

REACTIVE SCHEDULING IN
CELLULAR 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
Elif Görgülü
September, 1993

Thesis
TS
157.5
.G67
1993

REACTIVE SCHEDULING IN
CELLULAR MANUFACTURING SYSTEMS

A THESIS

SUBMITTED TO THE DEPARTMENT OF INDUSTRIAL
ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By

Elif Görgülü

September, 1993

B_ 1227

TS

157.5

19667

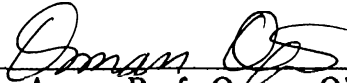
1983

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.



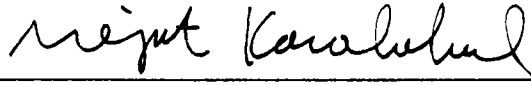
Asst. Prof. Selim Aktürk (Advisor)

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. Osman Oğuz

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.



Asst. Prof. Nejat Karabakal

Approved for the Institute of Engineering and Science:



Prof. Mehmet Baray
Director of the Institute

ABSTRACT

REACTIVE SCHEDULING IN CELLULAR MANUFACTURING SYSTEMS

Elif Görgülü

M.S. in Industrial Engineering

Advisor: Asst. Prof. Selim Aktürk

September, 1993

A real-time scheduling problem, specifically rescheduling under machine breakdowns, is considered in this thesis. A heuristic approach is developed for a manufacturing cell with a modified flow shop structure. The strategy reschedules part of the initial schedule to match up with the preschedule at some point. In contrast to other studies, the objective is to create a new schedule that is consistent with the other production planning decisions like material flow and purchasing by utilizing the time critical decision making concept. In the proposed heuristic approach, a new rescheduling strategy is developed and different match-up points are defined for each machine in order to maximize the flexibility during rescheduling. It is compared with alternative reactive scheduling methods in an experimental design and significance of some factors are tested by analysis of variance tests.

Keywords: Reactive Scheduling, Time Critical Decision Making

ÖZET

HÜCRESEL İMALAT SİSTEMLERİNDE TEPKİSEL ÇİZELGELEME

Elif Görgülü

Endüstri Mühendisliği, Yüksek Lisans

Danışman: Yard. Doç. Selim Aktürk

Eylül 1993

Bu tez çalışmasında, bir gerçel zaman çizelgeleme problemi, makina bozulması durumunda yeniden çizelgeleme, ele alınır. Akış tipi işlik yapısındaki imalat hücresi için sezgisel bir yaklaşım geliştirilmiştir. Geliştirilen strateji ilk çizelgeyi bir noktada yakalayabilmek için başlangıç çizelgesinin bir kısmını yeniden çizelgeler. Diğer çalışmalardan farklı olarak amaç, zaman kritik karar verme kavramını kullanarak malzeme akışı ve satınalma gibi diğer üretim planlama kararlarıyla tutarlı yeni bir çizelgeleme yaratmaktır. Önerilen sezgisel yaklaşımda, farklı bir yeniden çizelgeleme stratejisi geliştirilmiş ve her makina üzerinde farklı yakalama noktası tanımlanarak yeniden çizelgeleme sırasındaki esnekliğin arttırılması yoluna gidilmiştir. Bu yaklaşım bir deney tasarımı içinde alternatif tepkisel çizelgeleme metodları ile kıyaslanmış ve seçilen bazı etmenlerin anlamlılığı varyans analizi ile test edilmiştir.

Anahtar Sözcükler: Tepkisel Çizelgeleme, Zaman Kritik Karar Verme

To my family and friends,

ACKNOWLEDGMENTS

I would like to thank my advisor Selim Aktürk who has provided a pleasing research environment and motivating support during this study.

I would also like to thank to my classmates Selçuk Avcı, Hakan Okan Balköse, Orhan Dağhoğlğıl, Mehmet Özkan, Haluk Yılmaz for their friendship and patience.

Finally, I would like to thank my parents, Kemal Gülal, Nalan Görgülü and everybody who has in some way contributed to this study by lending moral support.

Chapter 1

Introduction

Most of the research in scheduling considers environments with the assumption of fixed and known future conditions. In actual production systems, these conditions are seldom the case. Most of the time, an environment can be faced with disruptions like machine breakdown, unexpected new jobs, material delays, etc. Scheduling in real-time simply means that the system must respond to such disturbances or dynamic factors immediately as these events occur. It may be quite easy to construct a schedule, what is difficult is the schedule revision required by the dynamic environment. Given an initial schedule and a perturbing event, system is rescheduled to cope with the new conditions in real-time scheduling.

The original schedule is used as an input when planning other activities of the system like material flow and purchasing. During rescheduling, keeping the consistency with these decisions is important. An inconsistent schedule is going to be inapplicable because of the problems like the lack of material at the scheduled operation starting times.

Another point about real-time scheduling is that the computation should be completed in a reasonable amount of time. It is a time critical decision making process where the shop waits to receive the new schedule.

In this research, a real-time scheduling problem, rescheduling under machine breakdowns is considered. A machine breakdown forces the system out of the prescribed state rendering the preplanned schedule invalid. In this case,

the initial schedule is altered to compensate for this disruption.

Bean and Birge [Bean&Birge 85] proposed a theoretical framework that adopts the old schedule to smooth out the difficulties created by the disruptions and match-up with the preschedule by utilizing the Turnpike Theory [McKenzie 76]. This theory states that a schedule is exactly like a turnpike paralleled by a network of minor roads, if origin and destination are far enough apart, it will always pay to get on to the turnpike and cover distance at the best rate of travel, even if this means adding a little mileage at either end.

Their proposed strategy follows the preschedule until a disruption occurs and then reschedules part of the preschedule to accommodate the disruption. They reschedule to match-up in a future time, which is done in a way that the state reached by the revised schedule is the same as that reached by the initial schedule.

Our proposed strategy utilizes this match-up idea while the environmental conditions and the solution approach are completely different. It also differs in that it considers other planning decisions like material flow. We create a new schedule that is still consistent with the flow plan.

The match-up scheduling strategy has two basic decisions, determination of a match-up point and rescheduling up to the match-up point. We should approach to this problem heuristically because of the computational complexity of solving for both decisions simultaneously.

In the proposed heuristic approach, two objectives are considered cautiously, minimization of deviation from the initial schedule and developing a fast solution strategy. The deviation from the existing schedule should be as small as possible due to the restrictions caused by other decision levels in the hierarchy which are contingent to the results of the initial schedule. Match-up point is an important factor on the computation time, it should not be too far that makes the problem size large.

Model is developed on a manufacturing cell with the modified flow shop structure. Breakdown time is not known a priori but immediately after the event occurs, the down duration can be determined. There are no alternative machines in the cell, therefore an operation on a machine cannot be swapped

to another machine.

The organization of the thesis is as follows:

- Chapter 2 gives a literature review about the real-time scheduling problem. Studies are discussed under five main topics, including the match-up approach. The chapter ends with a general discussion about the advantages and disadvantages of these approaches.
- Chapter 3 deals with the statement of the problem. The points where the proposed approach differs from the current literature are listed and the problem is defined with its objectives and constraints.
- Chapter 4 details the proposed heuristic approach. Initially the underlying strategy is introduced, and then algorithms are listed in detail.
- Chapter 5 illustrates the experimental study. It includes two experimental designs which are computational comparison and the analysis of variance test.
- Chapter 6 concludes the thesis and discusses the contribution our proposed heuristic. Finally, prospects for future work are stated.

The proofs of the Dominance Rule and its transitivity property used in Chapter 4 are listed in Appendix A.

Chapter 2

Literature Review

Real-time scheduling is becoming an increasingly important area of research. Its importance is primarily due to the nature of the environment that the schedule is performed. To issue the dynamic and stochastic characteristics, different approaches have been proposed throughout the literature. After a general statement of scheduling problems in Section 2.1, the real-time scheduling problem is described in Section 2.2 with a classification on the approaches to the general form of the problem. They are categorized into five main topics, which are stochastic modelling approaches, simulation based approaches, artificial intelligence approaches, off-line scheduling and on-line control approaches and match-up approaches. The last class, match-up approach, is accepted by this research as well, which is discussed in detail in Section 2.3.

2.1 Scheduling

Scheduling is the allocation of resources over time to perform a collection of tasks [Baker 74]. Tasks are described in terms of operations each requiring certain amounts of specified resources for a specified time interval called processing times, the time at which they should be completed and the specified routings for resource requirements.

This problem is often complicated by a number of constraints like interference relations that make it infeasible to use the same required resource

simultaneously or precedence constraints which specify which operations must precede which other activities. Other constraints for starting times, due dates, non-preemption etc. increase the complexity and these interrelationships make scheduling problems very difficult even for a deterministic and static environment.

Besides this common definition, the scheduling problem can be specified by the machine environment, job characteristics and the optimality criterion that together define a problem type. In the scheduling literature, the introductory textbooks by [Baker 74] and [Kan 76] are the general references. The review of the production scheduling by [Graves 81] gives a brief summary of the topic. [Morton 92] with his book presenting the foundations of exact and heuristic methods is a useful guide for scheduling in different environments. The survey on the algorithms and complexity of sequencing and scheduling by [Lawler 89] review the complexity results, and optimization and approximation algorithms.

Scheduling problems belong to the class of combinatorial problems and one of the most difficult problems to solve. Scheduling problems are generally proved to be NP-hard, as stated by [Kan 76] and [Lawler 89], which means that it can be reduced to an NP-complete problem in polynomial time. The NP-hardness suggests that it is impossible to always find an optimal solution quickly. However it may still be possible to use an approximation algorithm to find solutions that are provably close to the optimum. Hence, most solution approaches reported in the literature are based on heuristics where optimality of the solution is not guaranteed.

Most of the scheduling algorithms reported in literature consider a static situation where the essential characteristics of the system during the scheduling horizon are fixed. The theoretical approaches of Operations Research and Artificial Intelligence to scheduling often are not applicable to the dynamic characteristics of the actual situation [McKay 88]. In the industrial world, job-shop scheduling is still a matter of question, even three decades after the first research steps. It appears that one research paper, that by Johnson, set a wave of research in motion that devoured scores of person-years of research time on an intractable problem of little practical consequence [Dudek et al. 92]. So a fruitless optimization effort has been spent for some period of time. Little has been changed in the theoretical world, and the underlying assumptions and

structure of the theoretical research have remained unchanged for 30 years.

There is an important fact that is commonly neglected; every scheduling problem varies with its own dynamic fashion and environmental conditions which make it unique. For example, one can model the available time of a resource as lasting forever. This assumption is unrealistic and each environment has its own available resource durations that are unpredictable and lacking of a specified pattern. Such events or conditions make up their own constraints and mostly not included by the scheduling systems that selects a schedule by manipulating a model of the real world, instead of the real world itself. Events in the real world change the assumptions where the model is based upon, so real-time scheduling becomes a fundamental requirement for scheduling.

In the following section, real time scheduling problem with the above extensions and related approaches are presented. According to their basic underlying ideas and assumptions, real-time scheduling approaches are classified into five main topics in Section 2.2. They are described in different titles such as Stochastic Modelling in Section 2.2.1, Simulation in Section 2.2.2, Artificial Intelligence in Section 2.2.3, Off-Line Scheduling and On-Line Control in Section 2.2.4, and finally and most concerned Match-Up Scheduling in Section 2.2.5.

2.2 Real-Time Scheduling Problem

Variabilities in the production environment and modeling limitations result in operational deviations from schedules generated using predictive models. Scheduling systems select a schedule assuming that the conditions remain unchanged with the passing time. The challenge of stochasticity arises from inevitable mismatches between the model and the reality [Parunak 87]. Though most of the environments are dynamic, conventional models in scheduling problems are usually static. In static problems, all the jobs to be scheduled are available at the beginning of the planning interval; no legacy of work remains from earlier periods, and there is no possibility of deferring any portion of the work into a later work [Conway 91]. Therefore static models are more tractable than dynamic models, they have often captured the need for more complex,

dynamic systems. In principle, dynamic systems seem to more closely represent the real production world, except that in practice, time in that world is often artificially partitioned into distinct intervals [Conway 91]. Analysis of static problems has frequently uncovered valuable insights. There are heuristic principles that are useful in more general situations.

Similarly most of the scheduling models are based on the assumption that all problem data is known in advance though this may not be a valid assumption for most of the cases. Disruptions like machine breakdown, unexpected new jobs, rush or hot jobs with high priorities, material delays, change in release dates, tool unavailability, fluctuations in processing times are possible reasons introducing stochasticity to the system [Rodammer]. Scheduling in real-time simply means that the system must respond to such disturbances or dynamic factors immediately as these events occur.

What is meant by 'respond' is important. [Conway 91] states the theoretical problem as declared below:

Given a schedule S and a single perturbing event E , produce a rescheduling $R(S,E)$ that yields a new schedule S' .

This statement calls to mind a frequent comment heard in many scheduling shops:

There is no scheduling problem but rather a rescheduling problem.

It may be quite easy to construct a schedule, what is difficult is the constant schedule revision required by the dynamic environment [Graves 81].

An important challenge in rescheduling is that the run-time of a rescheduling method should end up as another disturbance. Another limit is on how long can the shop wait to receive the new schedule. This period should be short enough so that it will not require any special strategy to be followed during the run-time of rescheduling procedure.

The real-time scheduling problem aiming to search for characteristics of schedules compatible with the main manufacturing constraints to be satisfied

is especially very important for planning purposes. All the good decisions made at the upper levels will be inefficient if the shop-floor level is not able to execute them because of disturbances. To take into account of these disturbances, the production plans proposed to the workshops need to include many degrees of freedom of various types such as large flow times including time margins, underloading of the technical resources to take account of machine failures, etc. Consequently on the shop floor decision level, these degrees of freedom are available to react in real-time to the various disturbances without modifying the proposed production plan. Thus there exists a direct connection between the amount of degrees of freedom necessary to react to disturbances and the efficiency of the real-time scheduling procedures. The greater this efficiency is, the lower the necessary degrees of amount of freedom will be. But it must be noticed that the flexibility associated with these degrees of freedom is costly like high level of in process inventories, under loading of the resources, etc. Thus it is very important to be able to reduce the requested flexibility by using efficient real-time scheduling procedures on the shop-floor level [Erschler 89].

Real-time scheduling can be performed in two distinct ways. An off-line schedule can be developed that will help real-time scheduling by giving guarantee as to the insensitivity of the schedule to future information or a reactive approach alone applicable to any arbitrarily off-line schedule is developed. Because of the trade-off, using off-line schedules with costly flexibilities would not be preferable. Instead of pure strategies, a combination of these two basic ideas exist in the literature in different forms.

The main strategies for dealing with uncertainty of the environment that could be reached by the researcher are categorized into five distinct classes:

1. Stochastic Modelling Approaches,
2. Simulation Based Approaches,
3. Artificial Intelligence Approaches,
4. Off-Line Scheduling and On-Line Control Approaches,
5. Match-Up Approaches.

Each strategy is discussed in detail by the following subsections.

2.2.1 Stochastic Modelling Approaches

It is the scheduling model that reflects the uncertain nature by common statistical distributions with available information. [Pinedo 83] worked with random release and due dates, and [Lawler 89] studied on single machine scheduling with random breakdowns. These approaches give only an approximate solution for real-time case because of idealized distributions and rare events. The results are for relatively small problems. Research in this area is scattered and is not very encouraging. There is a great need for new mathematical techniques useful for simplifying the derivation of results in this area.

2.2.2 Simulation Based Approaches

[Davis&Jones 88], [Davis&Jones 89] proposed a scheduling methodology for real-time scheduling of operations on a stochastic job-shop. They solved the scheduling problem with a hierarchical decision making control architecture based on the decomposition approach of mathematical programming. They have the tools of real-time simulation and mathematical decomposition, and examined a two level scheduling control technique. Each level has responsibility for solving its own scheduling problem subject to constraints by its supervisor. The higher level scheduler (the supremal) would specify the earliest start time and the latest finish time for each task while the lower level scheduling modules (the infimals) would refine these limit times for each task by detailed scheduling of their assigned activities. Evaluation of the scheduling rules by supremal and infimals is the task of an on-line, distributed simulation package.

The on-line real-time simulation methodology is used to analyze several candidate scheduling rules in order to solve both infimal and supremal problems. The solution to the problem is composed of both production planning and control. The planning elements include simulation, the selection of evaluation criteria and scheduling rules. Control function generates event lists and coordinate them in presence of conflicts. The event lists are the input to the supremal units which are then sorted into a master schedule, a scheduling list and a process list.

The repetitive task of identifying the next action to be taken in a discrete

manufacturing system serves the real-time scheduling process since decisions are dynamic. Most commonly used dispatching rules such as Shortest Processing Time (SPT), Earliest Due Date (EDD), and First In First Out (FIFO) are used to order jobs in other simulation directed scheduling attempts. After disruption, the job at the beginning of the list is processed first. These rules have the advantage of being easy to understand and to implement. On the other hand, these rules have had little analytical studies, thus their efficacy is hard to predict theoretically. Their performance is highly dependent on the examined environment.

