system size and load allocation of the system), and the discussions (for the improvement in the objective function as a result of the increase of the partial schedule length, and for the CPU times) are the same as the mean tardiness case.

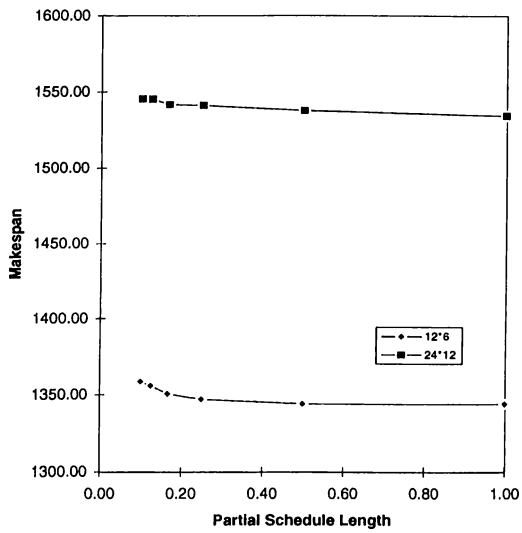
In the partial scheduling there are two major parameters (partial schedule length and scheduling frequency) that affects the solution quality. Since partial scheduling lengths are determined considering the scheduling frequency, these parameters are not independent. In the simulation experiments, the effects of partial schedule



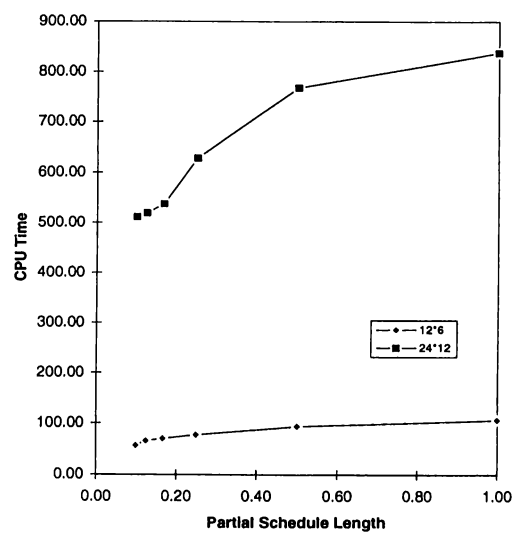
a) Frequency level of 4
Makespan vs partial schedule length



b) Frequency level of 4
CPU time vs partial schedule length

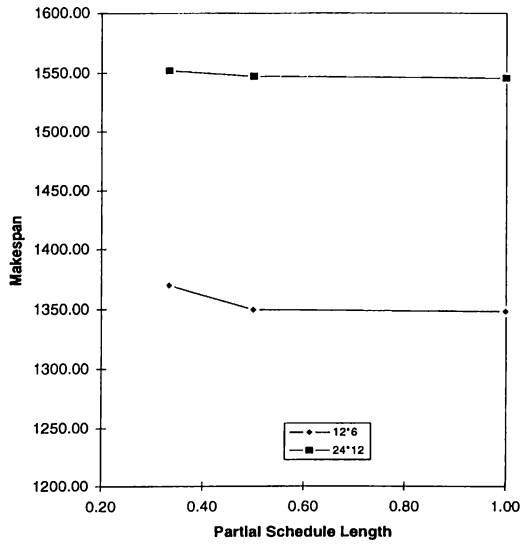


c) Frequency level of 14
Makespan vs partial schedule length

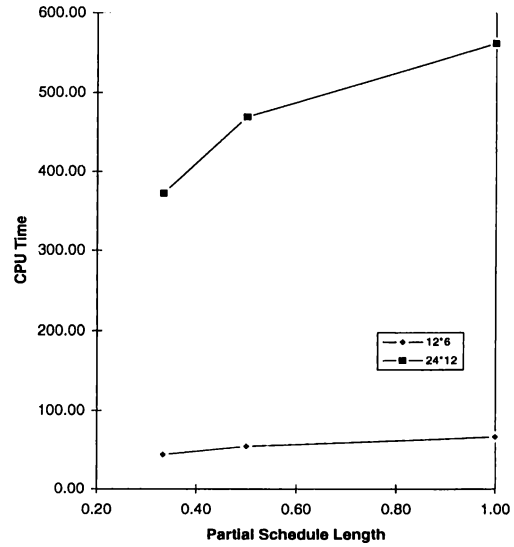


d) Frequency level of 14
CPU time vs partial schedule length

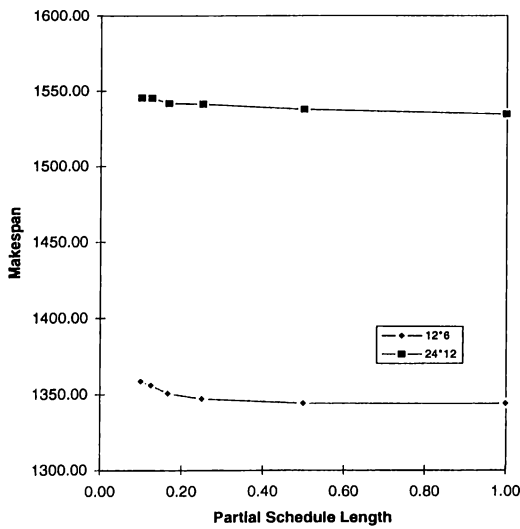
Figure 4.7: Interactions between makespan & CPU time and partial schedule length (Uniform Case)



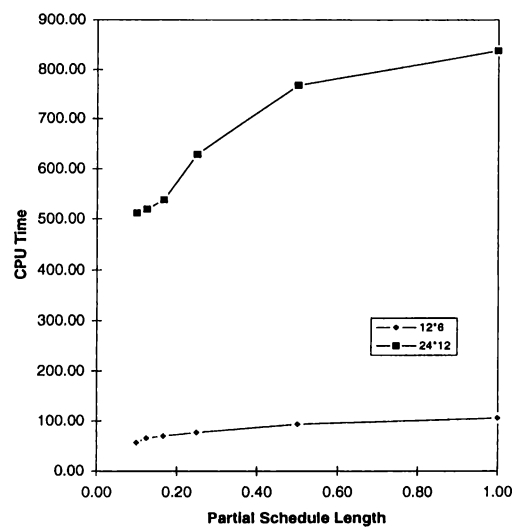
a) Frequency level of 4
Makespan vs partial schedule length



b) Frequency level of 4
CPU time vs partial schedule length



c) Frequency level of 14
Makespan vs partial schedule length



d) Frequency level of 14
CPU time vs partial schedule length

Figure 4.8: Interactions between makespan & CPU time and partial schedule length (Nonuniform Case)

lengths on the solution quality are analyzed for fixed scheduling frequencies. As for investigating the effects of scheduling frequency for the same partial schedule length, we compared the solutions the partial schedule length of $1/2$ at the scheduling frequency levels of 4 and 14. The results show that the solution quality at level 14 is always better than that of at level 4 for each problem. This observation confirms our finding in periodic response that as the scheduling frequency increases, the solution quality gets better.

So far we perform the simulation experiments for machine breakdowns of 90% efficiency level (with 360 time units mean uptime and 40 time units mean down time). We now investigate the effects of machine breakdown level on the solution quality of partial scheduling. For this purpose simulation experiments are repeated for 80% efficiency level (with 320 time units mean uptime and 40 time units mean down time) for both the mean tardiness and makespan criteria. The results of the partial scheduling at 90% and 80% efficiency levels are displayed together in Figure 4.9 and 4.10 for the mean tardiness and makespan, respectively (numerical results are also given in B14 and B15 in the appendix). The results indicate that solution quality is worse in the 80% efficiency level. Because at this level the system is subject to more machine breakdowns. However note that, solution pattern of partial schedule as a function of partial schedule length does not differ significantly. Since, the extra amount of idle time inserted in the schedule (due to partial scheduling) is not affected by the duration of machine breakdowns.

In summary, in the partial scheduling we observe that performance of the schedule worsens as the length of the partial schedule decreases. The deterioration in mean tardiness measure is more significant than the makespan. However, we also observe that, CPU times reduce as the length of partial schedule decreases for the mean tardiness and makespan performance measures. According to the experimental results, percent gain from CPU times is more than the percent loss from objective function for both criteria.

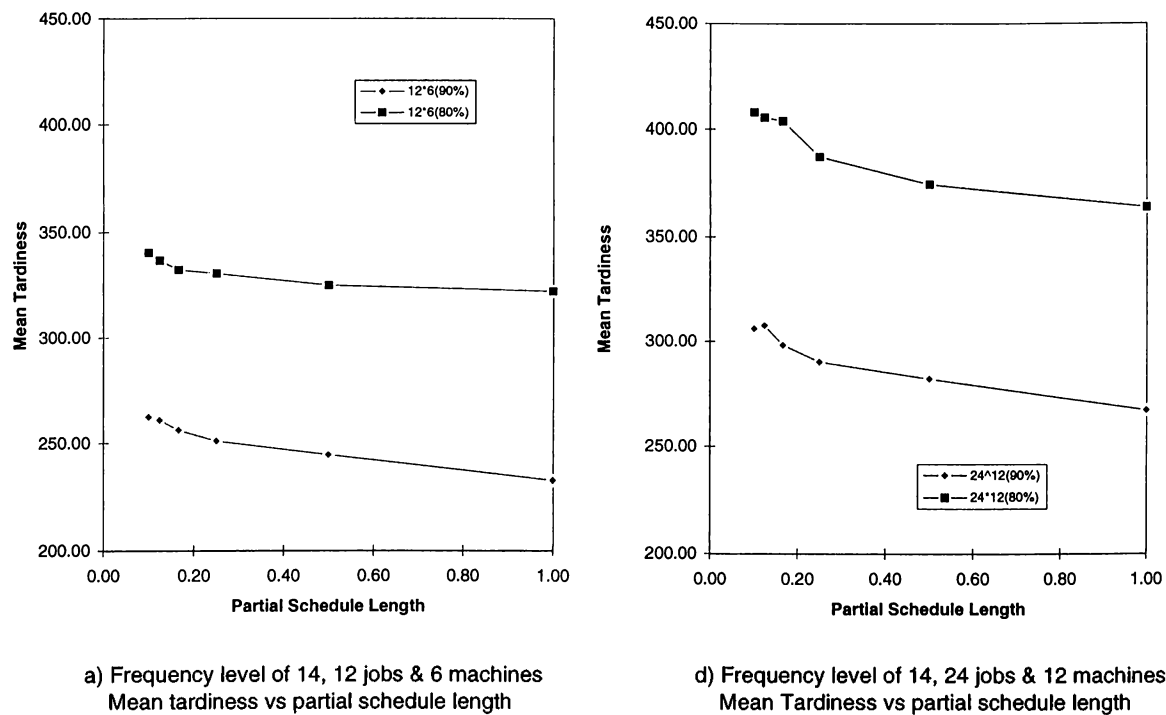


Figure 4.9: Mean Tardiness vs partial schedule length in 90% and 80% efficiency levels, Uniform Case

4.3.4 Partial Scheduling in Deterministic Environment

In the previous section, partial scheduling was analyzed as a part of the periodic rescheduling policy in the stochastic environments in which machine breakdowns were allowed to occur. Recall that in the stochastic environments, the schedule may need to be revised due to interruptions. There is no such requirement in the deterministic case. Hence, the implementation of partial schedule in a deterministic environment is nothing but a time-based decomposition of the entire problem into smaller problems.

As for decomposition methods in the scheduling literature there are a number of decomposition procedures for the single machine problem (Sidney, 1975; Potts and Van Wassenhove, 1982). Besides, Lagrangian relaxation techniques (Luh et al, 1990; Hoitom et al, 1993), as a part of integer programming approaches, are also used to evaluate different decomposition methods in order to form job level or operation level subproblems. Other decomposition methods available in the scheduling literature are the rolling horizon methods (Morton, 1981; Ovacik and

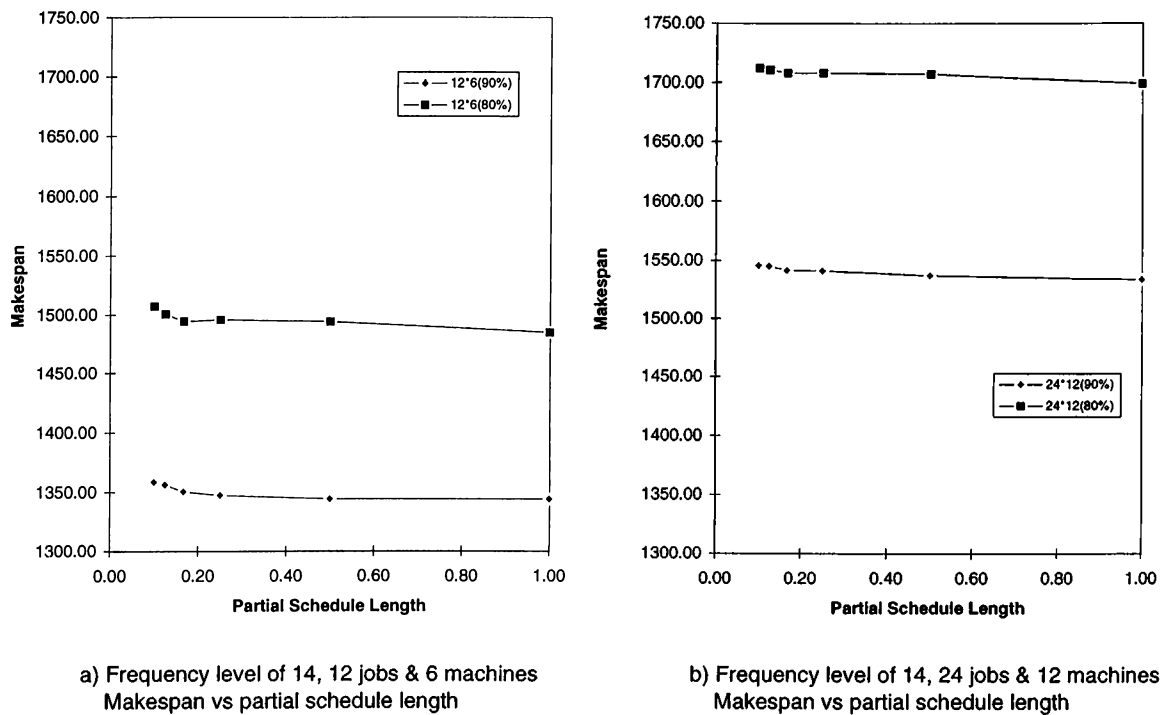


Figure 4.10: Makespan vs partial schedule length in 90% and 80% efficiency levels, Uniform Case

Uzsoy, 1994), the bottleneck dynamic approach (Morton and Pentico, 1993) and the graph decomposition technique (Wu et al, 1997). In addition, in this section we use the partial scheduling as a way of decomposing the static job shop scheduling problem.

In the deterministic environment, the frequency of scheduling should not have any effect on the performance of the partial scheduling method. Because in the periodic response policy implementation, rescheduling decision is not held at the next scheduling point unless there is a machine breakdown during the current period. In the deterministic environment, since we do not have any machine breakdown, rescheduling decision is only made after executing all the scheduled operations in the partial schedule. Therefore, partial schedule of certain length should give the same performance measure regardless of the scheduling frequency levels.

The previous experimental conditions are used in the analysis, except that the environment is deterministic (as explained in the section 4.3.1). Again, at the scheduling frequency levels of 4 and 14, the effects of various partial schedule lengths

are examined for the small and large shops in the both uniformly and nonuniformly loaded systems. Again the makespan and mean tardiness criteria are used as the performance measures.

Mean Tardiness Analysis

The effects of partial scheduling in deterministic environment with the above conditions are first tested for the mean tardiness performance measure. The changes in mean tardiness and CPU times as the function of partial schedule length are shown in Figure 4.11 (the numerical results are also given in Table B.17 and Table B.19 in the appendix for both uniformly and nonuniformly loaded systems).

The experimental results confirm our expectation that the scheduling frequency does not affect the results (see Table B.16 and B.17). The partial schedule lengths of $1/2$ and 1 give the same results for both scheduling frequency levels of 4 and 14 . For that reason graphs are given for only the frequency level of 14 are the results of the other scheduling frequency level are given in the appendix.

The first observation from the experiments is that as the length of partial schedule increases, the solution of the schedule improves. This is expected, and the myopic decisions and the need for the extra amount of inserted idle time in the schedules are again the reasons of this pattern.

In addition, the behavior of the segmented lines for the mean tardiness graphs is very similar to the ones in the partial scheduling section (see Figure 4.5c and Figure 4.11a). We again observe that the mean tardiness changes significantly for the moderate level of partial schedule length whereas the changes are insignificant for the short and very large partial schedule lengths. The explanation given previously for this type of behavior is also valid in this case.