Another control mechanism is developed by [Wu&Wysk 89] which dynamically varies the implementation of job dispatching heuristics based on the simulated information. Discrete simulation has been extensively used for testing dynamic dispatching rules. An accepted recognition among researchers is that a combination of simple dispatching rules, in many cases work better than individual dispatching rules.

Disadvantage of using such rules is that they do not deal with macro conditions, they are myopic and can lead to substantial error. Also using simulation to produce schedules is costly, both in the computer time used to generate the schedules and human modelling effort required to design and run the simulation model.

2.2.3 Artificial Intelligence Approaches

These approaches are logic based and draw upon computer science techniques with an attempt to automate the decision making process to replace human intervention in an automated production environment. These approaches utilize heuristics and intend to increase the speed of decision making process with less emphasis placed on optimality.

A family of knowledge-based scheduling systems, ISIS/OPIS/CORTES have been developed by the researchers at the Carnegie Mellon University for automatic scheduling that provides a framework for incorporating the full range of real-world constraints.

Their design is focused on two basic points [Fox 90]:

1. Constructing a knowledge representation that captures the requisite knowledge of the job shop environment and its constraints to support constraint directed search.
2. Developing a search architecture capable of exploiting this constraint knowledge to effectively control the combinatorics of the underlying search space.

ISIS/OPIS/CORTES methods are capable of reactively rescheduling jobs in response to disruptions that might occur as well. The chronological order of their evolution and performances are:

- ISIS 1 : Constraint guided scheduling,
- ISIS 2 : Hierarchical constraint guided scheduling,
- ISIS 3 : Multi-perspective scheduling,
- OPIS 1 : Opportunistic scheduling,
- OPIS 2 : Reactive scheduling,
- CORTES : Network-based constraint optimization.

ISIS model is a job centered scheduling system while OPIS dynamically switches between being job centered and machine centered. CORTES system has an operation centered view of scheduling. A more detailed discussion on these methods can be found in [Smith 92].

Detailed discussion on the opportunistic knowledge-based systems is presented in [Smith 87] , [Pape&Smith 87] , [Ow&Smith 88] , [Ow et al. 88] , [Smith et al. 90] , and [Smith 92].

The most interesting part of these procedures related with this study is the reactive plan revision. During constraint-based schedule repair in OPIS, five strategic schedule revision alternatives have been used:

- Order Schedule (OSC) - the contiguous part of a given order's production plan is revised by using the beam search which is utilized as a search technique of ISIS,

- Resource Scheduler (RSC) - a designated resource is revised by dispatching rules assuming that dealed resources have high contention so requires efficient utilization,
- Right Shifter (RS) - execution times of operations are pushed forward in time such that no time or capacity conflict is created,
- Left Shifter (LS) - similarly execution times of operations are pushed backward if possible so that no time or capacity constraint is going to be violated,
- Demand Swapper (DS) - pairwise interchange of two operations in the order schedule is performed if this causes any improvement in the total tardiness.

These five strategies are not used arbitrarily but the characteristics of the case like the existence of whether a time or capacity conflict avoidance, or choice of a resource-based or order-based optimization method are active during determining the appropriate technique or techniques to be chosen for reactive scheduling purpose. According to the current conditions, the strategy needed by the disrupted schedule varies and this is tabularly formulated in [Ow et al. 88].

2.2.4 An Off-Line Scheduling and On-Line Control Approach

[Odrey&Wilson 87] developed a controlled structure that is composed of a planning and a control methodology. Their work follows for a goal directed decision hierarchy proposed by NIST in Automated Manufacturing Research Facility (AMRF). A production control hierarchy possesses several control levels, and scheduling problem is addressed at the cell and workstation levels of the hierarchy. Off-line methods address the issues of controller design in a discrete event environment while on line methods focus on the regularity actions in real-time. Their overall approach is based on the mathematical decomposition and simulation similar to [Davis&Jones 89].

Their proposed system is initially in a steady state and remains in this state until a failure type disturbance (e.g. machine failures) or a load type disturbance (e.g. hot orders) occurs. Then the system enters a transient phase and the developed on-line control module generates an interim scheduling policy to prevent the accumulation of in-process inventory till the broken machine become operational, and then the system enters the transient phase. When the system enters the transient phase the control module enforces a set of operational strategies to bring the system back to the steady-state periodic schedule as quickly as possible [Odrey&Wilson 90].

The basic assumption with this study is that a rolling environment is used on the manufacturing cells with cycle times. Cycles have identical machines, jobs and operations so repeat themselves when everything is certain. However stochasticity does not allow this routine scheduling. A disruption that affects one cycle, when not compensated will be dangerous for other cycles. When match-up is not feasible, effects of disruption will increase dramatically as each cycle begins with a disruption caused by the delay in the previous cycle.

The off-line scheduling model part of this approach uses an optimization technique while on-line control structure utilize a hybrid approach of combinatorial optimization and AI techniques.

In order to accommodate all the parts that still need to be processed while returning to steady-state, the earliest start times of machines may be shifted L time units. The objective is keeping L as small as possible, even though its resulting effect on the rest of the schedule is not evaluated. A detailed description exists in [Saleh 88] and [Saleh et al. 91].

2.2.5 Match-up Scheduling Approach

Bean and Birge proposed a theoretical framework that adopts the old schedule to smooth out the difficulties created by the disruptions and match-up with the preschedule [Bean&Birge 85]. They employed Economic Turnpike Theory [Winston] by McKenzie by adopting the approach used in economics to the real-time reactive scheduling. Turnpike Theory in [McKenzie 76] is used in a growth model which is described as,

It is exactly like a turnpike paralleled by a network of minor roads. There is a fastest route between any two points; and if the origin and destination are close together and far from the turnpike, the best route may not touch turnpike. But if origin and destination are far enough apart, it will always pay to get on to the turnpike and cover distance at the best rate of travel, even if this means adding a little mileage at either end.

Analogous problems arise in the number of areas in economics. Turnpike Theory develops formal conditions under which returning to an original plan is provably as good as re-planning to the original objective. When these results are applied to the scheduling area, under a wide range of real-time conditions, the expensive approach of rescheduling to the original objective can be replaced by the much simpler problem of finding a way back to the original schedule.

Starting from this point of view, Bean and Birge used Turnpike Theory as a foundation for adaptive approaches. However this theory has strong assumptions that cannot be validated by every scheduling problem:

- The strongest turnpike results require uniform convexity of the incremental cost even though most measures of scheduling do not satisfy this assumption.
- Planning horizon must be long enough since turnpike results are asymptotic. This assumption is justified in the scheduling case in the limit as the horizon of the problem is increased.
- Preschedule is optimal.
- Penalties and setups are charged continuously which means in scheduling application, jobs could be preempted.
- A known upper bound for the number of jobs in the system is required by the theorem. However also the problem is considered as a discrete time, infinite horizon optimization problem which means there is infinite number of jobs in the system.

For a FMS system, real-time scheduling is very critical and its production environment better satisfies the above assumptions relative to other systems.

Therefore FMS is preferable for applications of turnpike results as performed by [Birge 85].

The assumptions become close to accurate since FMS has job loads that consist of many small pieces and setups are short. This will be advantageous to satisfy the assumptions of convex cost function and continuous payment of penalties and setups.

[Bean&Birge 85] declares that their heuristic approach does not guarantee an optimal solution in all situations. Though match-up point, T , can be chosen large enough to get close to an optimal schedule, it is not generally known how large T must be. A large T which is known to be leading to computational difficulties, makes the method impractical. Given these potential problems, algorithm is implemented as a heuristic, and T is determined to balance error and effort.

The method results as good as complete rescheduling as the rescheduling horizon is lengthened and the interval between disturbances increases. This is shown by economic Turnpike Theory. The error involved in using the method is also bounded by the difference between the match-up cost and a lower bound which is found by a local minimization. They suggest using comparison of the error bounds for different match-up problem sizes to determine the value of additional computational effort in specific situations.

Match-up real-time scheduling algorithm (MUR TSA) has three basic steps:

- Define new internal cost parameters.
- Determine the portion of the preschedule to release.
- Reschedule the released portion to minimize cost.

These steps are repeated until match-up is performed or some stopping criteria is satisfied. Stopping criteria can be the maximum planning horizon or an upper bound for allowable cost. Tardiness is taken as the unique cost measure dominating others.

In Step 2, the portion of the preschedule to be released is chosen. The objective in defining the new problem set is to allow for a smooth transition

back to the preschedule turnpike. The problem should be large enough to allow for sufficient adjustments to reduce disruption effects, but it cannot be too large for the scheduling algorithm.

The problem size is two dimensional, the pool of machines included in match-up and the length of the planning horizon. During choosing the machine pool, utilization of machines, their reliability, the slack times of operations on these machines and measures of the criticality of the pool information is involved. The concern of the length of the horizon is directly related with the number of operations to be included in the reschedule.

Schedule maker in Step 3 can determine whether additional problem sets and cost structures need to be defined. Methods like heuristic ordering using dominance rules, simple interchanges to find local optima, branch and bound procedures that can find feasible solutions quite easily and simulated annealing are proposed by [Birge 85] to improve the feasible schedule, if possible.

A more detailed algorithm called MUSA is given by [Bean et al. 91]. The planning horizon is explicitly searched beginning from an initial match-up point, T_1 . When the cost for the given match-up point exceeds the upper-bound, the match-up point is incremented by ΔT , until t exceeds a maximum value. Feasibility not only defines interference and precedence constraints but also upper bound for total cost and match-up point. If the current solution results in excessive tardiness cost which means the observed cost is greater than the threshold value, and the maximum match-up point, T_{max} , which is the ending time for the initial schedule, is exceeded then the system proceeds to Step 1 after enlarging the problem.

The initial rescheduling attempt is on a single machine, the disrupted one. Six different rules are selected for inclusion into MUSA which are SPT, EDD, Modified Due Date (MDD), A Priority Index (API), a ratio rule and the ordering based upon the current sequence. The heuristic calculates six feasible schedules based on these ordering rules and chooses the least cost schedule that is obtained. When scheduling attempt fails to result a feasible solution, machine number is increased and in the following steps multimachine lot reassignment is used to redistribute lot-to-machine assignments. They examine two approaches for reassignment; a multiple choice integer program (MCIP) formulation solved using the technique of [Bean 84], and a priority rule dynamic

assignment heuristic.

Experimental results are encouraging for the problems with real production data supplied by an automotive manufacturer. The results support the theory by showing that match-up scheduling costs are close to lower bounds. Furthermore the results were significantly better than results from pure static and dynamic strategies like pushing back the static schedule, dynamic priority rules and MCIP that are commonly used in practice with the exception of the third one with less common use..

A brief comparison of the models, their strengths and weaknesses are discussed in the next section to be used as a guideline throughout the current research.

2.3 Remarks

Scheduling models that try to beat unpredictable events in the manufacturing system differ in their approaches as classified in the previous section. Strategies developed for finding solutions to undesired disruption effects on the preschedule have two trends:

1. **Just-In-Case Efforts** : A structural solution is looked for that will minimize the troubles caused by stochastic events. The possibility of future disruptions is considered within the initial schedule by adding some extra flexibility. Deciding on the amount and type of flexibility is an hard job because of the existence of two random variables in the problem. These are interarrival times of disruptions since disruption definition is general so there are many reasons independent of each other causing a single disruption, and the duration of disruptions because it depends on the reason of the disruption. This type of an effort is expensive though not powerful. It keeps some additional resource (e.g. machine, material) for emergency use, so cause additional cost, increased makespan and low utilization to prevent an activity that may or may not occur during the given time period.

2. **Reactive Efforts** : An operational solution is used that creates solutions whenever a disruption occurs, not any time before. When the exact time and duration of the event is known, some action is taken to include it in the plans. Current flexibility by the initial solution is used to compensate this unexpected effect so will not be so costly as the first one, however would require much more effort.

The reactive approach dominates the just-in-case efforts that it is applicable to a wider range of disruptions and do not require extra resource that might be a waste. This research also develops a reactive method to deal with the real-time events. Artificial Intelligence is one of the tools used to solve the problem in a reactive manner. The proposed methodology by Smith et al. is utilized for both generating schedules and reacting to real-time events. A long-term scheduling does not exist since the existing schedule is updated with an event not necessarily a disruption but can also be completion of any operation within the regular time. Therefore there is no worry for matching-up a specific predefined schedule so no hard constraints to keep. It has a decision tree like structure to choose the appropriate action according to the conditions of the problem which is important for taking a feedback. Separating the disruption effects into categories and trying to formulate the specific situations of each class is a credible approach.

One of the approaches, by Odrey and Wilson, uses the match-up idea of Bean and Birge, which is the underlying concept of this research as well. However they allow the earliest start times of machines to be shifted by L units [Saleh et al. 91] with an objective of keeping this L minimum. They have a repetitive production assumption. Another assumption is that failure times should be smaller than cycle time. Any failure is tried to be compensated within the cycle it occurs and if not possible, L units of delay is permitted in the next cycle. The next cycle takes this delay as a failure so similarly tries to compensate it in its cycle time. This attempt continues in this fashion until in one of the cycles a schedule that will cause no delay for the following cycle is found. This repetitive attempt increases the nervousness of the system since the methodology has no stopping event when the delay is not compensated by one of the cycles, so in such a case, an endless effort continues.

This research, by the basic assumption, differs from Odrey and Wilson's work that it does not restrict the system as having a repetitive manufacturing. It is more general so theories and tools used by them becomes inappropriate for this study's problem definition.

To a greater extend, the approach by Bean and Birge is found to be more appropriate and followed throughout this study. Their idea is so basic that also included by other researchers. In Section 2.2.5 the basic three steps of their algorithm has been summarized. There are some lacking points in their algorithm that need further research which are as follows:

1. Bean and Birge, in their research, build a close relation between the Turnpike Theory and match-up scheduling to conclude that match-up policy is optimal or if some assumptions of the theory cannot be satisfied by the schedule, they find error bounds of their algorithm. Turnpike Theory, as stated in Section 2.2.5, has very strong assumptions that are impossible to satisfy for its application to the scheduling problem, even for FMS as in [Birge 85].
2. In Step 2, the portion of the initial schedule to be rescheduled is determined. For this purpose, a common T for all machines in the pool, as a match-up target, is determined. However, this approach has certain disadvantages such as:
 - i) Machines have different flexibilities (e.g. idle periods, slack of jobs on these machines) so this common T may not be equally appropriate for all of the machines.
 - ii) T , when taken same at all machines, will not be always at the beginning or end of an operation. Matching somewhere during processing a job gives no aid when preemption is not allowed.
 - iii) Choice of operations in the rescheduling pool is a consequence decision of choosing the T value. For that reason, having any operation in the pool will not be sufficient to include the other operations of the same job on the pool. When this is the case, slack of jobs, that gives additional flexibility needed to match-up, cannot be used on purpose but it will rather be an uncontrollable variable.

3. Scheduling horizon is searched by ΔT increments in order to find an adequate match-up point in Step 2. This incremental analysis success highly depends on the choice of ΔT . Additionally, this amount of increment is always fixed, no matter how much flexibility exists in the system or required to compensate the disruption at the moment. At the end of Step 3, when scheduling is found to be infeasible with the current pool, a feedback about the amount of extra flexibility that seems would lead to a successful rescheduling, can be given to be used in Step 2 during enlarging the pool size.
4. Bean and Birge mentioned that their approach has an intuitive appeal since material flows in the system are planned based on the preschedule [Bean&Birge 85]. Not only the material flow but other decisions like planning transportation tools, scheduling preceding or succeeding shops also use the initial schedule as an input. When some portion of it is redesigned, these decisions are all affected by this change. There are two possible actions. They are updating these consequence decisions with respect to the results of the reschedule or considering the effects of the changes in developing the reschedule. The primer alternative may cause a need for an update at the upper levels of the hierarchical architecture. There might be other decisions affected by these subsequent ones also. This will cause a dramatical increase in the dimensions of the problem so it will require too much effort. The latter alternative suggests adding new constraints during match-up that will represent the limitations caused by these consequence decisions. This attempt will restrict the feasibility space so might cause a harder problem but will be more efficient.
5. Step 3 deals with rescheduling after the dimensions of the problem is defined in Step 2. For rescheduling purpose, classical scheduling methods are proposed. These approaches are not always good enough for the match-up purpose since the rescheduling problem has its own constraints and conditions different than the general scheduling problems. For example, there are two kinds of due-dates which are job due-dates and match-up times. Their importance are not equal, for example a violation of the match-up point will result in the enlargement of the pool and another rescheduling try, while exceeding the due-dates only is acceptable. Therefore finding a feasible schedule within the pool is very important, and a good schedule is the one that produces feasible schedules with

smaller pools, smaller flexibilities, not the one that gives the minimum cost by increasing the number of trials so the computation time. In most cases, being unable to find a match-up schedule is not because of the insufficient flexibility in the pool but because of inefficient scheduling algorithm. So it deserves attention that is lacking in the previous studies.

All these aspects are reexamined and alternative strategies are developed based on these above points that are thought to be deficient. The following chapter gives a complete definition for the model with the assumptions valid in this study and the formulation of the problem.

Chapter 3

Problem Statement

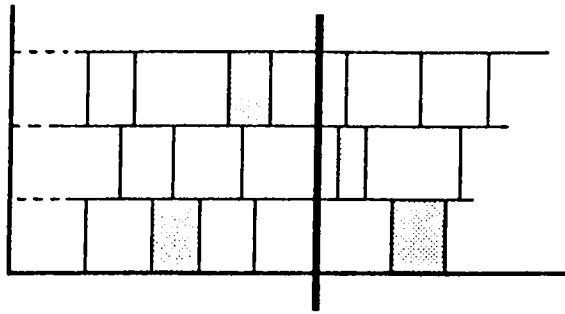
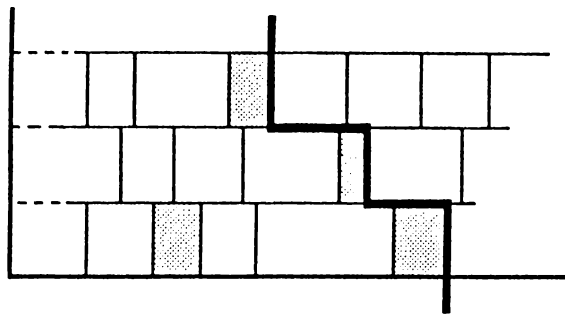
Although remarks about the past research are given at the end of the previous chapter, the dominant points that are emphasized by this research are outlined in Section 3.1. Assumptions are listed in Section 3.2, which will be in use in the remaining part of this thesis. Finally, objectives and constraints are discussed in Section 3.3 within this chapter.

3.1 A Different Approach To The Problem

Scheduling in real-time is an important problem that has attracted a lot of interest by researchers. Match-up perspective which tries to find a way for real-time scheduling has been worked on by Bean and Birge as summarized in the Literature Review. Their main idea of matching-up the initial schedule by rescheduling only some part of it when the whole schedule becomes infeasible to apply has given the basic motivation for this research.

The proposed methodology departs from the previous studies in the following basic points :

1. Instead of having a common match-up time T on all machines like Bean and Birge, match-up times unique to each machine is considered. This is needed since matching-up the schedule during processing a job has no meaning when preemption is not allowed. Gantt Charts given in Figures

Figure 3.1. Same T^M for all machinesFigure 3.2. Machine specific T^M

3.1 and 3.2 show both cases. As seen in Figure 3.1, some part of the job takes part before T and other part after T . Having the first portion in the subproblem will give no flexibility since its location on the chart is fixed, moreover it cannot be scheduled at any other time.

Another advantage of having different T_i values for machines is that, whenever an idle period exists in the initial schedule, it can be included by the subproblem independent of any other machines. Having more idleness at the reschedule means more flexibility during match-up. Location of idle times on the Gantt Chart are assumed to be random, so a common T that gives more idleness for one machine may not result so efficiently for another machine. Separate T_i values will let each machine to look for its own benefits and then the subproblem combined with all will be more flexible than the one determined by a unique T . Here, flexibility of a subproblem is used to describe a case of having tighter constraints.

2. A match-up point is looked for by trying different time points until one gives a feasible solution for the pool defined by that point. A good search algorithm should result fast with less effort. Trying to find a

feasible solution at each ΔT increment is not very powerful and does not satisfy the above requirement of having distinct T_i values as well. The difference in this research is that during developing a valid methodology, instead of rescheduling for fixed intervals, feedback is taken from the previous scheduling attempt to minimize the search effort. The amount of flexibility versus complexity added to the rescheduling problem with the new match-up point is considered for a healthier decision. What makes real-time scheduling critical is the time parameter. Real-time systems should be not only reliable but also have the time-critical decision making ability. Since match-up point is found by trying different alternatives, the number of alternatives should be limited. For that reason it is proposed that the enlargement amount should be determined by taking feedback from the type and quantity of the opportunities, and conflicts in the system instead of having a fixed amount of enlargement, ΔT , in every iteration.

3. Schedule in a manufacturing cell is the main concern but neglecting its interaction with other cells or any other elements of the manufacturing system is not very realistic. Some of these relations with the surrounding environment can be as follows:
 - **Preceding and Succeeding Cells** : Jobs that require more than one cell need feedback between these cells' schedules. For an interrupted cell, the preceding cell's completion times affect the earliest start times and the succeeding cell's beginning times affect the latest possible start times of jobs. When rescheduling the subproblem, the initially existing schedule's beginning and ending times become invalid. Therefore new beginning and ending times should be determined such that they would not violate these neighbor cells' schedules.
 - **Material Flow Into Cell** : During developing the material requirement plan throughout the system, the outputs of the initial schedule were used. When some part of is redesigned, the original material flow plan might be no longer available to apply the updated schedule. So, rescheduling should be performed by considering its validity.
 - **Other Resources** : Transportation devices, jigs, fixture, etc., which

are shared by other utilities and not considered directly at the schedule are all planned according to the results of the initial schedule. These interactions, similar to the first two factors, also put restriction to rescheduling.

Because of these stated interactions, rescheduling attempt might be useless when its results divert too much from the initial schedule's results. For that reason, an objective of minimizing the deviation from the previous schedule is defined in order to keep the new scheduling results applicable by other decisions which use it as an input. Our objective will be carried out by both hard and soft constraints, which will be introduced in the following sections.

4. Known scheduling heuristics and dispatching rules had been used for rescheduling purposes, even though a new rescheduling methodology might perform more efficiently by considering the specific constraints of the problem. Special interest is given to creation of the rescheduling policy in this research and a major portion of this study is devoted to develop one.

In the light of these four points, a methodology is developed with the purpose of finding a solution to the problem being stated up to now. All related assumptions and definitions are given at the following sections of this chapter.

3.2 Assumptions

To make the problem more clear and well defined, an environment that represents the real system is build up with the following assumptions:

1. A cellular manufacturing system is concerned. A schedule means a production plan at the cell level and the manufacturing cell is of a moderate size due to nature of the cellular manufacturing systems.
2. The cell has a modified flow-shop structure. A modified flow-shop falls between a job-shop and a pure flow-shop. Parts can enter or exit at any machine, as in job-shop, but follow a uni-directional flow, as in flow-shop, though they can skip some machines.

3. Any disruption is formulated as an additional job with a given starting time that cannot be predicted and a known duration. If the reason for disruption is the lack of a resource, necessary time to cover it is assumed to be known.
4. Disturbances have minor durations or idleness in the initial schedule is enough to cover this period.
5. No preemption is allowed, once an operation is started, it is continued until finished.
6. Setup times are independent of schedules and are included in processing times.

When compared with the work of Bean and Birge, it will be noticed that Turnpike theory is not strictly followed by assumptions like the optimality of preschedule or preemption. Instead, the practical value of the theoretical results motivates their approach to real problems. Nevertheless, it is the study that first uses the insight of the Turnpike Theory, and it also gives incentive to our research. The model built in this work, with its objectives and constraints, is given in the next section. It has the underlying idea of matching-up the initial schedule as proposed by Bean and Birge.

3.3 Model Formulation

3.3.1 Objectives

The main purpose of this study is to develop a heuristic that will reschedule the system to match-up the initial schedule which became invalid after a disruption. During searching for a solution, there are two important ambitions:

1. The deviation from the existing schedule should be as small as possible due to the restrictions caused by other decision levels in the hierarchy which are contingent to the results of the initial schedule. Two types of deviation are considered that are most expected to cause trouble because of the interactions stated in Section 3.1:

- i.) **earliness** : new beginning time can be smaller than the beginning time at the initial schedule,
- ii.) **tardiness** : new completion time can be greater than the completion time at the initial schedule.

Both cases are undesired at the reschedule. For example, when a job is rescheduled to an earlier time, the material required to process it may not be ready or if it is rescheduled to a later time, it may cause a delay in succeeding activities.

This work completely eliminates earliness by taking initial beginning times of jobs as hard ready time constraints. However, tardiness cannot be strictly restricted. For a job whose scheduled processing period overlaps with the down duration of the machine, even scheduling it immediately after the down duration may not be enough to prevent tardiness unless it has a certain amount of slack. In such cases, there is nothing to do except letting the job to be late but trying to minimize the amount of deviation. Tardiness is minimized by taking the initial completion times of jobs as soft due date constraints.

2. The heuristic that searches for a solution is operated in real-time that a minute of computation time means a minute of system idleness. Then, it is a time-critical decision making process which means the solution should be found as fast as possible.

The run time of the solution procedure depends on the match-up time searching attempt. T^M should not be too far that makes the subproblem size large and completion time long, but also it should not be too soon that the reschedule will be a fruitless effort and another replication will be required.

This objective is considered with two actions included by the procedure:

- i.) An efficient strategy to suggest reasonable T^M values is looked for.
- ii.) Computation time is limited with an upper bound which means that if it cannot produce a match-up schedule in a given time, it terminates and gives the currently best schedule on hand that has been developed till this point.

Both objectives are inherent and are closely followed in the heuristic algorithm developed for this problem, which will be discussed in Chapter 4. Additionally there are some constraints that state the problem. They will be listed below after calling the notation that is used in this study.

3.3.2 Notation

The notation used in the mathematical formulation and throughout this chapter is given below:

n	:	number of jobs in the match-up pool,
m	:	number of machines in the cell,
$p_{i,k}$:	processing time of job i on machine k ,
$q_{i,j,k}$:	$\begin{cases} 1 & \text{if operation } j \text{ of job } i \text{ requires machine } k, \\ 0 & \text{otherwise.} \end{cases}$
T_k^B	:	beginning time of schedule for machine k ,
T_k^M	:	match-up time of schedule for machine k ,
k_i^f	:	machine number on which job i enters the cell first time,
k_i^l	:	machine number on which job i leaves the cell,
L_i	:	tardiness of the last operation for job i ,
M	:	a very large positive number,
$X_{i,k}$:	starting time of job i on machine k ,
$X_{i,k}^{old}$:	starting time of job i on machine k in the initial schedule,
$Y_{i,h,k}$:	$\begin{cases} 1 & \text{if job } i \text{ precedes job } h, \text{ not necessarily directly, on machine } k, \\ 0 & \text{otherwise.} \end{cases}$

$$i, h = (1, \dots, n) \text{ and } k, j = (1, \dots, m)$$

$X_{i,k}$ and $Y_{i,h,k}$ are the decision variables of the model, and L_i is the consequence variable.

3.3.3 Constraints

There are three groups of constraints considered by the approach developed here:

1. Classical modified flow-shop constraints are still active which are:
 - i.) precedence relations,
 - ii.) non-interference constraints.
2. Match-up constraints are imposed to have the ending state of the reschedule at the match-up point identical to the state of the initial schedule at this time point.
3. The first objective of minimizing the deviation from the initial schedule adds the ready time constraints since earliness is taken as a hard constraint.

During rescheduling with a chosen match-up point, all these restrictions should be considered by the problem solver.

Assuming that T_k^B and T_k^M values have been already determined, the rescheduling problem can be formulated as a mixed integer program.

3.3.4 The Mixed Integer Model :

1. One of the global objectives is defined as minimizing deviation from the initial schedule which is considered by both hard and soft constraints. The soft constraint part, minimizing tardiness, is the objective of this MIP. As previously defined, tardiness is the delay in the completion times of the initial schedule that may be caused by shifting some jobs to the right in the planning horizon for rescheduling purpose. The formal objective definition is:

Min

$$z = \sum_{i=1}^n L_i$$

where

$$L_i \geq X_{i,k_i^f} - X_{i,k_i^f}^{old} \quad \text{for } i = (1, \dots, n).$$

2. Two classical scheduling constraints of modified flow-shop are still in use:

i.) precedence constraint which satisfy that a job can be processed on at most one machine at any time and define the process routings,

$$\sum_{k=1}^m q_{i,j,k} (X_{i,k} + p_{i,k}) \leq \sum_{k=1}^m q_{i,j+1,k} X_{i,k},$$

for $i = (1, \dots, n)$ and $j = (1, \dots, m)$.

ii.) non-interference constraints which satisfy that each machine can process at most one job at any time,

$$X_{h,k} - X_{i,k} \geq p_{i,k} - (M + p_{i,k}) (1 - Y_{i,h,k}),$$

$$X_{i,k} - X_{h,k} \geq p_{h,k} - (M + p_{h,k}) Y_{i,h,k},$$

for $i, h = (1, \dots, n)$ and $k = (1, \dots, m)$.

3. Since another objective is to minimize deviation from the initial schedule which is stated as a hard constraint, no earliness from the beginning times of the initial schedule are allowed,

$$X_{i,k_i^f} \geq X_{i,k_i^f}^{old} \quad \text{for } i = (1, \dots, n).$$

4. Beginning time and match-up time of the reschedule are hard constraints which restrict the time interval to perform reschedule,

$$q_{i,j,k} X_{i,k} \geq T_k^B,$$

$$q_{i,j,k} (X_{i,k} + p_{i,k}) \leq T_k^M,$$

for $i = (1, \dots, n)$ and $k, j = (1, \dots, m)$.

Beginning time constraint states that, since disruption is not foreseen, jobs processed previously cannot be rescheduled. Match-up time constraint brings the system to a state at which initial schedule is applicable again.

This model is helpful only for defining the rescheduling part of the approach. It is not directly applicable because:

i.) The match-up point, T_k^M , has to be known prior to formulation. However, it cannot be determined without enumeration. Correspondingly, the set

of jobs that will be included by the MIP model cannot be determined while T_k^M is not known.

- ii.) The second objective of this approach is to develop a time-critical decision making process which is difficult to get because of the computational complexity of mixed integer programming.

A heuristic methodology is developed to solve the problem stated in this chapter. It consists of three parts:

- determination of the job pool and match-up point consequently,
- rescheduling within the given pool,
- enlarging the pool size when match-up effort fails.

The following chapter includes heuristic algorithms developed for the three basic steps stated above.

Chapter 4

Match-up Algorithm

The statement of the problem with its assumptions, objectives and constraints is presented in Chapter 3. In the light of the stated remarks on other researches related with the problem, a new heuristic approach is developed. This proposed solution strategy decomposes the problem into three subproblems and apply different methodologies for each of them. In Section 4.1, the overall approach is discussed. Rescheduling methods for each decomposed part and determination of match-up points are explained in Section 4.2 and 4.3, respectively.

4.1 The Strategy

The problem and the system that owns it is defined in the previous chapter. It is a complex task to solve such a problem because of the two decisions that are closely related but cannot be solved together. They are:

1. Finding a feasible schedule within the given match-up time.
2. Finding a match-up time for which a feasible schedule exists.

The procedure is formulated to search for match-up points on the time horizon, and for a chosen point, feasibility is checked by solving the proposed rescheduling algorithm.

Different points on the time horizon are tested in order to find a match-up point. A pool is created by collecting some of the jobs and all of the machines of the cell for this rescheduling interval. A heuristic, with an output of either a feasible solution or a new match-up time to be used in the next iteration, is applied. This attempt continues until either a feasible schedule is found or the limit is reached. Limit can be either the ending time of the initial schedule, which means when reached, a complete reschedule has been tried, or it can be the user defined computation time to end the try, which is one of the important elements in real-time scheduling. The steps of the proposed heuristic approach can be represented as:

- S1 Find an initial rescheduling job pool with a match-up point.
- S2 Apply the rescheduling heuristic on the pool.
 - If a feasible solution is found, STOP.
 - Else, go to STEP 3.
- S3 Update the match-up point and the job pool according to the result of the rescheduling heuristic.
 - Go to STEP 2.

This sequence of actions are represented in a flow diagram in Figure 4.1.

Finding a good rescheduling heuristic is a difficult problem since it is like a modified flow shop problem with some additional constraints. A modified flow shop problem alone is NP complete. Besides the precedence and interference constraints of the modified flow-shop, match-up, ready-time and due-date constraints are also added in the defined rescheduling problem. Moreover, the heuristic also provides a feedback mechanism for enlargement decisions. In the case of failure of a feasible schedule within the given match-up point, a new point is tried at the next time which is suggested by the same heuristic too.

Accordingly, the two purposes in Step 2 can be summarized as:

1. Finding a feasible schedule within the given match-up point. If not possible, then,
2. Enlarging the pool that will lead to a feasible solution in the next try of rescheduling.