The simulation experiments are also repeated for the nonuniformly loaded systems (see Figure 4.11c and 4.11d). Note that, in the nonuniform systems, the marginal improvement of the mean tardiness values in response to increase in the length of the partial schedule, is less than its uniform counterpart (compare Figures 4.11a and 4.11c). Because in the nonuniformly loaded systems there is already extra amount

of slack in the schedule. Consequently the some of inserted idle time due to partial scheduling may be absorbed by the slack in the schedule and hence deterioration in the performance of the schedule decreases in the nonuniformly loaded systems.

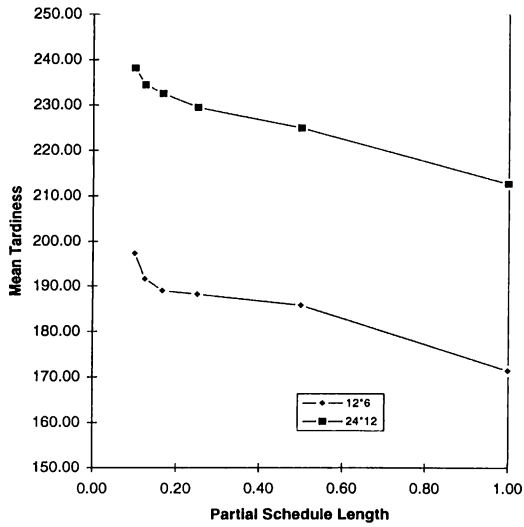
We also examine the effects of partial scheduling on the CPU time requirements. The results indicate that the CPU times do not considerably change as a function of the partial schedule length (see Figure 4.11b and 4.11d). Recall that in the proposed response system, a new partial schedule is generated after executing all the operations in the current partial schedule. Therefore, if the length of partial schedule is $1/4$, we repeat this process for 4 times in order to generate 1 complete schedule. In this case the amount of work required to generate 1 complete schedule is proportional to 4 times the work required to generate the partial schedule of length $1/4$ (which is equal to required work for generating the schedule without decomposition). Therefore, the amount of required work, and consequently the CPU time remains constant regardless of the partial schedule length.

In summary, partial scheduling in deterministic environment is not a recommended policy for the mean tardiness criterion. Because the performance of schedule deteriorates for the small partial schedule lengths without any CPU time saving.

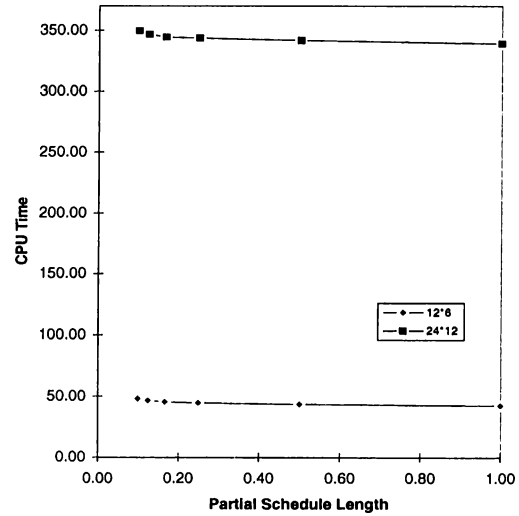
Makespan Analysis

The same analysis is also performed for the makespan performance measure. The results of the experiments for the scheduling frequency of 14 are shown in Figure 4.12 (the results are also given in Table B.21 and Table B.23 in the appendix for uniformly and nonuniformly loaded system, respectively). Again, the results of the analysis is given in the figures at the scheduling frequency of 14. Others are given in Table B.20 and B.22 in the appendix.

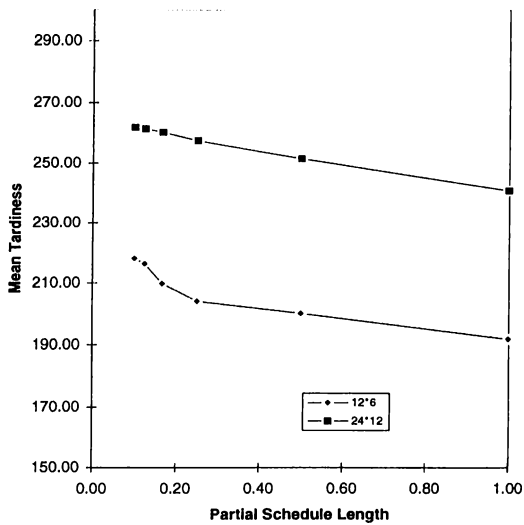
We observe from the graphs that the makespan values does not significantly improve as a result of the increase in the partial schedule length. This behavior of the segmented lines is very similar to the ones in the partial scheduling section. Therefore the explanation previously given for this behavior is also valid in this case.



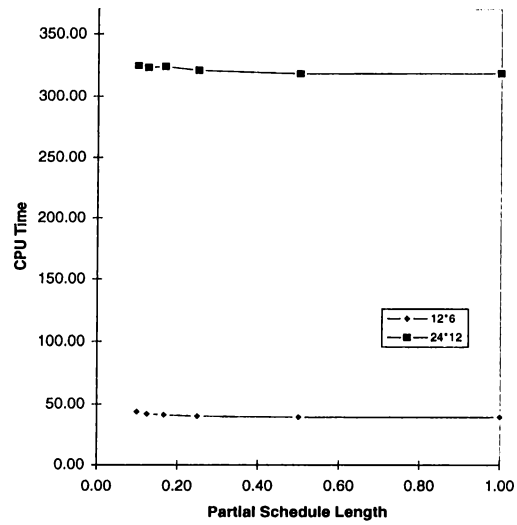
a) Frequency level of 14 (Uniform Case)
Mean tardiness vs partial schedule length



b) Frequency level of 14 (Uniform Case)
CPU time vs partial schedule length

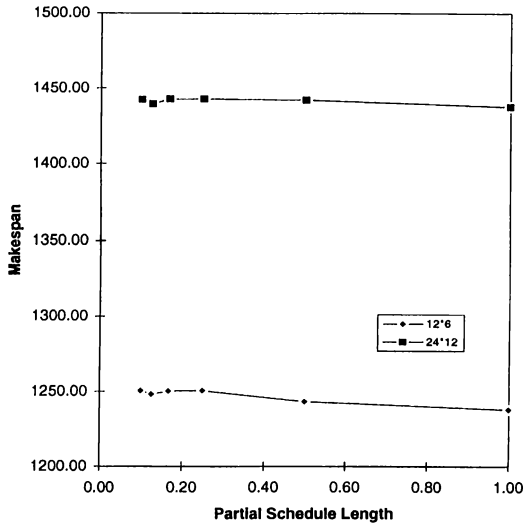


c) Frequency level of 14 (Nonuniform Case)
Mean tardiness vs partial schedule length

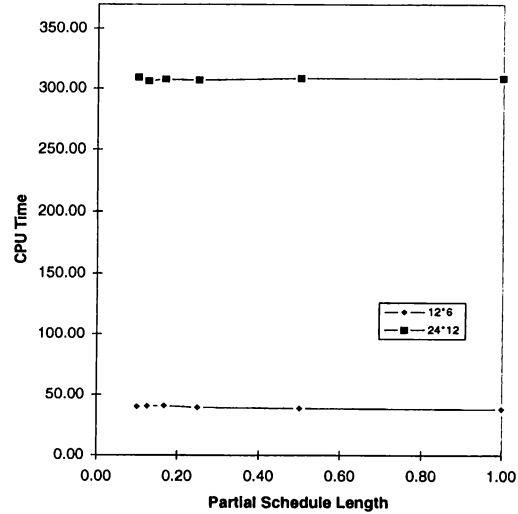


d) Frequency level of 14 (Nonuniform Case)
CPU time vs partial schedule length

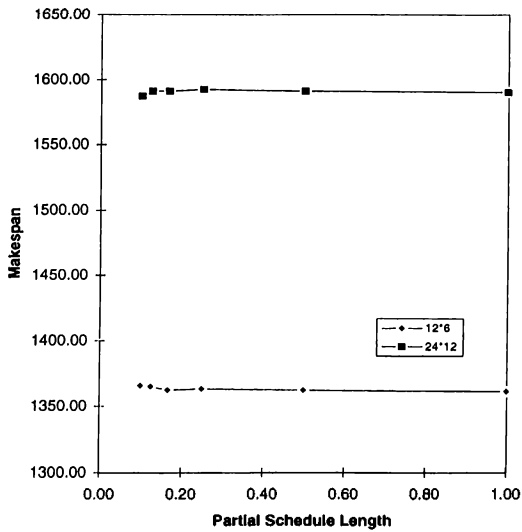
Figure 4.11: Interactions between mean tardiness & CPU time and partial schedule length in deterministic environment



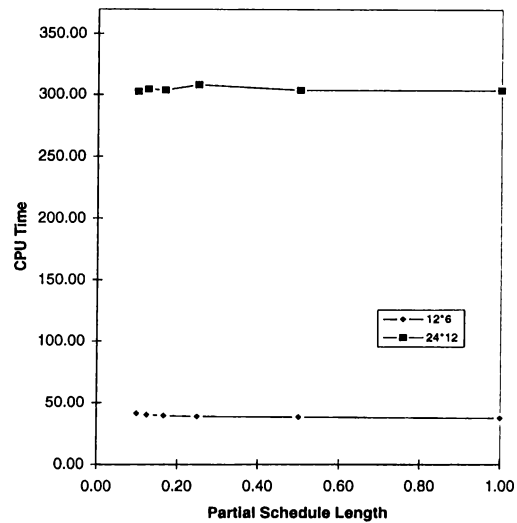
a) Frequency level of 14 (Uniform Case)
Makespan vs partial schedule length



b) Frequency level of 14 (Uniform Case)
CPU time vs partial schedule length



c) Frequency level of 14 (Nonuniform Case)
Makespan vs partial schedule length



d) Frequency level of 14 (Nonuniform Case)
CPU time vs partial schedule length

Figure 4.12: Interactions between makespan & CPU time and partial schedule length in deterministic environment

In addition, we note that the CPU time behavior is the same as in the case of the mean tardiness. Hence, the discussion given in the mean tardiness analysis can also be given here.

The results of the analysis for the nonuniformly loaded systems seem to be very similar to their uniform counterparts (compare Figure 4.12a and 4.12c). But, in the nonuniformly loaded systems, the marginal improvement in makespan values due to increase in the partial schedule length is slightly less than that of the uniformly loaded systems (see in Table B.21 and B.23 in the appendix). The same observation was also made in the mean tardiness analysis and the explanation given for this behavior in the mean tardiness analysis is also valid in this case.

4.4 Conclusion

In this chapter of the thesis we analyze the effects of shop floor configurations (system size and load allocation) on the performances of off-line and on-line scheduling methods. In the study a filtered beam search based algorithm and priority dispatching rule are used as off-line and on-line schedule generation methods. The performance of the system is evaluated for both the mean tardiness and makespan criteria.

The performance of the scheduling methods are analyzed first in the deterministic environment and we observe the following,

- System size does not affect the relative performance of the algorithms.
- The relative performance of the off-line algorithm gets better than the on-line algorithm in the crowded systems.
- In the nonuniformly loaded systems, the performance of both algorithms deteriorate, however deterioration in the performance of the on-line algorithm is less than that of the off-line algorithm.
- The patterns of the all results are very similar for both the makespan and the mean tardiness objective functions.

In the next part of the chapter, we assume that the machines in the system are

subject to random breakdowns. The on-line algorithm handles machine breakdowns while constructing the schedule. For the off-line algorithm we first analyze the *no response* policy in order to cope with the randomness. The results of this analysis is summarized as follows,

- The performance of the on-line algorithm is affected by the machine breakdowns less than performance of the off-line algorithm.
- System size again does not affect the amount of deterioration in solution of the scheduling algorithms.
- As only the number of jobs increased in the system, the machine breakdowns affect the performance of the scheduling algorithms more seriously.
- In the nonuniformly loaded systems the performance of the schedules does not decline as much as in the uniformly loaded systems.

In the second stage of the stochastic environment section, we examine the the effects of *periodic response* policy for the off-line scheduling algorithm. The results of the experiments are given as follows,

- The performance of the algorithm improves as the level scheduling frequency increases.
- The marginal improvement become smaller for the high scheduling frequency levels.
- Rescheduling gives relatively better results in the large systems.
- Effects of rescheduling is less significant in the nonuniformly loaded systems.

In the next section of the chapter, consequences of partial scheduling is examined by utilizing the constructive type of the algorithm. According to experimental results, we observe the following,

- (for the mean tardiness criterion) As the length of the partial schedule decreases, the performance of the schedules deteriorates due to extra amount of inserted idle time and myopic decisions.
- (for the makespan criterion) The amount of deterioration is less significant due to the nature of this objective function and the nondelay schedule generation scheme.
- (for both criteria) The CPU times get smaller as the length of partial schedule decrease, since the extra work to generate complete schedule is eliminated.
- (for both criteria) The effects of partial scheduling is the same in uniformly and

nonuniformly loaded systems.

Finally the notion of partial scheduling is used in the deterministic job shop problem in order to analyze the time based decomposition of this scheduling problem. The experiments show that,

- The patterns in the changes of the objective functions are the same as the ones in the partial scheduling analysis.
- The CPU times remain unchanged as the length of the partial schedules increases.

Chapter 5

CONCLUSION

This thesis consists of three parts. In the first part, we reviewed the reactive scheduling literature and classified the studies according to their problem environments, schedule generation methods and reactive scheduling implementations. In the next part, we developed a heuristic schedule generation algorithm based on the beam search technique and in the final part reactive scheduling policies are analyzed for various system size and load allocations. Detailed discussions of results of each study were presented in the conclusion sections of each chapters. Here, we summarize these findings in a comprehensive manner and outline further research directions.

At the first stage of the thesis, we used the beam search to solve the classic job shop scheduling problem for the makespan and mean tardiness criteria. We examined active and nondelay schedule generation schemes, different priority rules for both local and global evaluation functions, and various values of beam and filter width parameters. According to our computational results, we identified the proper settings of these parameters which were used in later parts of the thesis. Our computational experiments also indicated that beam search is a very good heuristic for the job shop problems.

In third chapter of the thesis we analyzed the effects of shop floor configurations (system size and load allocation) on the relative performances of off-line and on-line scheduling methods. In this study the filtered beam search based algorithm

and priority dispatching rule were used as off-line and on-line schedule generation methods. The performance of the system was evaluated for both the mean tardiness and makespan criteria.

The performance of the scheduling methods were first analyzed in the deterministic environment. We observed that system size did not affect the relative performance of the algorithms. In addition, in the nonuniformly loaded systems, the performance of the both methods deteriorated but we noted that deterioration of the dispatching rule's performance was less than the that of the algorithm.

In the next part of the chapter, we assume that the machines in the system were subject to random breakdowns. We first analyzed the *no response* policy for the off-line algorithm. The results of this analysis showed that the performance of the on-line method was affected less than performance of the off-line method. We again observed that the system size did not affect the amount of deterioration in solution of the scheduling methods. Besides, we noted that in the nonuniformly loaded systems the performance of the schedules does not decline as much as in the uniformly loaded systems.

In the second part of the stochastic environment section, we examined the effects of *periodic response* policy of the off-line scheduling algorithm. The results of the experiments showed that the performance of the algorithm improves as the scheduling frequency increases, but the marginal improvement became smaller for high scheduling frequencies. Here, rescheduling yielded relatively better results in the large systems. We also noticed that the effects of rescheduling were less significant in the nonuniformly loaded systems.

The performance of partial scheduling was examined by utilizing the beam search based constructive type algorithm. In the experiments various levels of partial schedule lengths were analyzed. The results indicated that as the length of the partial schedule decreases, the performance of the schedules deteriorates due to extra amount of inserted idle time and myopic decisions for the mean tardiness objective function. For the makespan case the amount of deterioration was less significant due to the nature of this objective function and the nondelay schedule generation scheme. We also noted that the CPU times became smaller as the length of partial

schedule decreased for both criteria. In addition, we observed that the effects of partial scheduling were the same regardless of the system size as well as the load allocations.

Finally, we used the partial scheduling in the deterministic job shop problem in order to analyze the time based decomposition of the job shop scheduling problem. Our simulation experiments showed that the patterns of changes in the objective functions were the same as the ones in the partial scheduling analysis discussed previously. In this section, we also noted the CPU times was not improved for the small partial schedule lengths. In conclusion, we can infer that decomposing the static job shop scheduling problem using partial scheduling concept would not help system controllers in the static environment. However, this decomposition scheme may work very well in the dynamic environment which require a further research.