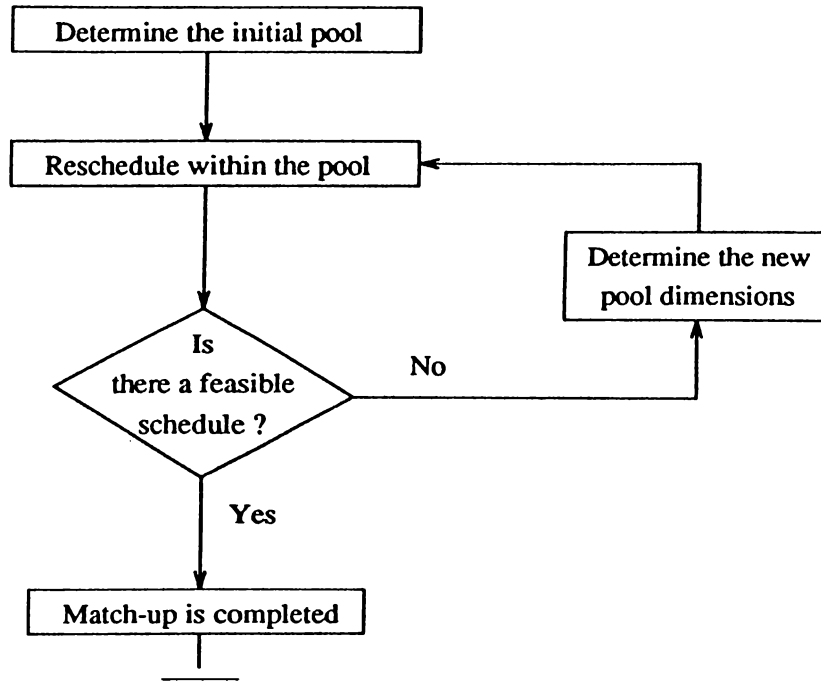


Figure 4.1. Flow-chart representation of the solution framework

These two missions are so closely related that they are covered by the same step. When an attempt of rescheduling results unsatisfactorily, the required amount of flexibility can be estimated and the new match-up point can be determined so that it will include the correct amount of flexibility. However the amount of enlargement needed to collect the required amount of opportunity depends on how flexibility is distributed on the planning horizon. As stated in the literature survey, people have either given extra flexibility to prevent effects of future disruptions or they react to these effects after the disruption by utilizing flexibility they have currently on hand. The second action is better suited to our problem, so the present allowable flexibility is an uncontrollable variable. Therefore, in deciding on the necessary amount of increase on the match-up point, the following factors are effectual:

- the amount of flexibility required to match-up,

- the length of additional time that includes the needed flexibility.

These two elements are considered both in Step 1, during determining the initial pool and in Step 2, to enlarge the pool size. The following subsection indicates how these ideas are put in a working order. Later in Section 4.1.2, the rescheduling part of Step 2 and its underlying scheme are represented.

4.1.1 Determination of a Pool

Starting from an initial pool of jobs, an initial rescheduling problem, the method checks the feasibility of match-up with current pool size and if not feasible, then guides on how to enlarge the pool, i.e. which operations to add. Choice of a right pool will give an answer to match-up time, but which pool size will result a successful match-up is unpredictable. During collecting machines and jobs in the rescheduling pool, having enough flexibility is the basic concern. But neither a direct measure for the required flexibility nor the classification of the types of flexibility can be well defined. How to measure flexibility is still a matter of question in general, but for this research word 'flexibility' is used for a more restricted meaning.

Flexibility of a given schedule includes the following two concepts:

- **machine flexibility** : amount of idleness on the machine,
- **job flexibility** : slack time of the job with given ready-time and due-date.

Machine flexibility can be, for an instance, compared with the disruption duration because the covered idleness by the pool should be at least equal to the disruption duration. Other than this condition, it is not easy to have a direct comparison between the required flexibility and the collected flexibility. The required amount depends on the other scheduling constraints as well as the duration of disruption. Moreover, a direct comparison of disruption length with the added flexibility will not be adequate since converting the job flexibility into machine flexibility, or vice versa, is not always possible. Instead of

a quantitative measure on these terms, some qualitative distinctions are used to decide on how much flexibility to bear.

To increase job flexibility, all operations of a job and consequently all machines processing it is included by the pool. That is because an out of pool operation of a job, that has other operations in the pool, will cause extra constraints for that job since its beginning time remains fixed for match-up purposes. Having all machines of the cell in the pool is a reasonable action in order to include a job completely in the pool, with its all operations. This will increase the pool size. However, cellular manufacturing is assumed to be applied with moderate cell size that will keep the machine number relatively small. Furthermore, the developed algorithms that are presented in the forthcoming sections are not very sensitive to the number of machines in terms of computation efficiency. Then having all machines automatically in the pool is decided to be an appropriate action and used from now on during determining the pools.

An important remark, before giving the exact approach, is needed to understand the relationship between the job pool and the match-up point more clearly. For a machine, taking the first n operations following the disruption in the prescheduled order, will indicate that this machine is expected to match-up the original schedule at the beginning of the $(n + 1)^{th}$ operation. During the arrangement of the pools, another point of view is that, instead of specifying operations on machines, time intervals can be considered. When different planning intervals for each machine is chosen, operations to be rescheduled appear spontaneously. This second one is a stronger argument because it checks the continuity of operations of a job in the pool. Continuity means operations in the pool belonging to the same machine should form a continuous time horizon. This means that an operation with the two other operations on both sides that are included by the pool cannot be out of the pool. In this work, pool is formed by operations instead of time intervals since it is practical to control the amount of flexibility included by this way. Weakness of this choice is compensated by building continuity checking inside the pool determination task.

Also another remark about the word 'disruption' becomes necessary such that it is used interchangeably with machine breakdown. Though breakdown

is not the only type of the disruption, many cases including breakdown can be considered as additional operation on a specified machine, with a constant and unpredictable starting time, and known duration. So, just using a specific case would not mean the lack of generality, results are applicable to similar disruption cases as well.

As mentioned in the previous section, pool size determination strategy has two stages, finding an initial pool, and updating the pool. They are described in detail in the following paragraphs. Table 4.1 summarizes the notation used throughout this chapter.

4.1.1.1 The Initial Pool

For a disruption caused by any reason, the down time covers the processing time of some jobs on the initial original schedule. This disruption duration has a beginning time, t_d , and completion time, t_u . To insert this period into the original schedule, idle times that already exist in the preschedule of down machine, k_d , are utilized. When the down time is $t_u - t_d$ units long, at least a sum of this amount of idle time is needed, assuming that rescheduling that machine with 100% utilization is possible though known that it is very rare. Finding a feasible solution for that initial pool is fine though it is not most likely. In the initial pool, flexibility just enough to cover the down duration is given, but the precedence and non-interference constraints are not considered. Because of that reason, finding a solution will not be possible most of the time. But the infeasible solution will still give feedback about the amount of enlargement that is expected to be required in the next iteration.

Initially, the job pool of the down machine, k_d , is collected and then the match-up point of this machine is determined. Job pools and match-up points of other machines are settled accordingly.

For this purpose, operations following the breakdown at the down machine, that is any job that satisfies the following condition is defined:

$$X_{i,k} \geq t_d .$$

Furthermore those jobs are sequenced in a chronological order:

n	:	number of jobs in the match-up pool
m	:	number of machines in the cell
i, j	:	job index
k	:	machine index
(j, k)	:	operation of job j on machine k
(j, k^{prec})	:	preceding operation of (j, k) for job j
(j, k^{suc})	:	succeeding operation of (j, k) for job j
k_d	:	the disrupted machine
k_j^f	:	first machine in the cell that job j is operated by
k_j^l	:	last machine in the cell that job j is operated by
r_j	:	ready time for job j on its first machine k_j^f
d_j	:	due date for job j on its last machine k_j^l
T_k^B	:	beginning time of schedule for machine k
T_k^M	:	match-up time of schedule for machine k
t_d	:	the beginning time of disruption
t_u	:	the ending time of disruption
$O_{[i]}$:	the i^{th} operation after the disruption in the initial sequence
$p_{j,k}$:	processing time of operation (j, k)
$X_{j,k}$:	planned starting time of operation (j, k) in the initial schedule
S_k	:	the set of jobs to be rescheduled on machine k
S	:	the set of jobs in the pool
T_{max}	:	maximum match-up time introduced by the planning horizon restriction

Table 4.1. Notation Summary

$$O_{[1]}, O_{[2]}, O_{[3]}, \dots$$

They are indexed on their beginning times in the initial schedule which satisfies:

$$t_d \leq X_{O_{[1]}} < X_{O_{[2]}} < \dots$$

Jobs to be rescheduled on k_d are collected in the set \mathcal{S}_{k_d} using the following procedure:

S0 Set $n = 1$.

S1 If $t_u + \sum_{i=1}^n p_{O_{[i]}} - X_{O_{[n+1]}} > 0$ then go to Step 2,
 else STOP, set $T_{k_d}^M = X_{O_{[n+1]}}$
 and $\mathcal{S}_{k_d} = \{ O_{[1]}, O_{[2]}, \dots, O_{[n]} \}$.

S2 $n = n + 1$, go to Step 1.

Step 1 checks whether enough amount of idle time is collected to cover the down duration, or not. In Step 2, adding new jobs to the set \mathcal{S}_{k_d} is continued until total idle time in the set exceeds the length of the down period.

Determination of \mathcal{S}_k for any machine other than k_d is performed as follows:

S0 Set $k = 1$.

S1 Set $T_k^M = \max_{j \in \mathcal{S}_{k_d}} X_{j,k} + p_{j,k}$

S2 For any j in the initial schedule that satisfies

$$t_d \leq X_{j,k} \leq T_k^M \quad \text{for } k \neq k_d,$$

add j to \mathcal{S}_k .

S3 If $k < m$ then $k = k + 1$,

else STOP,

$$\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2 \cup \dots \cup \mathcal{S}_n$$

$$n = |\mathcal{S}|.$$

All the jobs in the set S_{k_d} are included by other machines in Step 1. This will allow utilizing the slack of jobs during rescheduling. Determining the job pool instead of time horizon to be rescheduled has a pitfall. If a job that is sequenced in between two jobs of the pool but not included by the pool exists, this will cause a discontinuous replanning horizon. In order not to have such a case, Step 2 adds such jobs to the pool even when they are not included by S_{k_d} . Finally in Step 3, the overall job pool is formed by the union of machine pools.

4.1.1.2 Updating The Pool

In the case of failure of finding a schedule for the given pool, a larger pool is looked for which is supposed to have a feasible schedule. For a given pool, there is no way of answering the question of ‘Does it have a feasible schedule?’ without solving the rescheduling problem. The attempt of rescheduling the pool may be successful or the result may be a failure. In the case of a failure, when a feasible schedule cannot be found within the given match-up points, this result is used to decide on the next match-up point that a schedule will be looked for.

An unsatisfactory try gives feedback for future tries so makes the decision about the choice of the pool that will be considered in the next step more realistic. Modified flow shop problem’s own scheduling constraints plus the constraints about the disruption and match-up point are the potential reasons of infeasibility. To overcome these in the next try with a new pool, more flexibility is looked for. This is possible in two ways:

- i) jobs that include larger slack amounts can be included by the pool,
- ii) idleness allowed at the rescheduling period can be increased.

These two criteria are considered during enlarging the pool by adding new jobs and deciding on the new T^M . A pool that is thought to be more helpful to the rescheduling attempt is formed by this way.

Pool determination is a very important decision that will affect the overall

success of the solution methodology. A pool which is too large will unfortunately increase the dimensions of the problem and this is going to amplify the computation time. Additionally, nervousness of the system will increase since cumulative deviation is going to be much more than a smaller pools total deviation. The advantages of match-up relative to complete rescheduling will disappear under such a large pool. If the pool is too small, then a feasible solution will not exist so it will be a fruitless attempt to look for a match-up with such a pool dimension. The trade-off between two extremes should be well balanced to find the right pool size that will be most helpful for finding a feasible solution.

Updating the pool, with the insight given above, is not an independent task but is build within the rescheduling algorithm that is introduced in the next subsection.

4.1.2 Rescheduling Within The Pool

After deciding on a pool size, domain set for rescheduling is formed. This is made up of the machines and jobs, with the constraints of the problem. Constraints and objectives of the problem have been stated in Section 3.3. In this form, problem is a constraint directed scheduling problem with no direct cost objective. A breakdown event avoids processing of a machine for some period of time. Due date is considered by having an objective function of minimizing tardiness. This criterion is taken as a soft constraint while ready times of jobs are still hard. This is a necessity more than a convenience. When a job that overlaps with the disruption period is considered, the earliest time it can be operated is the end of the disruption period, t_u . However, this may not be enough if its slack becomes negative after disruption because its slack value at the initial schedule has not been enough to meet such a disruption. In such a case, there is not many possibilities to do except letting it to be tardy. So having due-dates as hard constraints might cause an infeasible problem that will lead to a unsuccessful attempt for any match-up point.

A secondary reason for having due-dates soft is that due-dates are the constraints caused by later events when compared with ready times. Relaxing the due-dates will give more time for updating the plans affected by the reschedule

than relaxing the ready times.

The interrupted machine is going to be in a non-working condition for some time though it has previously scheduled jobs, to be processed on that period. Then it becomes bottleneck because of delayed jobs and a high utilization is required to cover this disruption duration. An approach similar to the Shifting Bottleneck Procedure developed by [Adams et al.87] is applied and priority is given to the down machine during rescheduling. This attitude separates the problem and the pool into three distinct parts which are the bottleneck machine, machines in the upward direction of it and ones in the downward direction, according to the order of machines in the flow-shop.

Initial ready times, r_j , are the ready times for the first operation of job j on the cell and due dates, d_j , are for the last operation. A job enters the cell on machine k_j^f and leaves the cell on machine k_j^l . Another definition for machine specific ready-times and due-dates are called as $ES_{j,k}$, the earliest start time of job j on machine k , and $LF_{j,k}$, the latest finish time of job i on machine k . These constraints are applied on the disrupted machine by computing earliest start and latest finish times of jobs on that machine with respect to these ready times and due dates.

The following algorithm describes how we can decompose the problem. Rescheduling the down machine is defined as an $(n/1/r_i/\sum T_i)$ problem and a consistency check between the results of these three partitions are applied.

S1 Update r_j and d_j for $\forall j \in \mathcal{S}$, according to $T_{k_j^f}^B$ and $T_{k_j^l}^M$.

That is,

$$\begin{aligned} r_i' &= \max(r_i, T_{k_j^f}^B), \\ d_i' &= \min(d_i, T_{k_j^l}^M). \end{aligned}$$

S2 Make forward and downward scheduling and calculate

$ES_{j,k}, LF_{j,k}$ for $\forall j \in \mathcal{S}$ and $k = 1, \dots, m$.

Let (j, k^{prec}) be the last machine at which job j is processed by before beginning its operation at machine k . In other words, (j, k^{prec}) is the immediate predecessor of operation (j, k) .

Similarly, call (j, k^{suc}) as the immediate successor of operation (j, k) for job j , that it will be processed by machine k^{suc} after finishing its operation on machine k .

S2.1 $ES_{j,k^f} = r'_j$ for $\forall j \in \mathcal{S}$.

S2.2 Calculate

$$ES_{j,k} = \begin{cases} \max(ES_{j,k^{prec}} + p_{j,k^{prec}}, T_k^B) & \text{if } j \in \mathcal{S}_{k^{prec}} \\ \max(X_{j,k^{prec}} + p_{j,k^{prec}}, T_k^B) & \text{if } j \notin \mathcal{S}_{k^{prec}} \end{cases}$$

for $k \neq k_j^f$ and $\forall j \in \mathcal{S}_k$.

S2.3 $LF_{j,k_j^l} = d'_j$ for $\forall j \in \mathcal{S}$.

S2.4 Calculate

$$LF_{j,k} = \begin{cases} \min(LF_{j,k^{suc}} - p_{j,k^{suc}}, T_k^M) & \text{if } j \in \mathcal{S}_{k^{suc}} \\ \min(X_{j,k^{suc}}, T_k^M) & \text{if } j \notin \mathcal{S}_{k^{suc}} \end{cases}$$

for $k \neq k_j^l$ and $\forall j \in \mathcal{S}_k$.

S3 k_d is the bottleneck machine, schedule it first.

With

$$ES_{j,k_d}, LF_{j,k_d} \text{ for } \forall j \in \mathcal{S}_{k_d} \text{ and } |\mathcal{S}_{k_d}| = n_{k_d},$$

this is a $(n_{k_d}/1/r_i / \sum T_i)$ problem which is NP-Hard.

Solve this problem using the algorithm proposed in Subsection 4.2.1.

S4 Update ES and LF bounds for operations on the upward and downward machines ,

$$\begin{aligned} LF'_{j,k_d^{prec}} &= X_{j,k_d} & \text{for } \forall j \in \mathcal{S}_{k_d^{prec}} \\ LF'_{j,k} &= LF'_{j,k^{suc}} - p_{j,k^{suc}} & \text{for } k = 1, \dots, k_d - 1 \text{ and } \forall j \in \mathcal{S}_k \\ ES'_{j,k_d} &= X_{j,k_d} + p_{j,k_d} & \text{for } \forall j \in \mathcal{S}_{k_d+1} \\ ES'_{j,k} &= ES'_{j,k^{prec}} + p_{j,k^{prec}} & \text{for } k = k_d + 1, \dots, m \text{ and } \forall j \in \mathcal{S}_k \end{aligned}$$

S5 Upward machines' problem is a modified flow-shop problem

with hard r_i and d_i constraints.