From these conclusions, we suggest the following future research topics:

- One drawback of the beam search algorithm is imperfect assessment of the promise of nodes. As a result of this, the nodes that can lead to good solutions, are sometimes erroneously discarded. For the makespan problems, however, strong lower bounds are available in the literature. Hence, these lower bounds can be used as local evaluation function to improve the performance of the proposed beam search algorithm in future studies.
- Recall that in the global evaluation we estimate promise of a node by computing objective function of the complete schedule generated from the partial schedule represented by the node. In the beam search algorithm we used dispatching rules in the global evaluation function. Other methods that produce more accurate estimation can be used to improve the quality of the global evaluation function. In this context, instead of dispatching rules, a filtered beam search with simple evaluation functions and small filter and beam width parameters can be used to asses the promises of the nodes.
- In the analysis of reactive scheduling policies we use only machine breakdowns as disruption to the system. The events like, process time variation, arrival of urgent job, due date change can also be analyzed in this context. Moreover

the effects of different duration of mean machine up and down times for the same efficiency level can also be analyzed in the simulation experiments.

- In this study we use the static job shop problem. The same analysis can be performed for the dynamic environment. This way we may be able to understand whether all the conclusion made in this thesis for the static scheduling problem are also valid in the dynamic environment.
- Partial scheduling (in both deterministic and stochastic system) does not have outstanding performance in the static scheduling problems. However, we strongly believe that implementation of partial scheduling in the dynamic environment would be more beneficial for the system operations.

Bibliography

- [1] Aarts, E.H.L., Van Laarhoven, P.J.M., Ulder, N.L.J., "Local search based algorithms for job shop scheduling", Working Paper, 1991, Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands.
- [2] Aarts, E.H.L., Van Laarhoven, P.J.M., Lenstra, J.K., Ulder, N.L.J., "A computational Study of Local Search Algorithms for Job Shop Scheduling", *ORSA Journal on Computing*, 1994, Vol 6, No 2.
- [3] Adams, J., Balas, E., Zawack, D., "The Shifting Bottleneck Procedure for Job Shop Scheduling", *Management Science*, 1988, Vol 34, 391-401.
- [4] Akturk, S., Gorgulu, E., "Reactive Scheduling Under a Machine Breakdown", Technical Report IEOR-9316, 1993, Bilkent University, Department of Industrial Engineering.
- [5] Applegate, D., Cook, W., "A Computational Study of Job-Shop Scheduling", Technical Report CMU-CS-90-145, Carnegie Mellon University, School of Computer Science, 1990.
- [6] Baker, K.R., *Introduction to Sequencing and Scheduling*, Wiley, New York, 1974.
- [7] Baker, K.R., "Sequencing Rules and Due-Date Assignments in Job Shop", *Management Science*, 1984, Vol 30, No 9, 1093-1104.
- [8] Banks, J. Carson, J.S., *Discrete-Event System Simulation*, Prentice Hall Inc, 1984.

- [9] Bean, J., Birge, J.R., Mittenthal, J., Noon, C.E., "Matchup Scheduling with Multiple Resources, Release Dates and Disruptions", *Operations Research*, 1991, Vol 39, No 3, 470-483.
- [10] Bengu, G., "A Simulation-based Scheduler for Flexible Flowlines", *International Journal of Production Research*, 1994, Vol 32, No 2, pp 321-344.
- [11] Chang, Y., Matsuo, H., Sullivan, R.S., "A Bottleneck-based Beam Search for Job Scheduling in a Flexible Manufacturing System", *International Journal of Production Research*, 1989, Vol 27, No 11, 1949-1961.
- [12] Church, L.K., Uzsoy, R., "Analysis of Periodic and Event Driven Rescheduling Policies in Dynamic Shops", *International Journal of Computer Integrated Manufacturing*, 1992, Vol 5, No 3, 153-163.
- [13] De, S., Lee, A., "Flexible Manufacturing System (FMS) Scheduling Using Filtered Beam Search", *Journal of Intelligent Manufacturing*, 1990, Vol 1, 165-183.
- [14] Dutta, A., "Reacting to Scheduling Exceptions in FMS Environments", *IIE Transactions*, 1990, Vol 22, No 4, 300-314.
- [15] Farn, C.K., Muhleman, A.P., "The Dynamic Aspects of a Production Scheduling", *International Journal of Production Research*, 1979, Vol 17, No 15.
- [16] Fox, M.S., "Constraint Directed Search: A Case Study of Job Shop Scheduling", Ph.D Thesis, Carnegie Mellon University, 1983.
- [17] Fox, M.S., Smith, S.F., "ISIS - A Knowledge Based System for Factory Scheduling", *Expert Systems*, 1984, Vol 1, 25-49.
- [18] Garey, M.R., Johnson, D.S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, California, 1979.
- [19] Glover, F., "Tabu Search, Part-I", *ORSA Journal on Computing*, 1989, Vol 1, No 3, 190-206.

- [20] Glover, F., "Tabu Search, Part II", *ORSA Journal on Computing*, 1990, Vol 2, No 1, 4-32.
- [21] Hatzikonstantis, L., Besant, C.B., "Job-Shop Scheduling Using Certain Heuristic Search Algorithm", *International Journal of Advance Manufacturing Technology*, 1992, Vol 7, 251-261.
- [22] Hoitom, D.J., Luh, P.B., Pattipati, K.R., "A Practical Approach to Job Shop Scheduling Problems", *IEEE Transactions on Robotics and Automation*, 1993, No 9, pp 1-13.
- [23] Holloway, C.A., Nelson, R.T., "Job Shop Scheduling with Due Dates and Variable Processing Times", *Management Science*, 1974, Vol 20, No 9.
- [24] Jain, S., Foley W.J., "Real time control of manufacturing systems with redundancy", *Computer and Engineering*, 1987, No 2.
- [25] Jain, A.S., Meeran, S., *The Job Shop Problem: Past, Present and Future*, Department of Applied Physics and Electronic and Mechanical Engineering, University of Dundee, UK, 1996.
- [26] Kanet, J.J., Zhou, Z., "A Decision Theory Approach to Priority Dispatching for Job Shop Scheduling", *Production and Operations Management*, 1993, No 2, Vol 1, pp 2-14.
- [27] Kim, M.H., Kim, Y., "Simulation Based Real Time Scheduling in a Flexible Manufacturing Systems", *Journal of Manufacturing Systems*, 1994, Vol 13, No 2, 85-93
- [28] Kiran, A.S., Smith, M., "Simulation Studies in Job Shop Scheduling-I, A Survey", *Computer and Industrial Engineering*", 1984, Vol 8, No 2, 87-93.
- [29] Kiran, A.S., Alptekin, S., Kaplan A.C., "Tardiness Heuristic for Scheduling Flexible Manufacturing Systems", *Production Planning and Control*, 1991, Vol 2, No 3, 228-241.
- [30] Kutanoglu, E., Sabuncuoglu, I., "Experimental Investigation of Scheduling Rules in a Dynamic Job Shop with Weighted Tardiness Costs", *Third Industrial Engineering Research Conference*, 1994, 308-312.

- [31] Kutanoglu, E., "Job Shop Scheduling Under Dynamic and Stochastic Manufacturing Environment", Master Thesis, 1995, Bilkent University, Department of Industrial Engineering.
- [32] Lawrence, S., "Resource Constrained Project Scheduling: An experimental Investigation of Heuristic Scheduling Techniques", GSIA, Carnegie Mellon University, 1984.
- [33] Law, A.M., Kelton, W.D., *Simulation Modeling and Analysis*, 1991, McGraw-Hill.
- [34] Lowerre, B.T., The HARPY speech recognition system. PH.D. thesis, Carnegie-Mellon University, U.S.A., 1976.
- [35] Luh, P.B., Hoiomt, D.J., Max, E., Pattipati, K.R., "Schedule Generation and Reconfiguration for Parallel Machines", *IEEE Transactions on Robotics and Automation*, 1990, No 6, pp 687-696.
- [36] Martin, O., Otto, S.W., and Felten, E.W., "Large-step Markov Chains for Traveling Salesman Problem", *Complex Systems*, 1989, Vol 5, 299-326.
- [37] Matsua, H., Suh, C.J., and Sullivan, R.S., "A controlled search simulated annealing method for the general job shop scheduling problem", Working paper, 1988, 03-04-88, Graduate School of Business, The University of Texas at Austin, Texas, USA.
- [38] Matsuura, H., Tsubone, H., Kanezashi, M., "Sequencing, Dispatching, and Switching in a Dynamic Manufacturing Environment", *International Journal of Production Research*, 1993, Vol 31, No 7, 1671-1688.
- [39] Morton, T.E., Pentico, D.W., *Heuristic Scheduling Systems*, John Wiley & Sons, New York, 1993.
- [40] Muhleman, A.P., Lockett, A.G., Farn, C.K., "JobShop Scheduling Heuristics and Frequency of Scheduling", *International Journal of Production Research*, 1982, Vol 20, No 2, 227-241.