Solve this problem using the algorithm proposed in Subsection 4.2.2.

that assumes all operations of jobs are processed with no delay, for all jobs, earliest starting times on each machine, $ES_{j,k}$, is calculated. In a similar fashion, by a backward schedule, their latest finishing times, $LF_{j,k}$'s are determined in Step 2.

Steps 3, 5 and 6 are solved independently by developing their own solution methodologies since their problem statements are dissimilar. Constraints and objectives of each one is illustrated in Figure 4.2.

Algorithms used to schedule each of the decomposed portion are introduced in Section 4.2. Initially the disrupted machine is rescheduled with a heuristic that is discussed in Subsection 4.2.1. Then, by the resulting new beginning times of jobs on this machine, current $ES_{j,k}$ and $LF_{j,k}$ values are updated in Step 4 of the above algorithm to be used by the upward and downward machines. These two groups of jobs and machines have different problem characteristics, so they are scheduled with two different heuristics that are discussed in Subsections 4.2.2 and 4.2.3 respectively.

4.2 Rescheduling Algorithm

As represented in Figure 4.2, the rescheduling problem, when a match-up point is given, is decomposed into three subproblems. The scheduling attempt begins with the most critical resource which is the broken-down machine. The heuristic algorithm developed for this part of the problem is represented in Section 4.2.1. After scheduling this machine, due-dates of the jobs in the upward machine pool are updated and the algorithm discussed in Section 4.2.2 is applied. For the last part of the problem, the heuristic algorithm developed in Section 4.2.3 is used. Table 4.2 summarizes the additional notation used in this section.

4.2.1 Disrupted Machine's Schedule

Scheduling n jobs having different due dates but a unique ready time with an objective of minimizing the total tardiness problem, ($n/1/r_i = 0/\sum T_i$), has recently been proved to be NP-hard [Du&Leung 90] which implies that

$i \prec j$: job i precedes job j in the sequence
s_j	: slack time for job j
$UNSCH$: set of unscheduled jobs
ΔT	: net idleness on the breakdown machine
R_t	: set of jobs ready in the cell at time t
UB	: upper bound on tardiness on the breakdown machine

Table 4.2. Notation Summary For The Rescheduling Heuristic

the current $(n/1/r_i/\sum T_i)$ problem is also NP-hard. Unequal ready times case is harder to deal with since the alternative of inserting idleness is needed to be evaluated. It can be shown that inserted idleness never improves a static scheduling problem with equal ready times, however, it might work well for a dynamic problem with varying ready times. This dynamic pattern is handled with a branch and bound algorithm that uses a static rule and idleness insertion together.

A heuristic method is formed by developing a dominance rule to be used in the pairwise comparison of the jobs first and then by applying this static solution to the dynamic case considering ready times and inserted idleness. This proposed dominance rule and its underlying idea are mentioned in the following paragraphs.

4.2.1.1 A Dominance Rule To Minimize Tardiness

At any time t when the machine is ready, a criterion is needed to choose one job among the n_t elements of the job set R_t , which includes jobs that are ready to be processed,

$$R_t = \{j : r_j \leq t\}.$$

The proposed criterion is a combination of processing time, p_i , and remaining slack time, $t - d_i - p_i$. It differs from the common dispatching rules in a way that it is a function of t instead of a constant value which would be same at all the time slots that a decision is looked for.

A function, $f(j, t)$, that is the ratio of slack time at t to the processing time, is defined and properties of it are used for concluding the following results.

The function, $f(j, t)$ is defined as follows:

$$f(j, t) = \begin{cases} \frac{d_j - r_j - p_j}{p_j} & \text{if } t \leq r_j \\ \frac{d_j - t - p_j}{p_j} & \text{if } t > r_j \end{cases}$$

Slack time is the remaining time for a job to be processed with zero tardiness. That is if slack time is negative then this job is already tardy. The latest time point that a job can start processing with zero delay is $d_j - p_j$ which is the intersection of $f(j, t)$ with x-axis. The vertical distance of a job at a given t to its intersection point with x-axis indicates the urgency of the job. Another important criterion that affects the choice of jobs is the slope of the $f(j, t)$ function, $-1/p_j$. In Figure 4.3, these two concepts are illustrated geometrically using the $f(j, t)$ function.

As the slope increases, the speed of reaching to the point of intersection with the x-axis or departing from that point increases. The job with the highest slope or at present closest to its own intersection point is more urgent than other jobs that are also ready at this time point. This property is used to develop a dominance rule for choosing one of the two jobs that are ready at time t in order to minimize tardiness. The following theorem is written by using the geometric interpretation of the $f(j, t)$ function. A detailed proof is given in Appendix A-1.

Theorem 4.1. *Given two jobs i and j with $p_i \leq p_j$, both ready at time t^c , with the apsis of the intersection of $f(i, t)$ and $f(j, t)$ called t^* where,*

$$t^* = \frac{p_j d_i - p_i d_j}{p_j - p_i},$$

under the specified conditions stated below, given rules minimize the tardiness.

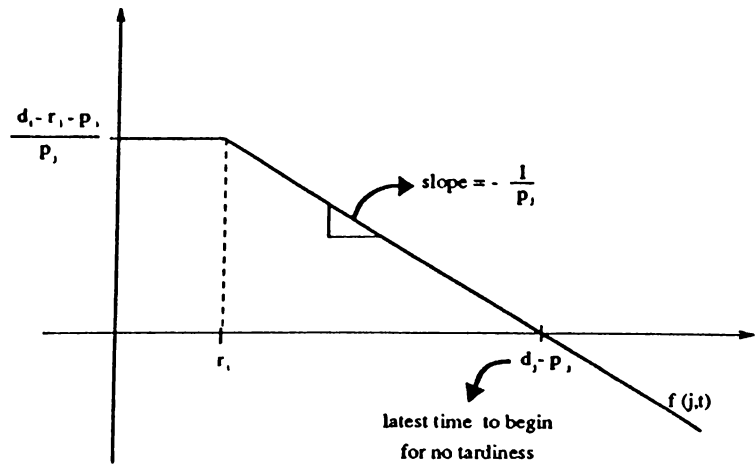


Figure 4.3. Interpretation of the $f(j, t)$ function

i) for $p_i \neq p_j, t^* > t$,

if $f(i, t^c) \geq 0$ and $f(j, t^c) \geq 0$, then use **EDD**,
 if $f(i, t^c) \geq 0$ and $f(j, t^c) < 0$, then if $d_i - p_j \geq t^c$ $j \prec i$,
 else $i \prec j$,
 if $f(i, t^c) < 0$ and $f(j, t^c) < 0$, then use **SPT**,

ii) for $p_i \neq p_j, t^* \leq t^c$, use **SPT**,

iii) for $p_i = p_j$, use **EDD**.

Additionally, it is proved in Appendix A-2 that this dominance rule is transitive such that if job i precedes job j and job j precedes job h , then job i precedes job h as well. Then job i can be called the best choice among the three alternatives to minimize tardiness.

Theorem 4.2. The sequencing operator ' \prec ' defined in Theorem 4.1. has transitivity property.

The dominance rule given above helps choosing one job among a number

The dominance rule given above helps choosing one job among a number of them by using the transitivity property of the rule. The static problem of $(n/1/r_i = 0/\sum T_i)$ can be solved by these two results.

This rule, alone, is applicable to the static system where all jobs are ready at the beginning of the schedule. In the dynamic case, some alternatives may not be ready at the decision point. So there is trade-off between waiting idle for a critical job to become ready or choosing one of the currently ready jobs. In such a case, tardiness that is found by scheduling with the above dominance rule without inserting any idleness can be used as an upper bound and other alternative schedules having inserted idleness can be evaluated by a branch and bound technique. A branch and bound approach developed for this purpose is introduced in the next paragraphs.

4.2.1.2 A Branch and Bound Approach On $(n/1/r_i/\sum T_i)$

In the dynamic case, at point t , only some of the jobs are ready and there are two alternative decisions, either choosing one among the ready ones and start processing immediately or considering jobs that will be ready soon and waiting idle for one of them to become ready when this job seemed critical. Introducing the second alternative increases the computational complexity. A branch and bound application is used in this research to schedule one machine, n jobs problem with ready times and due dates to minimize total tardiness.

The levels in the tree indicates the sequence. A number of candidates exist for a given level which are the branches.

If this branching procedure was carried out completely, there would be $n!$ ending leaves on the tree, each representing a distinct feasible solution. This will be a complete enumeration of all sequences. The function of the bounding process is to curtail this enumeration. For this purpose, initially, ignoring the inserted idleness issue, dominance rule is applied without violating the ready times. The resulting total tardiness value is used as an upper bound during branching.

During branch and bound, LIFO rule is used to determine which subproblems should be solved next. Initially, by using the dominance rule, a schedule

with no inserted idleness is determined and its total tardiness amount is used as an upper bound to evaluate the future candidates. Search begins with the last level. If a candidate can be found for the present node, then we branch on this node and determine the sequence of jobs succeeding it by the dominance rule till upper bound is exceeded or all jobs are scheduled. When a solution with lower tardiness value is found, upper bound is updated and search for a better one continues beginning from the node that has been branched last time. Searching attempt ends after the alternatives of the first level are checked.

The following branch and bound algorithm considers inserted idleness and tries to minimize total tardiness :

- S0 Create an initial candidate by sequencing jobs with the **Dominance Rule**.
Calculate the total tardiness and let UB be equal to the total tardiness.
- S1 Go to the last created branch at level $n-1$.
Find an alternative for this branch by inserting idleness.
Create a branch for this alternative.
Repeat this try until no more alternatives exist.
Set $l = n - 1$.
- S2 Go to the last branched node at level l .
Try to find an alternative for this branch by inserting idleness.
If an alternative exists, then go to Step 3.
Else, if $l \neq 1$, set $l = l - 1$, go to Step 2,
else STOP, branch and bound completed.
- S3 Sequence the jobs that are not yet assigned up to the current branch with the **Dominance Rule** and create the corresponding branches till current tardiness exceeds UB or all jobs are scheduled.
If at any level UB is exceeded, then this branch is fathomed, go to Step 2.
Else if the complete schedule's total tardiness is less than UB , update the value of UB with this total tardiness value and go to Step 1.

Finding alternatives at a certain level for a given branch is performed by checking the ready times of jobs. An alternative should not be ready at or before the beginning time of the job on the given branch. Moreover, because of the well-defined match-up pool, the total idleness, ΔI , that the machine is allowed is known,

$$\text{total idleness}(\Delta I) = \text{idleness in the pool} - \text{disruption duration.}$$

Whenever some idleness is inserted to the schedule, the value of ΔI is updated by reducing this amount from the previous value. A schedule cannot use more than this amount of idleness, otherwise match-up point is exceeded. Therefore this total idleness limit introduces an upper bound for the ready time of the alternative. Another point is that if the job on the given branch can be completed by the time that the chosen alternative becomes ready, examining this alternative will not improve the current solution. When inserted idleness is greater than the processing time of the currently ready job, then that job can be processed within this period without affecting the beginning time of the alternative job.

When job i is at the last branch which an alternative is looked for, and t is its beginning time, the set of alternative jobs can be defined as:

$$ALTJOBS = \{j \mid t < r_j \leq t + \min(p_i, \Delta I)\}$$

This kind of a limitation on the alternatives reduces the number of branches, so better curtail the enumeration in the tree.

4.2.2 Upward Machines' Schedule

After a feasible solution is found for the disrupted machine, its jobs' beginning times become input for the upward machines. They define the possible latest ending times of operations on these machines which are calculated as described in the algorithm given in Subsection 4.1.2.

This is a constraint directed problem with no objectives but strict bounds. A feasible solution may not always exist and when this is the case, this heuristic lets the violation of due dates caused by the schedule of the disrupted machine.

The resulting solution will be infeasible but this output will provide a feedback mechanism on deciding how much to enlarge the rescheduling pool.

The proposed heuristic for this part of the decomposition is stated below:

S0 Calculate slack values, s_i , of each job as:

$$s_i = d_i - r_i - p_i.$$

Jobs with $s_i = 0$ are called critical.

If there exists any critical job, schedule it through all machines.

Set $k = 1$.

S1 Update $ES_{i,k}$ and $LF_{i,k}$ based on the scheduled jobs

for $l = 1, \dots, k_d - 1$.

$$t = \min_{j \in UNSCH} ES_{j,k}.$$

S2 S2.1 $R_t = \{i \mid t \geq ES_{i,k}, i \in UNSCH\}$.

S2.2 For $\forall j \in R_t$, have a one step look ahead:

- i) if it is scheduled at time t , do any of the remaining jobs could become infeasible,
- ii) if it is scheduled at time t , does it overlap with any of the previously scheduled jobs.

S2.3 If the number of feasible alternatives is more than one, choose the most critical job, with minimum s_i value, to be scheduled at time t , call j^* .

S2.4 If the number of feasible alternatives is none, choose the one causing minimum tardiness on other unscheduled jobs.

S3 Update $ES_{j,k}$ and $LF_{j,k}$ based on the scheduled job.

If any job becomes critical when all other jobs have positive slack, schedule it through the all upward machines.

If there are unscheduled jobs on the current machine k , then

set $t = t + p_{j^*,k}$,

go to Step 1.

Else if machine k 's schedule is completed,

set $k = k + 1$,

go to Step 1 till all upward machines are scheduled,

go to Step 5 when scheduling is completed.

S4 Check the feasibility of the schedule.

If at least one job is scheduled after its due-date, then schedule is infeasible.

S5 If the schedule is not feasible, then

apply the Pool Enlargement Procedure which is described in Section 4.3.

Scheduling begins with the most restricted jobs. These jobs have no slack and should be scheduled at their ready times since tardiness is not allowed. They are the jobs that are scheduled on their earliest start times on the down machine so that no slack has been left for the operations on the upward machines. Step 0 schedules them to be processed on time, and in Step 1, unscheduled jobs' earliest start and latest finish times are updated accordingly.

In this heuristic, since ready-times and due-dates are hard bounds, feasibility is the basic concern. For this purpose, one step look ahead feasibility checks are performed to find candidate jobs for a certain interval by Step 2.

At the beginning of the procedure, job oriented scheduling is performed for the critical jobs in Step 0, though for the remaining jobs, machine oriented schedule is applied at Step 3. After scheduling a job, earliest start times and latest finish times of other jobs' operations processed by the same machine are updated. Similarly, when a machine is completely scheduled, other operations of the jobs' are updated as well.

After all jobs on all machines are scheduled, Step 4 checks if its results are compatible with the beginning times of jobs in the down machines' schedule. In the case of infeasibility, the Pool Enlargement Procedure referred in Step 5 determines the necessary amount of enlargement for the next iteration.

In the absence of a feasible solution, scheduling attempt continues to estimate the amount of extra idleness to be added in the next iteration. In this case, Step 2 tries to minimize the amount of lateness. For the downward part of the problem, a similar approach is used. Scheduling attempt continues even after it becomes obvious that the solution will result in infeasibility. The heuristic algorithm developed for this downward part of the problem is presented in the following section.

4.2.3 Downward Machines' Schedule

The downward machines' scheduling problem can be defined as a multi machine problem where the jobs have specific ready times and due-dates. There are two kinds of due-date constraints:

1. the completion time of last operations of the jobs' in the initial schedule,
2. match-up times of each machine.

As discussed in the previous sections, the first type of due-dates are carried as soft constraints and the second type as hard constraints. In other words, deviation from the completion times at the initial schedule, tardiness, is allowed if match-up points are not violated. There are two objectives related with this claim:

1. Finding a feasible schedule within the match-up points,
2. Minimizing the tardiness during rescheduling.

The distinction on the due-dates makes this problem different than a classical scheduling problem.

An operation oriented scheduling scheme is followed in the proposed heuristic algorithm which is a hybrid approach of machine oriented and job oriented applications. The underlying idea of the algorithm is that, a cycle which begins from the last machine to the first machine of the downward stream, is followed till all jobs are scheduled. For the current machine, candidate operations are chosen and if they specify the stated conditions of the algorithm, they are scheduled. Otherwise, no job is scheduled in that trial and the same attempt is repeated for the next machine. After searching the first machine for candidate jobs, it turns back to the last machine in order to initiate to a new cycle.

A backward scheduling method is performed by the proposed algorithm. Due-dates of the jobs are treated as if they are ready-times and the ready-times as the due-dates in that backward fashion. Scheduling begins from

the match-up point and is directed to the beginning point so that match-up time constraints are strictly followed. However, this might cause a violation of ready-times. Such a violation indicates that the developed downward machines' scheduling is not feasible. When it is seen that no feasible solution exists, scheduling continues which means match-up with these dimensions is impossible but still a schedule, though infeasible, is looked for to estimate the necessary enlargement to determine the next pool size.