- [41] Nakano, R., Yamada, T., "Conventional genetic algorithm for job shop problems", , *Proceedings of the 4th International Conference on the Genetic Algorithms and Their Applications*, 1991, San Diego, California, USA, 474-479.
- [42] Nelson, R.T., Holloway, C.A., Wong, R.M., "Centralized Scheduling and Priority Implementation Heuristics for a Dynamic Job Shop Model", *AIIE Transactions*, 1977, Vol 9, No 1.
- [43] Nof, S.Y., Grant, F.H., "Adaptive/Predictive Scheduling: Review and a General Framework", *Production Planning and Control*, 1991, Vol 2, No 4, 298-312.
- [44] Ovacik, I.M., Uzsoy, R., "Exploiting Shop Floor Status Information to Schedule Complex Job Shops", *Journal of Manufacturing Systems*, Vol 13, No 2.
- [45] Ovacik, I.M., Uzsoy, R., "Rolling Horizon Algorithms for a Single Machine Dynamic Scheduling Problem with Sequence Dependent Setup Times", *International Journal of Production Research*, 1994, Vol 32, No 6, 1243-1263.
- [46] Ow, P.S., Morton, T.E., "Filtered Beam Search in Scheduling", *International Journal of Production Research*, 1988, Vol 26, No 1, 35-62.
- [47] Potts, C.N., Van Wassenhove, L.N., "A Decomposition Algorithm for the Single Machine Total Tardiness Problem", *Operations Research Letters*, 1982, Vol 1, No 5, pp 177-181.
- [48] Sabuncuoglu, I., Karabuk, S., "A Beam Search Algorithm and Evaluation of Scheduling Approaches for FMSs", Accepted for publication in *IIE Transactions*, 1997.
- [49] Sabuncuoglu, I., Karabuk, S., "Analysis of Scheduling Rescheduling Problems in a Stochastic Manufacturing Environment", Technical Report IEOR-9704, 1997, Bilkent University, Department of Industrial Engineering.
- [50] Sidney, J.B., "Decomposition Algorithms for Single Machine Sequencing with Precedence Relations and Deferral Costs", *Operations Research*, 1975, Vol 23, No 2, pp 283-298.

- [51] Smith, S.F., Ow, P.S., Potvin, J.Y., Muscettola, N., Matthys, D., "An Integrated Framework for Generating and Revising Factory Schedules", *Journal of Operational Research Society*, 1990, Vol 41, No 6, 539-552.
- [52] Szelke, E., Kerr, R.M., "Knowledge-based Reactive Scheduling", *Production Planning and Control*, 1994, Vol 5, no 2, 124-145.
- [53] Wu, S.D., Wysk, R.A., "Multipass Expert Control System - A Control/Scheduling Structure for Flexible Manufacturing Cells", *Journal of Manufacturing Systems*", 1988, Vol 7, No 2, p 107-120.
- [54] Wu, S.D., Wysk, R.A., "An Application of Discrete-event Simulation to On-line Control and Scheduling in Flexible Manufacturing", *International Journal of Production Research*, 1989, Vol 27, No 9, 1603-1623.
- [55] Wu, S.D., Storer, R.H., Byeon, E.S., "Decomposition Heuristics for Robust Job Shop Scheduling", *IEEE Transactions on Robotics and Automation*, 1997, Under Review.
- [56] Yamamoto, M., Nof, S.Y., "Scheduling in the Manufacturing Operating System Environment", *International Journal of Production Research*, 1985, Vol 23, No 4, 705-722.

Appendix A

Beam Search

.

Table A.1: Percent deviation from optimal solution vs filterwidth

f	b:1	b:2	b:3	b:4	b:5	b:6	b:7	b:8	Dispatch
Active Branching, Local: MWR, Global: MWR Dispatching Rule									
1	23.45	21.38	20.28	19.95	19.73	19.73	18.59	18.56	14.34
2	8.36	7.09	7.29	6.68	5.58	5.41	6.67	6.20	14.34
3	7.29	6.84	6.52	6.51	6.05	6.05	6.16	6.16	14.34
4	7.28	6.83	6.21	6.07	5.61	5.61	5.72	5.62	14.34
5	6.97	6.53	6.21	6.07	5.47	5.47	5.57	5.57	14.34
6	6.74	6.29	5.97	5.83	5.37	5.37	5.48	5.48	14.34
7	6.74	6.29	5.97	5.83	5.37	5.37	5.48	5.48	14.34
8	6.74	6.29	5.97	5.83	5.37	5.37	5.48	5.48	14.34
Nondelay Branching, Local: MWR, Global: MWR Dispatching Rule									
1	12.47	12.47	12.12	11.59	11.32	10.85	9.99	9.99	14.34
2	8.02	7.78	7.41	7.20	6.42	6.15	6.16	6.13	14.34
3	6.86	6.46	6.32	6.15	5.43	5.43	5.88	5.78	14.34
4	6.85	6.45	6.31	6.19	5.48	5.48	5.93	5.82	14.34
5	6.61	6.30	6.21	5.87	4.86	4.86	5.31	5.22	14.34
6	6.61	6.31	6.21	6.13	5.13	5.13	5.58	5.29	14.34
7	6.61	6.31	6.21	6.13	5.13	5.13	5.58	5.29	14.34
8	6.61	6.31	6.21	6.13	5.13	5.13	5.58	5.29	14.34
Active Branching, Local: LB, Global: MWR Dispatching Rule									
1	34.16	32.43	31.25	30.68	30.33	30.17	28.28	28.28	14.34
2	13.56	12.12	11.06	10.57	9.69	8.87	9.49	9.49	14.34
3	7.77	7.38	6.85	6.84	6.24	6.24	6.39	6.39	14.34
4	7.28	6.83	6.21	6.07	5.72	5.72	5.72	5.72	14.34
5	6.98	6.53	6.21	6.07	5.57	5.72	5.72	5.72	14.34
6	6.74	6.28	5.97	5.83	5.48	5.48	5.48	5.48	14.34
7	6.74	6.28	5.97	5.83	5.48	5.48	5.48	5.48	14.34
8	6.74	6.28	5.97	5.83	5.48	5.48	5.48	5.48	14.34
Nondelay Branching, Local: LB, Global: MWR Dispatching Rule									
1	13.55	13.55	12.91	12.38	12.09	11.63	10.84	10.84	14.34
2	8.41	8.16	7.82	7.29	6.43	6.16	6.16	6.16	14.34
3	6.86	6.46	6.33	6.15	5.44	5.44	5.88	5.77	14.34
4	6.85	6.45	6.31	6.19	5.48	5.48	5.93	5.82	14.34
5	6.60	6.29	6.21	5.87	4.86	4.86	5.31	5.23	14.34
6	6.61	6.31	6.21	6.14	5.13	5.13	5.58	5.49	14.34
7	6.61	6.31	6.21	6.14	5.13	5.13	5.58	5.49	14.34
8	6.61	6.31	6.21	6.14	5.13	5.13	5.58	5.49	14.34