The proposed heuristic for downward machines is stated below:

- S0 Set $t_k = T_k^M$, for $k = k_d + 1, \dots, m$.
- S1 Choose a job j with the Latest Due Date rule, **LDD**, on the last machine, m , and schedule that job so that it will be completed at t_m .
Call the chosen job, j_m^* .
- S2 Set $t_m = t_m - p_{j_m^*, m}$.
and $k = m - 1$.
- S3 S3.1 If $k > k_d$, then go to Step 3.2,
else, go to Step 1.
- S3.2 Choose the job with the **Dominance Rule** on machine k .
Call the chosen job, j_k^* .
- S3.3 If $d_{j_k^*, k} \geq t_k$ or k is the last machine of j_k^* , then,
if $t_k - p_{j_k^*, k} \geq t_{k+1}$, then,
schedule it to be completed at t_k ,
set $t_k = t_k + p_{j_k^*, k}$,
go to Step 3.2,
else,
set $k = k + 1$,
go to Step 3.1,
else
if $d_{j_k^*, k} - p_{j_k^*, k} \geq t_{k+1}$, then,
schedule it to be completed at $d_{j_k^*, k}$,
set $t_k = d_{j_k^*, k} - p_{j_k^*, k}$,
go to Step 3.2,
else,

set $k = k + 1$,
go to Step 3.1.

S3.4 Set $t_k = d_{j_k^*,k} - p_{j_k^*,k}$.

S3.5 Recalculate $ES_{j,l}$ and $LF_{j,l}$ based on the scheduled job
for $l = k_d + 1, \dots, m$.

S4 Go to Step 2 until all jobs are scheduled.
Otherwise go to Step 5.

S5 Check the feasibility of the schedule,
If at least one job is scheduled before its ready time,
then schedule is infeasible.

S6 If the schedule is feasible, then
apply left shift procedure to minimize the tardiness,
else if it is not feasible,
apply the Pool Enlargement Procedure.

Scheduling begins with the last machine of the cell. The initial sequence of this machine remains unchanged after Step 1. This is because the completion times in the initial schedule are taken as due-dates during rescheduling and when LDD rule is applied, the sequence would be same. However the completion times might be different from the initial schedule. The idea during scheduling the last machine is that, tardiness is tried to be minimized by keeping the sequence unchanged but possible right shifts are allowed to increase the slackness of jobs.

After a job is scheduled on the last machines, other machines, from $m - 1$ to $k_d + 1$, are checked to schedule jobs by step 2. There are some rules about this scheduling attempt:

- On a machine, a candidate job is looked for by the Dominance Rule, stated in Subsection 4.2.1. Ready times are taken as due-dates and due-dates as ready times in this backward scheme.
- The candidate job cannot be scheduled if its beginning time is smaller than the the beginning time of the last job scheduled at the succeeding machine. Such a restriction avoids giving a decision for a time interval

on some machine before the decisions about the same time interval for the succeeding machines are given.

- Latest finish time of job i on machine k , $LF_{i,k}$, represents the due-date if k is the last machine of the job or it stands for the the beginning time of the succeeding operation of the same job which is already scheduled. For the first case, $LF_{i,k}$ is a soft constraint which means that a job can be tardy. However, in the second case it is a hard constraint since $LF_{i,k}$ represents the precedence relations.

The schedule determined at the end of Step 4 might be incompatible with the schedules previously determined for the down machine and upward machines. Operations of the same job in different machine groups (e.g. operations on upward machines and the down machine) might overlap. If it is found to be infeasible by Step 5, in Step 6 the Pool Enlargement Procedure is applied to decide on the size of the pool in the next iteration. The details of this procedure are represented in the next section.

4.3 Enlarging The Pool Size

The application of the heuristic that is described in detail throughout Section 4.2, might not result in a feasible schedule. When this is the case, a new pool is formed to be rescheduled. Determination of the pool is quite important since a good choice may decrease the number of trials while a bad one may result with no solution very frequently. The basic idea behind insisting to complete schedules even after it becomes clear that it will not be feasible, is giving a feedback about the additional flexibility required by the problem. Then the problem pool is enlarged in order to cover this additional flexibility within the new problem limits.

In this section, the enlargement procedure is discussed under two main topics, initially the general underlying idea in enlargement is covered in Subsection 4.3.1, and subsequently the procedure developed for enlargement in order to include this extra amount is presented in Subsection 4.3.2.

4.3.1 Reasoning Behind Enlargement

As illustrated in Figure 4.2 in Subsection 4.1.2, machines in the cell are grouped in three parts:

1. the brokendown machine,
2. upward machines,
3. downward machines.

During developing the algorithms for the decomposed parts, it has been seen that the defined problems for each of the three groups are strictly bounded. When a feasible solution does not exist, some of the constraints are allowed to be relaxed. However, any arbitrary infeasible solution cannot always indicate the amount of enlargement required to get a feasible solution. This leads us to consider some of the hard constraints as soft ones which means that violating these constraints are still undesirable though not strictly restricted.

In all of the three machine groups, the purpose of the individual algorithms is to keep the schedule in between the beginning, T^B , and match-up, T^M , and to follow the precedence relations on the machines of the same group. But none of them guarantees that their result fits with the results of the remaining two groups. The purpose of such a relaxation is to get feedback about the amount of infeasibility for the next iteration when a feasible solution does not exist.

In the hierarchical scheme, initially the brokendown machine is scheduled. New starting times of jobs on this machine are used as due dates for the operations on upward machines during scheduling the upward machines. For the ideal case, these due dates should be hard constraints in order to have completeness between the machine groups. Instead they are considered as soft constraints, otherwise the problem was too constrained. But match-up points are still hard, no operation's completion time can exceed the machine's match-up point. For the downward machines, the completion times on the down machine are used as ready times by the operations on the downward machines. As the jobs are scheduled from future to present fashion in the downward pool, they might be scheduled before their ready times, though such a behavior is penalized in the algorithm. Both of the occurrences cause a single type of

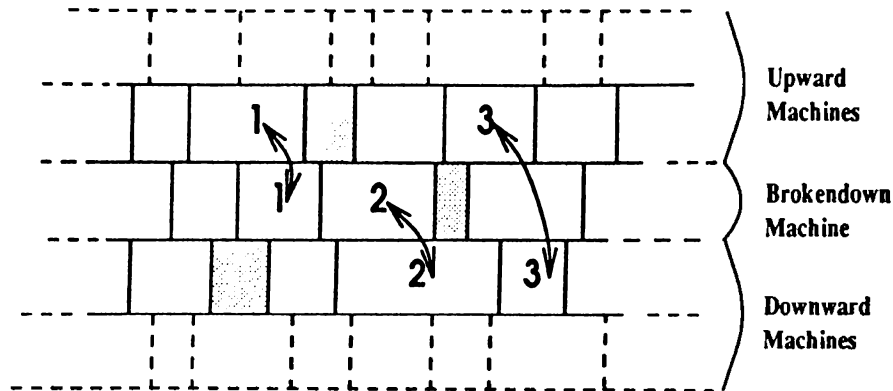


Figure 4.4. Examples for infeasibilities between the machine groups

infeasibility; the completion time of a job's operation might be greater than the beginning time of its succeeding operation only when they are performed in two different machine groups. In Figure 4.4, an illustrative example is given on the Gantt Chart.

As illustrated by the figure, there are three possible pairs that can have infeasibilities between each other:

1. upward machines' group and the brokdown machine,
2. the brokdown machine and the downward machines' group,
3. upward machines' group and the downward machines' group.

When the rescheduling attempt in the current pool is not successful, then the reasons should be questioned. The most important reason is the insufficient flexibility in the pool. This should be supplied by enlarging the pool in order to gain the following advantages:

1. increasing total idleness in the pool,
2. increasing the freedom of jobs in terms of number of possible sequences and the time interval that a job can be scheduled.

I	: job set of upward machines
II	: job set of the brokendown machine
III	: job set of downward machines
$C_{j,}$: completion time of job j on machine group (.)
$B_{j,}$: beginning time of job j on machine group (.)
$e.(k)$: amount of k^{th} infeasibility on pair (.)
$e_max_{brokendown}$: amount of enlargement on the brokendown mac.
$e_max_{downward}$: amount of enlargement on the downward macs.

Table 4.3. Notation Summary For The Pool Enlargement Algorithm

The trade-off is that an enlargement also brings some disadvantages like:

1. the number of jobs will increase which means the problem size expands, so complexity increases,
2. the match-up point moves far from the current time and this increases the nervousness.

It can be concluded with this trade-off that the enlargement in the schedule should give the required amount of flexibility by adding a minimum number of jobs to the pool. The algorithm that is developed for enlargement tries to offer a pool to consider these purposes.

For all pairs, amounts of infeasibilities are checked job by job. The maximum amount of infeasibility is accepted as the required enlargement quantity for each pair. This is the amount of idleness that is needed to be added to the pool to get a feasible schedule for the corresponding pair.

Initially, enlargement is performed by the machine group that is in the downstream according to the other group in the pair. New jobs are added to

its job pool till amount of idleness added is equal to the demanded quantity. Consequently, these newly contained jobs are also added to the other group in the pair. When there is no infeasibility between groups, then no direct enlargement is performed, but because of other pairs, some jobs are added to their pool to have a complete pool. This is similar to initial pool creation method described in Subsection 4.1.1.

In the following subsection, a detailed algorithm with an underlying idea as discussed in the previous paragraphs is given.

4.3.2 The Pool Enlargement Algorithm

After it is observed that, with the current job pool heuristic does not result a feasible schedule, a new pool is needed for another scheduling attempt. This new pool should be large enough to include required flexibility but small enough to find a sooner match-up point. Table 4.3 summarizes the additional notation used in this section.

The proposed algorithm for enlargement is stated below:

S1 Check the feasibility of the current schedule:

S1.1 Check feasibility between upward machines
and the brokendown machine:

Set $c = 0$.

For $\forall j \in \mathcal{I} \cap \mathcal{II}$,

if $C_{j,II} > B_{j,I}$, then,

job j causes infeasibility,

it is added to the set \mathcal{F}_1^c ,

set $c = c + 1$,

calculate $e_1(c) = C_{j,II} - B_{j,I}$.

S1.2 Check feasibility between downward machines
and the brokendown machine in a similar way.
Calculate $e_2(c)$ values and determine the set \mathcal{F}_2^c .

S1.3 Check feasibility between upward machines
and downward machines in a similar way.

Calculate $e_3(c)$ values and determine the set \mathcal{F}_3^c .

S2 If $e_1(k)$, $e_2(k)$, $e_3(k)$ are equal to 0, then,
 STOP, schedule is feasible, else,
 enlargement is needed, go to S3.

S3 Find $\max \{ e_1(k) \}$, call $e_max_{broktdown}$.
 Find $\max \{ e_2(k), e_3(k) \}$, call $e_max_{downward}$.

S4 If $e_max_{broktdown} > 0$ and $e_max_{downward} = 0$, then,

S4.1 enlarge the broktdown machine's pool to add
 $e_max_{broktdown}$ amount of idleness,

S4.2 enlarge the pools of the other machines to
 include the new jobs added to the broktdown
 machine's pool, STOP.

Else if $e_max_{broktdown} = 0$ and $e_max_{downward} > 0$, then,

S4.1 enlarge the pool of the every downward machine
 to add $e_max_{downward}$ amount of idleness,

S4.2 enlarge each of the downward machines' job pool
 to make sure that any job in one of these machines'
 is also included by other downward machines' pools,

S4.3 enlarge the pools of upward machines and the
 broktdown machine to include the new jobs added
 to the downward machines' pools, STOP.

Else if $e_max_{broktdown} > 0$ and $e_max_{downward} > 0$, then,

S4.1 enlarge the pool of the every downward machine
 to add $e_max_{downward}$ amount of idleness,

S4.2 enlarge each of the downward machines' job pool
 to make sure that any job in one of these machines'
 is also included by other downward machines' pools,

S4.3 enlarge the pools of upward machines and the
 broktdown machine to include the new jobs added
 to the downward machines' pools,

- S4.4 if this indirect enlargement of the brokendown machine allows at least $e_max_{brokendown}$ amount of idleness then STOP,
 else enlarge the brokendown machine's pool to complete $e_max_{brokendown}$ amount of idleness,
- S4.5 enlarge the pools of the other machines to include the new jobs added to the brokendown machine's pool, STOP.

This algorithm terminates with a new set of jobs on machines to be rescheduled if the enlargement is possible, that is the planning horizon is long enough to collect required amount of idleness. When the time is not enough for match-up, this algorithm cannot perform enlargement so match-up heuristic ends with unsatisfaction.

The enlargement procedure has two main tasks, computation of the required idleness quantity and determination of new job pool for each machine to cover this required amount. Step 1 checks the feasibility of the schedule and if not feasible, measures the infeasibility by calculating the length of overlap, if there exists any. The maximum length of overlap is set as the enlargement amount by Step 3 after enlargement is initiated by Step 2.

As described in Subsection 4.3.1, enlargement is directly performed by the machine group that is in the downstream according to the other group in the pair. Other group is enlarged indirectly to cover the jobs added to the downstream during direct enlargement. This indicates that either the brokendown machine or the downward machines are enlarged directly, but never the upward machines.

The enlargement amount for these two machine groups is determined by Step 3 by using the enlargement amounts calculated in Step 1. All three combinations of enlargement values are considered individually in Step 4 and pools of each machine are determined after necessary enlargements. For the case where both the brokendown machine and the downward machines should be enlarged simultaneously, initially the downward machines' pools are enlarged and other machines pools are updated accordingly. After that, the procedure checks if the brokendown machine has included its idleness requirement with

this indirect enlargement. When it still requires additional amount of idleness, an other enlargement is performed.

In the proposed heuristic, a number of match-up points are tried until one of them results a feasible schedule. It is desired to reach the feasible match-up point with minimum number of trials. The stability of the algorithm is effected by the number of iterations and the length of match-up period. Because of this momentous effect of creating the right match-up points, feedback mechanism is especially preferred in the algorithm.

4.4 An Example

In this section, the general idea of the developed heuristic is represented on an illustrative example. Only the first iteration of the heuristic is described, where the steps are represented on Gantt Charts.

The initial schedule is given in Figure 4.5. It is disrupted by a breakdown on the fifth machine. Duration of breakdown is 13 units of time and covers operations of jobs 7, 9 and 12 on the fifth machine.

Initial pool is determined by the procedure detailed in Section 4.1.1. Idle time just enough to cover the down duration is collected by the disrupted machine and the job pools of other machines are formed to include the pool of the brokendown machine. In Figure 4.6, the boundaries of the overall pool and disruption duration is represented on the Gantt Chart.

In the defined pool, Rescheduling Algorithm described in Section 4.2 is applied. Machines are decomposed into three groups and for each group the proposed algorithms are applied. New Gantt Chart after rescheduling is illustrated in Figure 4.7.

Since precedence relations between machine groups are relaxed during decomposition, new schedule does not guarantee the feasibility. As seen from the Gantt Chart, this schedule is not feasible. This indicates that the heuristic is not able to find a feasible schedule for the given match up point. Then the pool enlargement procedure is used to determine the new match up point to be used

in the next iteration. For this specific example, the maximum amount of infeasibility is caused by job 16, which gives the required amount of enlargement for the next iteration.

4.5 Summary

After introducing the problem in Chapter 3, this chapter displays the proposed hierarchical scheduling approach as a solution tool to the problem. An overall description of the solution approach is discussed in Section 4.1. It has two basic steps which are stated as follows:

1. Finding a feasible schedule within the given match-up time.
2. Finding a match-up time that a feasible schedule exists.

To begin with, an initial pool is determined with the procedure described in Subsection 4.1.1, that includes the minimum amount of flexibility that is expected to solve the problem. Afterwards, the rescheduling algorithm in Section 4.2 is applied on the defined pool. This algorithm decomposes the problem into three subproblems, scheduling the brokendown machine, the upward machines and the downward machines. For each of the three subproblems, a different methodology is developed and they are solved in the given sequence.

After the defined pool is rescheduled with the proposed algorithm, its feasibility is checked. If the resulting schedule is infeasible, then the job pool is updated. This is done by adding new jobs to the pool which means by shifting the match-up point right in the planning horizon. This enlargement is performed by taking feedback from the infeasibilities met in the previous pool. The details of this task is described in Section 4.3.

In the forthcoming chapter, the experimental design that is build to test the performance of this proposed heuristic is going to be presented.

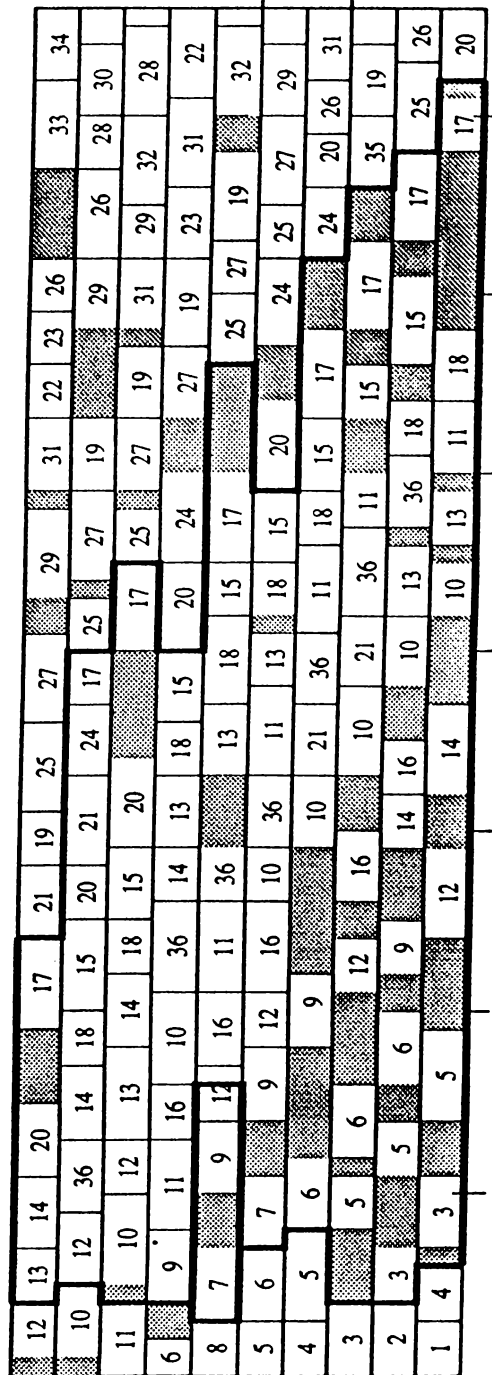


Figure 4.6. Breakdown on the 5th Machine and Initial Pool Boundaries

Chapter 5

Computational Analysis

A reactive strategy that reschedules part of the initial schedule to accommodate the disruption has been proposed in Chapter 4. In order to test the effectiveness of this algorithm, a group of experiments are designed to be performed in this chapter.

There are two major questions that an answer is looked for:

1. How does our heuristic perform when compared with other alternatives serving for the same purpose?
2. Which variables are most influential on the response of it?

The first question points the evaluation and comparison of the algorithm. The latter one includes the determination of key design parameters that impact the performance of the algorithm.

Initially, an experimental design which includes a series of tests is built. In the experiments, purposeful changes are made on the input variables of the heuristic in order to observe and identify the reasons for changes in the response of the heuristic. According to the intuition that a variable is influential or not, it is classified as a fixed parameter or as a factor in Section 5.1. Additionally, the response of the system is defined in terms of some chosen performance measures in this section.

Two types of experiments are designed about the questions above which

are:

1. comparison of our heuristic's performance over some alternative h ,
2. comparison of different factors effects on our heuristic's performance.

Computational comparison is used for the first part. Section 5.2 gives alternatives to see if there is an improvement gained by the developed heuristic. For comparison purpose, performance measures defined in Section 5.1 are used.

For the second part, analysis of variance technique is used. It is applied on the factors defined in Section 5.1 as discussed in Section 5.3. The significance of the factors are checked on the chosen performance measures.

5.1 Experimental Design

This section provides an introduction to the use of statistical design during performing experiments. Experimental design provides a way of deciding before the runs are made which particular configurations to consider so that the desired information can be obtained with the least amount of work.

In experimental design terminology, the input parameters and structural assumptions composing a model are called factors, and the output performance measures are called responses. To use the statistical approach in designing and analyzing an experiment, it is important to define the choice of factors and the response variables.

The decision as to which parameters are considered fixed aspects of the model and which are experimental factors is given in Subsection 5.1.1. The response variables of the reschedule and performance measures are detailed in Section 5.1.2.

5.1.1 Choice of Factors

As an experimenter, we should choose the factors to be tested in the experiment, the ranges over which these factors can take values, and the specific

levels at which runs must be made. By varying the levels of the chosen factors, purposeful changes are made in order to observe and identify the changes in the reschedule found by the heuristic. After discovering some relations between the factors and outcome, the reasoning of these relations are going to be discussed.

There are two major input variables for the heuristic, initial schedule and the duration of the breakdown. Though the down duration is a measurable quantity, this is not easy for the initial schedule. That causes a need for decomposing the factors that affect the initial schedule. These are classified as idle time percentage in the schedule, the variance of processing times, ready times and due dates. Idle time percentage is an important factor that defines the amount of flexibility in the schedule and this amount is utilized during rescheduling. As this percentage increases, opportunities of performing a match-up is expected to be increased. Since the other three variables are expected to affect the distribution of idle times throughout the planning horizon, they are considered as factors. These factors can cause idle times to be placed in large quantities or small, and this affects the frequency of the idle times even when the total amount of idleness is same for both cases. Also the distribution of idle times may vary, they can be collected in a small portion of the horizon, not spread over the schedule, due to these three variables. If the ready times are uniformly distributed over the makespan, idle times can be evenly distributed. Otherwise all the idleness of the system can be collected in a small region and cannot be used properly by the heuristic.

In Table 5.1, the fixed parameters of the system are represented. Though these variables remain unchanged at the experiment, factors that are given in Table 5.2 are varied within the given ranges. All of the factors have two levels which the runs will be made, called tight and loose. This choice is made by using a combination of practical experience and theoretical understanding of the problem.

After defining five factors with two levels, factorial design is required for this type of experiment. A factorial design means that in each complete trial or replication of the experiment, all possible combinations of the levels of the factors are investigated [Montgomery]. For example, if there are a levels of factor A and b levels of factor B, then each replicate contains all ab treatment combinations. When factors are arranged in a factorial design, they are often

number of machines	10
number of jobs	300
broken-down machine	5 th machine
time of breakdown	starting time of 5 th job
precedence relationships	% 75
mean processing time	4 units
mean idle time	2 units

Table 5.1. Fixed Variables

said to be crossed.

This experimental design is a case with 5 factors, each at only two levels. The levels of the factors are called 'loose' if the allowed flexibility is large and 'tight' otherwise. It is called a 2^5 design. The 2^5 design is particularly useful since it provides the smallest number of runs with which 5 factors can be studied in a complete factorial design. It has 32 treatment combinations. The number of replications is taken to be 5, which means that there are 160 runs in the design. The choice of the response variables will be represented in the following subsection.

5.1.2 Performance Measures

The response variables of the heuristic are chosen to measure the performance of the reschedule. Performance measures are categorized into two groups which measure:

- the quality of the schedule,
- the stability of the schedule.

Factors	Tight	Loose
idle time percentage	0.20	0.40
processing time variability	$U\sim[3, 5]$	$U\sim[1, 7]$
ready time	$U\sim[0, \text{makespan}]$	$U\sim[0, (0.8 * \text{makespan})]$
k (due-date coeff.)	$U\sim[2, 3]$	$U\sim[4, 5]$
breakdown duration	$U\sim[6, 8]$	$U\sim[12, 14]$

Table 5.2. Experimental Factors

Schedule quality focuses on the deviation of the reschedule from the preschedule. Stability measures check the efficiency and effectiveness of the algorithm.

Schedule quality is quantified in terms of two performances. These are:

1. earliness,
2. tardiness.

Earliness is a measure on the first operation of a job and tardiness is a measure on the last operation. Both terms measure the amount of change between the preschedule and the reschedule. The quality improves as the deviation gets smaller.

Schedule stability is measured by three performance measures. They are:

1. computation time,
2. match-up point,
3. number of successful solutions.

Since machine breakdowns occur in real time, the run time of the heuristic is quite important. As the match-up point enlarges, the problem size expands and

computation time increases. It can be said that these two measures are directly proportional. The third item is stated to give an idea about the efficiency of a heuristic since no one of them guarantees finding a feasible match-up point.

A general intuition is that as the schedule becomes unstable, its quality will also reduce. A late match-up point means a large pool of jobs is revised so deviation is increased. In such a case both groups of measures will indicate low performance simultaneously.

These performances stated above are a direct reflection of the two objectives presented in Chapter 3. One objective was the time critical decision making criterion because of the real-time nature of the environment, considered by the schedule stability. Other objective was to minimize deviation to decrease the nervousness of the system, that is checked by the quality of the schedule measure.

5.2 Computational Comparison

In this part of the computational design, the proposed heuristic's performance is going to be compared with some other alternative procedures' performances. The five measures classified in Subsection 5.1.2 will be used for quantitative comparison purpose.

The heuristic developed in the previous chapter has two major steps that are:

1. Determination of the pool with different match-up points on machines which are determined by the proposed feedback mechanism (Variable ΔT_i).
2. Rescheduling the pool by decomposing into partitions and applying the proposed algorithms (Hierarchical Scheduling).

In the literature review of Chapter 2, we have remarked on a study performed by Bean and Birge. Their approach corresponding to the above steps can be summarized as:

1. Determination of the pool with fixed increments without feedback (Fixed ΔT).
2. Rescheduling the pool with the best of the four dispatching rules (Dispatching Rules).

These four rules are shortest processing time (SPT), earliest due date (EDD), R&M heuristic [Morton et al.84], modified R&M heuristic for inserted idleness [Morton 92]. In SPT, jobs are sequenced in nondecreasing order of processing times. EDD sequences by giving priority to the jobs with smaller due dates. R&M rule uses the following priority rule during scheduling:

$$\pi_j = \frac{1}{p_j} \left\{ \frac{-(S_j^+)}{k p_{av}} \right\},$$

where S_j^+ is the positive slack of job j , p_{av} is the average processing times of jobs competing for top priority, k is a free parameter between 1 and 3, and π_j is the priority of job j . For modified R&M rule, the following priority rule is used during scheduling:

$$\pi'_j = \pi_j \left\{ 1 - B \frac{(r_j - T)^+}{p_{min}} \right\},$$

where T is the current time, p_{min} is the minimum processing time of currently competing jobs and B is the correction factor suggested to be equal to 1.6 or 2.

A given pool is scheduled with all of the four rules and the one with the best result is implemented. This alternative is very powerful since best of the four schedules is taken even though R&M heuristic itself performs very good as shown by [Morton 92].

Alternative heuristics are created by combining the different pairs of the two approaches. Moreover, Bean and Birge's first step represented above is tested for three different ΔT values. ΔT is the increase in the match-up time which means the amount of enlargement in the pool at an iteration. In Figure 5.1, the heuristics developed in this way are represented. Since our hierarchical approach is highly correlated with the variable ΔT ; enlargement procedure, obviously it will not perform well with the fixed ΔT enlargement procedure.

	VARIABLE ΔT_k	FIXED ΔT ($\Delta T = TMAX/10$)	FIXED ΔT ($\Delta T = TMAX/20$)	FIXED ΔT ($\Delta T = TMAX/40$)
HIERARCHICAL APPROACH	OUR'S			
DISPATCHING RULES	ALT 1	ALT 2	ALT 3	ALT 4

Figure 5.1. Alternative Heuristics For Comparison

It should be acknowledged that our heuristic does not allow any earliness for the purpose of considering other production planning activities like material flow, while the alternatives do. Since other planning decisions are given in conjunction with the initial schedule, when a job is scheduled to an earlier during rescheduling, the suppliers may not have enough time to deliver the material on the required time. Extra earliness allows extra flexibility, so they have an advantage that we regret to have in order not to violate the flow plans. After running each alternative algorithm on the problems created by the experimental design described in Section 5.1, all five performance measures are collected. The output of this experiment is represented in Table 5.3.

In terms of tardiness, our heuristic performs worst. This was expected since earliness is forced to be equal to zero during rescheduling while the alternatives allow earliness. The hybrid algorithm, Alternative 1, is the second worst. Among the three algorithms with Fixed ΔT , Alternative 4 gives the lowest tardiness. Obviously, our heuristic dominates others in earliness. Alternative 1 is the second best and Alternative 2 follows it with a small difference. Then earliness sharply increases for Alternatives 3 and 4. But in general, the true

	Tard.	Earl.	Comp.T. (sec.)	Match-up Point	Effcy. %	Overall Perform.
Our's	96	0	0.67	91	100	0.301
Alt 1	86	62	1.04	115	100	0.401
Alt 2	73	72	0.79	225	100	0.437
Alt 3	55	249	0.73	156	100	0.477
Alt 4	53	225	0.81	128	100	0.447

Table 5.3. Comparison of the Average Performances of Alternatives

measure is the total deviation which is the summation of tardiness and earliness. In this case, our heuristic performs better than them with a significant amount of improvement.

For the computation time, the largest value is 1.04 seconds that belongs to Alternative 1 and the smallest is 0.67 seconds that belongs to our heuristic. In this measure, our heuristic performs better again.

Match-up point is also an important measure, that a large match-up value is not desirable. Also here, our algorithm outperforms and the amount of improvement is notable. The match-up time is quite related with the computation time. The reason of our heuristic's low computation time is the low match-up value which decreases the size of the pool to reschedule and the number of iterations.

Efficiency is the percentage of problems that a feasible match-up point is found. With the current conditions, all five alternatives have 100% performance so it is not an identifying measure in this experimental setting.

In order to allow an overall evaluation, a unique measure is desired that includes all of the four performances in it. Efficiency is not included since they all give the same value. Eigenvector Normalization is used for this purpose.

The results are represented in the last column of Table 5.3. They also notify the dominance of our heuristic to the alternatives. Roughly, it gives 25% improvement on the overall measure when compared with the second best alternative.

Alternative 1, which is a mixture of our enlargement procedure and the dispatching rules is the second best heuristic. Its performance measures are quite lower than other algorithms' values but still higher than our heuristic's results. This also indicates that both steps of our algorithm makes significant improvements individually. Algorithm 1 has the variable enlargement idea and is better than the other three alternatives with fixed enlargement while they are all using the same dispatching rules. When we substitute the dispatching rule with the proposed hierarchical scheduling rule, the results become even better.

Among alternatives of 2, 3 and 4, the first one which has the largest amount of enlargement performs better than the other two in the overall. Though, it has a greater match-up point, since the pool includes greater amount of flexibility with a larger pool, the quality of the schedule becomes better. Its earliness is significantly smaller than the other two alternatives. As the enlargement is done with smaller increments, i.e. as ΔT decreases, the match-up point also decreases. But for the computation time, that is just opposite, enlargement with smaller increments cause more iterations, so require longer computation time.

Average figures might sometimes be misleading without considering the dispersion of the data. In Table 5.4 ranges of the results are represented to give an idea about the variance. For tardiness, the interval is too wide for Alternative 1. The reason is that this alternative has a case that matches at the end of horizon while for the remaining cases, this value is quite small. The massive dispersion in the computation time and match-up point is because of this unique special case. For the match-up point, both the minimum values of our heuristic and Alternative 1 are considerably smaller than the other three alternatives. This indicates the dominance of the proposed enlargement procedure over the fixed amount of enlargement strategy. For the maximum match-up value, our heuristic is still significantly smaller than all alternatives.

As a conclusion, our heuristic dominates its alternatives for earliness, total

	Tardiness	Earliness	Computation Time(sec.)	Match-Up Point
Our's	(0 , 1394)	(0 , 0)	(0.47 , 5.33)	(41 , 277)
Alt 1	(0 , 2382)	(0 , 3404)	(0.48 , 24.77)	(43 , 1134)
Alt 2	(0 , 837)	(4 , 341)	(0.58 , 2.43)	(144 , 456)
Alt 3	(0 , 446)	(18 , 618)	(0.48 , 3.73)	(100 , 384)
Alt 4	(0 , 745)	(33 , 581)	(0.50 , 5.40)	(72 , 357)

Table 5.4. Comparison of Performance Ranges of Alternatives

deviation, computation time and match-up point. The difference for earliness is obvious while for the match-up point and total deviation, improvement is in a considerable amount. In the overall performance, it still performs significantly better than all alternatives. In the next section, the response of our heuristic to changes on different variables is going to be investigated.

5.3 Analysis of Variance

The appropriate procedure for testing the equality of several population means is the analysis of variance. In this section we want to test the equality of observed responses from the different treatments of the chosen factors. In other words, the significance of treatment of the factors on the variance is going to be analyzed.

Since our model has five factors with two levels for each factor, 2^5 factorial design is used to estimate of :

1. How each factor affects the responses?
2. How the factors interact with each other (i.e., whether the effect of one

factor depends on the levels of the others)?

The form of the experiment can be compactly represented in a tabular form as in Table 5.5 with the factorial design of the crossed combinations, in which '-' stands for tight, and '+' for loose cases. Such experiments are easy to analyze since each effect is estimated independently of the others. Significance is easy to assess, and, because of the replication, it is possible to detect an interaction between factors if such an interaction exists.

For hypothesis testing, the model error is assumed to be a normally distributed random variable with mean zero and variance σ^2 . The 32 available observations are sufficient to estimate all effects and the value of σ^2 is assumed to remain constant for both levels of the factor. The appropriate analysis is the standard two-way ANOVA (analysis of variance) which states that an effect is considered to be significant if its value is large relative to the estimated value of σ^2 .

Variances are estimated by sum of squares. Determination of the sum of squares of the 32 different orderings becomes a complex task but calculations can be reduced to a simple mechanical formula by Yates' method. A more detailed discussion on the Yates' method can be found in Chapter 7 of [Montgomery].