Table A.2: Mean tardiness vs filterwidth analysis

f	b:1	b:2	b:3	b:4	b:5	b:6	b:7	b:8	Dispatch
Active Branching, Local: SPT, Global: SPT Dispatching Rule									
1	1641.7	1585.7	1485.6	1415.2	1300.9	1300.6	1266.4	1254.2	485.6
2	636.1	486.7	403.7	352.2	349.4	335.7	317.1	317.1	485.6
3	398.3	326.0	264.0	212.1	212.1	212.1	201.0	201.0	485.6
4	295.1	263.2	213.9	200.2	200.2	200.2	198.5	198.5	485.6
5	268.5	265.5	216.2	202.2	202.2	202.2	200.5	200.5	485.6
6	268.5	265.5	216.2	202.2	202.2	202.2	200.5	200.5	485.6
7	268.5	265.5	216.2	202.2	202.2	202.2	200.5	200.5	485.6
8	268.5	265.5	216.2	202.2	202.2	202.2	200.5	200.5	485.6
Active Branching, Local: SPT, Global: SPT Dispatching Rule									
1	450.7	434.3	402.9	371.8	369.4	369.4	364.4	364.4	485.6
2	323.6	307.0	303.7	274.6	277.4	271.1	270.9	252.5	485.6
3	316.9	293.5	267.1	232.9	241.0	241.0	228.3	228.3	485.6
4	287.0	270.4	244.0	232.6	240.7	240.7	228.0	228.0	485.6
5	287.0	274.7	248.3	236.9	245.0	240.7	232.3	232.3	485.6
6	287.0	274.7	248.3	236.9	245.0	240.7	232.3	232.3	485.6
7	287.0	274.7	248.3	236.9	245.0	240.7	232.3	232.3	485.6
8	287.0	274.7	248.3	236.9	245.0	240.7	232.3	232.3	485.6
Active Branching, Local: EDD, Global: EDD Dispatching Rule									
1	1300.9	1214.5	1181.9	1175.1	1139.4	1104.4	1124.2	1100.0	485.6
2	333.5	311.2	309.9	266.3	249.3	249.3	249.3	248.9	485.6
3	258.4	250.5	240.0	237.5	237.5	226.9	220.2	216.2	485.6
4	256.8	240.3	239.6	220.6	220.6	218.3	217.0	216.2	485.6
5	256.8	240.3	239.6	220.6	220.6	218.3	217.0	216.2	485.6
6	256.8	240.3	239.6	220.6	220.6	216.6	216.6	216.6	485.6
7	256.8	240.3	239.6	220.6	220.6	216.6	216.6	216.6	485.6
8	256.8	240.3	239.6	220.6	220.6	216.6	216.6	216.6	485.6
Nondelay Branching, Local: EDD, Global: EDD Dispatching Rule									
1	475.3	475.3	414.8	388.4	376.7	374.9	360.0	360.0	485.6
2	366.4	316.4	291.1	276.1	268.0	254.4	211.6	211.6	485.6
3	342.7	293.9	280.3	268.2	258.7	247.9	202.6	202.6	485.6
4	324.7	275.7	267.1	256.9	247.4	223.0	200.3	200.3	485.6
5	312.3	271.2	262.6	252.4	247.4	223.0	200.3	200.3	485.6
6	312.3	271.2	262.6	252.4	247.4	232.3	209.3	209.6	485.6
7	312.3	271.2	262.6	252.4	247.4	232.3	209.3	209.6	485.6
8	312.3	271.2	262.6	252.4	247.4	232.3	209.3	209.6	485.6

Table A.3: Mean tardiness vs filterwidth analysis

f	b:1	b:2	b:3	b:4	b:5	b:6	b:7	b:8	Dispatch
Active Branching, Local: MDD, Global: MDD Dispatching Rule									
1	1221.3	1182.0	1155.0	1129.6	1117.6	1101.9	1133.8	1116.2	485.6
2	365.8	333.9	319.4	285.3	274.4	273.2	266.9	265.7	485.6
3	294.1	286.0	250.4	262.6	248.3	217.5	189.1	189.1	485.6
4	274.0	269.9	265.6	243.2	241.5	224.1	202.7	202.7	485.6
5	274.0	270.4	262.0	243.2	241.5	224.1	202.7	202.7	485.6
6	274.0	270.4	262.0	243.2	241.5	227.8	206.4	206.4	485.6
7	274.0	270.4	266.1	243.2	241.5	227.8	206.4	206.4	485.6
8	274.0	270.4	266.1	243.2	241.5	227.8	206.4	206.4	485.6
Active Branching, Local: MDD, Global: SPT Dispatching Rule									
1	1267.3	1224.9	1141.9	1103.0	1103.0	1085.4	1129.6	1110.2	485.6
2	577.4	455.8	351.3	317.6	296.5	276.0	281.2	281.2	485.6
3	242.7	233.4	222.5	192.6	192.6	186.1	185.9	183.8	485.6
4	233.7	230.7	219.8	198.2	198.2	191.7	188.3	188.3	485.6
5	233.7	230.7	219.8	198.2	198.2	191.7	188.3	188.3	485.6
6	233.7	230.7	219.8	198.2	198.2	191.7	188.3	188.3	485.6
7	233.7	230.7	219.8	198.2	198.2	191.7	188.3	188.3	485.6
8	233.7	230.7	219.8	198.2	198.2	191.7	188.3	188.3	485.6
Active Branching, Local: EDD, Global: SPT Dispatching Rule									
1	1282.9	1256.6	1166.6	1134.4	1131.7	1095.3	1086.7	1075.7	485.6
2	569.4	450.1	340.9	313.5	291.8	276.0	276.5	276.5	485.5
3	242.7	232.7	221.8	192.6	192.6	186.1	185.9	183.8	485.6
4	232.9	229.9	219.0	198.2	198.2	191.7	188.3	188.3	485.6
5	232.9	229.9	219.0	198.2	198.2	191.7	188.3	188.3	485.6
6	232.9	229.9	219.0	198.2	198.2	191.7	188.3	188.3	485.6
7	232.9	229.9	219.0	198.2	198.2	191.7	188.3	188.3	485.6
8	232.9	229.9	219.0	198.2	198.2	191.7	188.3	188.3	485.6
Active Branching, Local: MOD, Global: SPT Dispatching Rule									
1	1053.5	995.6	989.7	989.7	989.7	989.7	994.8	986.7	485.6
2	376.0	326.0	304.1	269.6	264.4	259.2	259.2	259.2	485.6
3	281.1	267.4	220.7	192.0	192.0	186.8	179.0	179.0	485.7
4	269.4	246.6	199.9	196.9	196.9	191.7	186.9	186.9	485.7
5	269.4	252.7	206.0	194.9	194.9	189.7	184.9	184.9	485.7
6	269.4	252.7	206.0	194.9	194.9	189.7	184.9	184.9	485.7
7	269.4	252.7	206.0	194.9	194.9	189.7	184.9	184.9	485.7
8	269.4	252.7	206.0	194.9	194.9	189.7	184.9	184.9	485.7

Appendix B

Reactive Scheduling

Table B.1: Interactions between mean tardiness and scheduling frequency

Frequency	Uniform Case			
	jobs * machines			
	9*6	12*6	18*12	24*12
0	118.9	242.5	138.9	285.7
2	119.8	242.0	134.1	280.6
4	117.1	242.1	130.9	273.1
6	113.3	237.7	130.4	271.7
8	109.7	236.7	126.5	273.1
10	107.7	234.9	125.4	269.9
12	107.9	233.7	126.7	269.1
14	108.7	232.5	124.7	267.3
16	107.0	233.4	123.7	267.9
1000	105.5	230.1	119.9	263.5

Table B.2: Interactions between mean tardiness and scheduling frequency

Frequency	Nonuniform Case			
	jobs * machines			
	9*6	12*6	18*12	24*12
0	169.9	288.3	172.1	312.9
2	164.1	283.3	163.2	306.8
4	166.1	282.8	161.2	308.8
6	160.2	282.7	159.9	301.1
8	161.9	281.1	156.5	300.4
10	158.6	281.5	154.6	299.4
12	160.9	278.8	153.5	302.9
14	158.4	277.4	154.8	299.2
16	158.5	275.2	155.2	300.7
1000	155.9	274.3	154.0	298.5

Table B.3: Interactions between CPU time and scheduling frequency

Frequency	jobs * machines			
	9*6	12*6	18*12	24*12
0	14.8	42.9	116.2	368.2
2	19.0	51.3	137.9	412.7
4	27.8	74.2	200.9	590.8
6	30.2	96.9	255.2	639.3
8	34.4	103.4	319.6	661.3
10	38.2	105.0	364.0	681.2
12	37.7	115.2	394.9	730.8
14	37.9	116.1	403.1	788.1
16	39.3	120.5	416.4	797.9
1000	46.8	156.6	572.3	820.9

Table B.4: Interactions between makespan and scheduling frequency

Uniform Case				
jobs * machines				
Frequency	9*6	12*6	18*12	24*12
0	1119.2	1372.3	1218.8	1562.8
2	1117.2	1358.9	1213.9	1561.5
4	1117.0	1347.5	1214.6	1545.3
6	1113.0	1347.7	1214.3	1547.5
8	1111.0	1341.2	1216.0	1540.7
10	1108.5	1345.0	1212.7	1538.3
12	1109.5	1346.9	1206.2	1532.0
14	1108.1	1344.1	1208.0	1534.0
16	1108.6	1341.5	1203.2	1529.4
1000	1108.0	1340.2	1200.4	1526.7

Table B.5: Interactions between makespan and scheduling frequency

Nonuniform Case				
jobs * machines				
Frequency	9*6	12*6	18*12	24*12
0	1292.4	1472.1	1317.3	1714.2
2	1291.2	1470.9	1317.6	1705.6
4	1284.9	1478.7	1323.9	1707.3
6	1279.2	1465.6	1319.8	1712.6
8	1280.6	1467.5	1324.1	1702.9
10	1284.3	1470.2	1320.6	1695.9
12	1283.4	1467.6	1322.2	1698.8
14	1280.5	1472.9	1317.7	1690.0
16	1278.8	1471.5	1315.7	1692.6
1000	1279.4	1456.8	1311.2	1686.2

Table B.6: Interactions between mean tardiness & CPU time and partial schedule length

Frequency level of 4 (Uniform Case)

Partial Length	Mean Tardiness		CPU Time	
	12*6	24*12	12*6	24*12
1/3	262.66	295.25	52.56	411.63
1/2	252.58	288.80	64.95	512.08
1	242.08	273.12	74.23	590.76

Table B.7: Interactions between mean tardiness & CPU time and partial schedule length

Frequency level of 14 (Uniform Case)

Partial Length	Mean Tardiness		CPU Time	
	12*6	24*12	12*6	24*12
1/10	262.20	306.24	69.35	624.67
1/8	260.81	307.63	74.94	646.82
1/6	256.12	298.43	79.91	689.55
1/4	250.93	290.31	90.80	765.49
1/2	244.30	282.05	107.26	776.96
1	232.49	267.33	116.14	788.08

Table B.8: Interactions between mean tardiness & CPU time and partial schedule length

Frequency level of 4 (Nonuniform Case)

Partial Length	Mean Tardiness		CPU Time	
	12*6	24*12	12*6	24*12
1/3	271.52	324.23	50.61	409.97
1/2	266.13	320.63	62.01	502.50
1	250.16	302.65	70.73	576.41

Table B.9: Interactions between mean tardiness & CPU time and partial schedule length

Frequency level of 14 (Nonuniform Case)

Partial Length	Mean Tardiness		CPU Time	
	12*6	24*12	12*6	24*12
1/10	280.59	332.13	67.09	513.83
1/8	278.08	330.76	68.71	561.35
1/6	272.39	325.26	72.24	673.50
1/4	263.32	320.91	81.04	800.12
1/2	249.19	302.50	96.88	872.32
1	243.65	290.19	113.70	890.03

Table B.10: Interactions between makespan & CPU time and partial schedule length
 Frequency level of 4 (Uniform Case)

Partial Length	Makespan		CPU Time	
	12*6	24*12	12*6	24*12
1/3	1369.6	1551.6	43.83	372.39
1/2	1349.2	1547.0	54.89	469.25
1	1347.5	1545.3	66.23	562.24

Table B.11: Interactions between makespan & CPU time and partial schedule length
 Frequency level of 14 (Uniform Case)

Partial Length	Makespan		CPU Time	
	12*6	24*12	12*6	24*12
1/10	1358.5	1558.2	57.04	511.67
1/8	1355.9	1557.8	65.90	519.67
1/6	1350.5	1550.3	70.43	537.68
1/4	1347.1	1548.9	76.83	628.40
1/2	1344.3	1547.0	94.01	767.41
1	1344.1	1545.3	106.76	837.42

Table B.12: Interactions between makespan & CPU time and partial schedule length
Frequency level of 4 (Nonuniform Case)

Partial Length	Makespan		CPU Time	
	12*6	24*12	12*6	24*12
1/3	1484.5	1689.5	50.61	409.97
1/2	1482.5	1673.8	62.01	502.50
1	1478.7	1707.3	70.73	576.41

Table B.13: Interactions between makespan & CPU time and partial schedule length
Frequency level of 14 (Nonuniform Case)

Partial Length	Makespan		CPU Time	
	12*6	24*12	12*6	24*12
1/10	1477.7	1702.3	67.09	513.83
1/8	1475.0	1698.0	68.71	561.35
1/6	1467.3	1696.6	72.24	673.50
1/4	1467.8	1693.7	81.04	800.12
1/2	1470.4	1690.8	96.88	872.32
1		1472.9	1690.0	890.03

Table B.14: Mean tardiness vs partial schedule length in 90% and 80% efficiency levels (Uniform case)

Partial Length	12*6(90%)	12*6(80%)	24*12(90%)	24*12(80%)
1/10	262.2	340.26	306.24	407.93
1/8	260.81	336.53	307.63	405.32
1/6	256.12	332.22	298.43	403.70
1/4	250.93	330.54	290.31	387.11
1/2	244.30	324.91	282.05	374.34
1	232.49	322.27	267.33	363.97

Table B.15: Makespan vs partial schedule length in 90% and 80% efficiency levels (Uniform case)

Partial Length	12*6(90%)	12*6(80%)	24*12(90%)	24*12(80%)
1/10	1358.5	1506.9	1545.2	1712.0
1/8	1355.9	1500.4	1544.8	1710.7
1/6	1350.5	1494.1	1541.3	1708.0
1/4	1347.1	1495.6	1540.9	1707.7
1/2	1344.3	1493.7	1537.2	1706.8
1	1344.1	1485.2	1534.0	1689.9

Table B.16: Interactions between mean tardiness & CPU time and partial schedule length in deterministic environment

Frequency level of 4 (Uniform Case)

Partial Length	Mean Tardiness		CPU Time	
	12*6	24*12	12*6	24*12
1/3	185.83	229.46	44.28	345.60
1/2	184.25	224.94	43.56	341.98
1	171.35	212.61	42.61	341.10

Table B.17: Interactions between mean tardiness & CPU time and partial schedule length in deterministic environment

Frequency level of 14 (Uniform Case)

Partial Length	Mean Tardiness		CPU Time	
	12*6	24*12	12*6	24*12
1/10	197.26	238.13	48.23	349.55
1/8	191.59	234.36	46.71	346.71
1/6	189.06	232.49	45.61	344.43
1/4	188.26	229.46	44.83	343.73
1/2	185.83	224.94	43.67	341.73
1	171.35	212.61	42.65	339.25

Table B.18: Interactions between mean tardiness & CPU time and partial schedule length in deterministic environment

Frequency level of 4 (Nonuniform Case)

Partial Length	Mean Tardiness		CPU Time	
	12*6	24*12	12*6	24*12
1/3	206.13	259.20	39.82	322.66
1/2	200.22	251.37	39.61	319.42
1	191.95	240.89	38.94	318.48

Table B.19: Interactions between mean tardiness & CPU time and partial schedule length in deterministic environment

Frequency level of 14 (Nonuniform Case)

Partial Length	Mean Tardiness		CPU Time	
	12*6	24*12	12*6	24*12
1/10	218.17	261.74	43.65	324.90
1/8	216.45	261.29	41.91	323.23
1/6	209.92	260.08	41.42	324.00
1/4	204.18	257.25	40.17	320.93
1/2	200.22	251.37	39.58	318.37
1	191.95	240.89	39.39	318.89

Table B.20: Interactions between makespan & CPU time and partial schedule length in deterministic environment

Frequency level of 4 (Uniform Case)

Partial Length	Makespan		CPU Time	
	12*6	24*12	12*6	24*12
1/3	1245.0	1443.1	38.63	312.07
1/2	1243.1	1441.9	38.57	310.85
1	1237.9	1437.4	37.95	311.32

Table B.21: Interactions between makespan & CPU time and partial schedule length in deterministic environment

Frequency level of 14 (Uniform Case)

Partial Length	Makespan		CPU Time	
	12*6	24*12	12*6	24*12
1/10	1250.4	1442.1	39.88	309.39
1/8	1248.0	1439.4	40.56	306.46
1/6	1250.2	1442.4	40.57	307.87
1/4	1250.5	1442.6	39.21	307.19
1/2	1243.1	1441.9	38.54	308.49
1	1237.9	1437.4	37.78	308.6

Table B.22: Interactions between makespan & CPU time and partial schedule length in deterministic environment

Frequency level of 4 (Nonuniform Case)

Partial Length	Makespan		CPU Time	
	12*6	24*12	12*6	24*12
1/3	1363.0	1590.2	38.38	298.68
1/2	1362.4	1590.8	38.35	303.19
1	1361.8	1590.2	37.36	302.41

Table B.23: Interactions between makespan & CPU time and partial schedule length in deterministic environment

Frequency level of 14 (Nonuniform Case)

Partial Length	Makespan		CPU Time	
	12*6	24*12	12*6	24*12
1/10	1366.1	1587.1	41.32	302.41
1/8	1365.1	1590.8	40.06	304.29
1/6	1362.4	1590.8	39.52	303.66
1/4	1363.6	1592.2	38.74	307.87
1/2	1362.4	1590.8	38.23	303.73
1	1361.8	1590.2	37.43	303.54