Among the five performance measures defined in Subsection 5.1.2, ANOVA is applied for only three of them. These are tardiness, computation time and match-up point. Earliness is constant, equal to zero, and efficiency is not an appropriate response for ANOVA because of its very limited outcomes.

For all of the three measures, the analysis of variance are summarized in Table 5.5 through 5.8. Tables confirm the significance levels of 2^5 combinations of factors. The last column gives the Type 1 error of accepting the hypothesis, α that the considered combination has equal effect on the performance with others. These values are computed by Minitab. When α value is extremely small for a factor, then the probability that it is insignificant is quite low. This indicates that the heuristic is sensitive to the changes on that factor.

ANOVA results show that the two factors, A, that is the percentage of idle time in the initial schedule, and C, that is the ready time interval, are significant

Runs	Factors				
	Fac. 1	Fac. 2	Fac. 3	Fac. 4	Fac. 5
1	-	-	-	-	-
2	+	-	-	-	-
3	-	+	-	-	-
4	+	+	-	-	-
5	-	-	+	-	-
6	+	-	+	-	-
7	-	+	+	-	-
8	+	+	+	-	-
9	-	-	-	+	-
10	+	-	-	+	-
11	-	+	-	+	-
12	+	+	-	+	-
13	-	-	+	+	-
14	+	-	+	+	-
15	-	+	+	+	-
16	+	+	+	+	-
17	-	-	-	-	+
18	+	-	-	-	+
19	-	+	-	-	+
20	+	+	-	-	+
21	-	-	+	-	+
22	+	-	+	-	+
23	-	+	+	-	+
24	+	+	+	-	+
25	-	-	-	+	+
26	+	-	-	+	+
27	-	+	-	+	+
28	+	+	-	+	+
29	-	-	+	+	+
30	+	-	+	+	+
31	-	+	+	+	+
32	+	+	+	+	+

Table 5.5. 2^5 Factorial Design

Source of Variation	Sum of Squares	d.f.	Mean Square	F_0	Significance Level(α)
<i>A</i> (Idle Percentage)	618020	1	618020	13.75	0.00031
<i>B</i> (Processing Time Var.)	31753	1	31753	0.71	0.40102
<i>AB</i>	39313	1	39313	0.87	0.35271
<i>C</i> (Ready Time)	429733	1	429733	9.56	0.00244
<i>AC</i>	250114	1	250114	5.56	0.01989
<i>BC</i>	13177	1	13177	0.29	0.59116
<i>ABC</i>	2117	1	2117	0.05	0.82342
<i>D</i> (Duc Date)	20748	1	20748	0.46	0.49885
<i>AD</i>	21809	1	21809	0.49	0.48520
<i>BD</i>	20115	1	20115	0.45	0.50355
<i>ABD</i>	22562	1	22562	0.50	0.48079
<i>CD</i>	5108	1	5108	0.11	0.74068
<i>ACD</i>	3186	1	3186	0.07	0.79176
<i>BCD</i>	4796	1	4796	0.11	0.74068
<i>ABCD</i>	3478	1	3478	0.08	0.77776
<i>E</i> (Breakdown Duration)	176491	1	176491	3.93	0.04957
<i>AE</i>	34928	1	34928	0.78	0.37880
<i>BE</i>	41538	1	41538	0.92	0.33929
<i>ABE</i>	30802	1	30802	0.69	0.40771
<i>CE</i>	41602	1	41602	0.93	0.33668
<i>ACE</i>	18276	1	18276	0.41	0.52312
<i>BCE</i>	76038	1	76038	1.69	0.19593
<i>ABCE</i>	30526	1	30526	0.68	0.41112
<i>DE</i>	141	1	141	0.00	1.00000
<i>ADE</i>	462	1	462	0.01	0.92049
<i>BDE</i>	697	1	697	0.02	0.88776
<i>ABDE</i>	73	1	73	0.00	1.00000
<i>CDE</i>	68	1	68	0.00	1.00000
<i>ACDE</i>	403	1	403	0.01	0.92049
<i>BCDE</i>	40	1	40	0.00	1.00000
<i>ABCDE</i>	51	1	51	0.00	1.00000
Error	5754374	128	44956		
Total	7692540	159			

Table 5.6. Analysis of Variance for Tardiness

Source of Variation	Sum of Squares	d.f.	Mean Square	F_0	Significance Level(α)
<i>A</i> (Idle Percentage)	3.07	1	3.07	9.00	0.00325
<i>B</i> (Processing Time Var.)	0.02	1	0.02	0.07	0.79176
<i>AB</i>	0.06	1	0.06	0.18	0.67209
<i>C</i> (Ready Time)	2.84	1	2.84	8.32	0.00460
<i>AC</i>	2.30	1	2.30	6.73	0.01059
<i>BC</i>	0.12	1	0.12	0.35	0.55516
<i>ABC</i>	0.18	1	0.18	0.54	0.46378
<i>D</i> (Due Date)	0.01	1	0.01	0.04	0.84180
<i>AD</i>	0.01	1	0.01	0.03	0.86277
<i>BD</i>	0.02	1	0.02	0.05	0.82342
<i>ABD</i>	0.01	1	0.01	0.02	0.88776
<i>CD</i>	0.01	1	0.01	0.03	0.86277
<i>ACD</i>	0.01	1	0.01	0.03	0.86277
<i>BCD</i>	0.00	1	0.00	0.00	1.00000
<i>ABCD</i>	0.00	1	0.00	0.00	1.00000
<i>E</i> (Breakdown Duration)	0.41	1	0.41	1.19	0.28961
<i>AE</i>	0.24	1	0.24	0.71	0.40102
<i>BE</i>	0.68	1	0.68	1.98	0.16181
<i>ABE</i>	0.55	1	0.55	1.61	0.20678
<i>CE</i>	0.40	1	0.40	1.16	0.29552
<i>ACE</i>	0.19	1	0.19	0.55	0.45968
<i>BCE</i>	0.68	1	0.68	1.99	0.16076
<i>ABCE</i>	0.68	1	0.68	2.00	0.15972
<i>DE</i>	0.02	1	0.02	0.05	0.82342
<i>ADE</i>	0.03	1	0.03	0.09	0.76467
<i>BDE</i>	0.00	1	0.00	0.01	0.92049
<i>ABDE</i>	0.03	1	0.03	0.08	0.77776
<i>CDE</i>	0.01	1	0.01	0.02	0.88776
<i>ACDE</i>	0.00	1	0.00	0.01	0.92049
<i>BCDE</i>	0.01	1	0.01	0.03	0.86277
<i>ABCDE</i>	0.01	1	0.01	0.02	0.88776
Error	43.73	128	0.34		
Total	56.34	159			

Table 5.7. Analysis of Variance for Computation Time (CPU)

on the response of our heuristic relative to other factors. This conclusion is same for all of the three performance measures. One level of factor C is that ready times are distributed throughout the makespan and the other level is that they are distributed on the interval covering the initial 80% of the makespan. The second level causes a decrease on the idle percentage of the initial schedule since in this case the possibility of waiting idle for a job to be ready reduces. Hence both factors A and C decrease the total amount of idleness in the system. More idleness means more flexibility in a system so, reduction in the idleness amount of the initial schedule reduces the feasibility range of match-up and increases the complexity.

Factors B, that is the processing time variability, and D, that is the due date, are exactly insignificant on the performance for all of the three measures. In fact, these different performance measures' results are quite parallel. Their ANOVA results suggest the same factors as significant and insignificant.

For factor E, that is the duration of time that the machine is down, the situation is different. It cannot be directly classified as significant or not. It is significant at 5 percent for tardiness while it is only significant at 30 percent for computation time and match-up point. Though it is expected to have more effect on the response, this result is reasonable in the sense that when down duration becomes larger, this reduces quality of the schedule but does not have a big impact on the number of iterations. Therefore, both the match up point and the computation time are slightly affected by the down duration. A brief summary of the three tables' ANOVA results are represented in Figure 5.2. Their effect on the performance of the algorithm is categorized into four groups as high, medium, low and not significant to make the results manageable.

In this chapter, the heuristic proposed in Chapter 4 is experimented. In the first part, its performance is compared with four alternatives. Performance is evaluated in terms of the quality of the resulting schedule and time criticality of the heuristics which are our two basic objectives stated in Chapter 3. Our heuristic results better than alternatives for both type of measures. In the second half of the study, the effects of the different factors on the performance of our heuristic is tested. It is concluded that idle time percentage and range of ready times in the initial schedule have significant influence on the results relative to other factors. Also duration of time that the breakdown machine

Factors	Tard.	Comp. Time	Match. Point
A (Idle Time%)	high	high	high
B (Pr.TimeVar)	n.s.	n.s.	n.s.
C (ReadyT.Int)	high	high	high
D (Due Date)	n.s.	n.s.	n.s.
E (DownDur.)	medium	low	low

high : significant at 0.5 % level

medium : significant at 5 % level

low : significant at 30 % level

n.s. : not significant

Figure 5.2. Extract of ANOVA Results

relative to other factors. Also duration of time that the broktdown machine is off has some effect though not as much as the other two variables.

Chapter 6

Conclusion

The research presented in this thesis develops a framework for rescheduling manufacturing cells when machine breakdown invalidates the preplanned schedule. The strategy utilizes the match-up idea of Bean and Birge which indicates that an optimal recovery from a single disruption does indeed return to the preschedule under certain conditions. It reschedules to match-up with the preschedule at some time in the future so that the state reached by the revised schedule is the same as that reached by the initial schedule.

We model a cellular manufacturing system with machine breakdowns and assume that the preschedule can be followed if no disruptions occur. After a machine breakdown, a match-up point is determined and part of the initial schedule that covers the time interval between the disruption and match-up time is rescheduled. We approached to this problem heuristically because of the computational complexity of determining both the match-up point and rescheduling in this determined match-up point simultaneously.

Our heuristic approach initially proposes a match-up point, then reschedules the job and machine pool defined with the match-up point. Though two basic steps are similar to that in Bean and Birge, the solution approach for each step is quite different. Initially the match-up point search procedure is improved. Instead of searching the time horizon with equal increments, a feedback mechanism is developed to determine the amount of increase of the match-up value at each iteration. Match-up value of each machine is different since they all have different conditions in terms of flexibilities they have. Then,

a specific scheduling heuristic is proposed which considers the extra constraints of the rescheduling problem that classical approaches do not. It decomposes the flow shop into three sets, the breakdown machine, downward machines and upward machines, and develops different algorithms for each set.

We developed an experimental frame where the proposed approach is compared with alternative methods and tested the significance of some factors by analysis of variance test. For total deviation, match-up point and computation time, our heuristic approach dominates other alternatives. We totally eliminate earliness in order to keep the new schedule consistent with the flow plan. It gives 25% improvement on the overall measure when compared with the second best alternative. Also it results 31% reduction for the overall cost when compared with the best of the three alternatives created on the heuristic by Bean & Birge.

Analysis of variance tests are performed for five factors. Four of them are idle time percentage in the schedule, the variance of processing times, ready times and due dates that shape the initial schedule and the last one is the breakdown duration. Test results show that two factors which are the percentage of idle time in the initial schedule, and the ready time interval are highly significant on the response of our heuristic relative to other factors. Since, we are trying to exploit the flexibilities inherent in the initial schedule when the disruptions occur.

In our heuristic approach, during determining the match-up point, the ideas of collecting flexibility in the rescheduling pool, and using a feedback mechanism are used. As denoted in the computational comparison results, Alternative 1, which is a mixture of our enlargement procedure and the classical dispatching rules is the second best heuristic, which means that even only the proposed match-up determination strategy improves the performance measures. The second improvement is made by developing a rescheduling procedure where the combination of our match-up determination strategy and the rescheduling procedure performs better than all alternatives.

Finally, we state two potential research areas related to our study:

- **Considering other disruption reasons :** Disruptions like unexpected new jobs, rush or hot jobs with high priorities, material delays, change

Appendix A

The Dominance Property

A.1 Proof of the Dominance Rule

Proof of Theorem 4.1 is done by using the geometric interpretation of the function $f(j, t)$. For any given two jobs, i and j , one of them is chosen to be scheduled at the instance depending on their $f(i, t)$ and $f(j, t)$ functions. The apsis of the intersection of these lines is called as t^* where:

$$t^* = \frac{p_j d_i - p_i d_j}{p_j - p_i}$$

The ordinate of the intersection is called as f^* where:

$$f^* = f(i, t^*) = f(j, t^*)$$

The current time that a choice should be made is called as t^c . For jobs i and j , tardiness cost of job j after choosing job i is denoted as $T_{i < j}^j$ and total cost of i and j after this decision is denoted as $T_{i < j}$ where:

$$T_{i < j} = T_{i < j}^i + T_{i < j}^j$$

To be used in the proof of Theorem 4.1, an auxiliary theorem is stated.

Theorem A.1. *Given two jobs that are both ready at time t^c , where $p_i \leq p_j$ and $t^c \geq t^*$,*

1. *if $t^c \leq t^*$, then $f(i, t^c) < 0$, $f(j, t^c) \geq 0$ case is infeasible,*

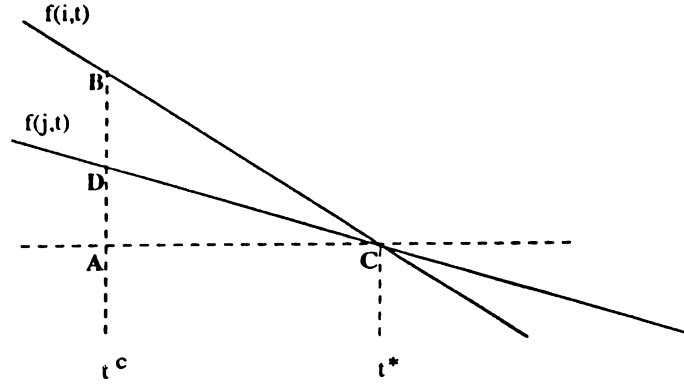


Figure A.1. Case 1.

2. if $t^c > t^*$, then for $f(j, t^c) < 0$, $f(i, t^c) \geq 0$ case is infeasible.

Proof: The intersection point of lines $f(i, t)$ and $f(j, t)$ is (t^*, f^*) .

Case 1. $t^c \leq t^*$

This case is illustrated in Figure A.1.

By using triangle $\triangle DAC$,

$$\text{slope of } f(j, t) = \frac{|DA|}{|AC|}$$

$$\Rightarrow \frac{1}{p_j} = \frac{f(j, t^c) - f^*}{|AC|}$$

$$\Rightarrow f(j, t^c) = \frac{|AC|}{p_j} + f^*$$

By using triangle $\triangle BAC$,

$$\text{slope of } f(i, t) = \frac{|BA|}{|AC|}$$

$$\Rightarrow \frac{1}{p_i} = \frac{f(i, t^c) - f^*}{|AC|}$$

$$\Rightarrow f(i, t^c) = \frac{|AC|}{p_i} + f^*$$

Since $p_j > p_i$,

$$\frac{|AC|}{p_j} < \frac{|AC|}{p_i}$$

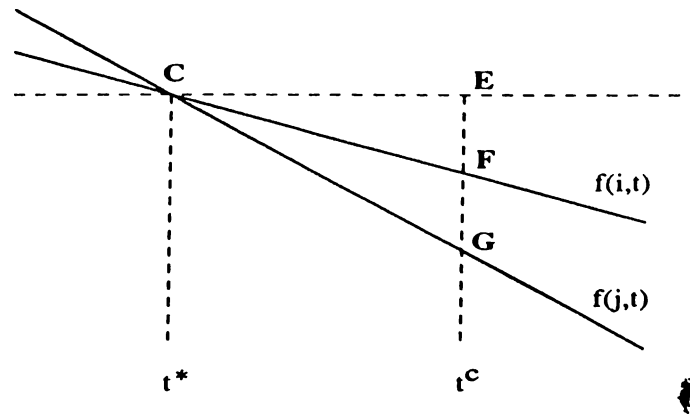


Figure A.2. Case 2.

$$\Rightarrow \frac{|AC|}{p_j} + f^* < \frac{|AC|}{p_i} + f^*$$

$$\Rightarrow f(j, t^c) < f(i, t^c)$$

So if $f(i, t^c) < 0$ is given, then $f(j, t^c) < f(i, t^c) < 0$

Case 2. $t^c > t^*$

This case is illustrated in Figure A.2.

By using triangle $\triangle CEF$,

$$\text{slope of } f(j, t) = \frac{|EF|}{|CE|}$$

$$\Rightarrow \frac{1}{p_j} = \frac{f(j, t^c) - f^*}{|CE|}$$

$$\Rightarrow f(j, t^c) = \frac{|CE|}{p_j} + f^*$$

By using triangle $\triangle CEG$,

$$\text{slope of } f(i, t) = \frac{|EG|}{|CE|}$$

$$\Rightarrow \frac{1}{p_i} = \frac{f(i, t^c) - f^*}{|CE|}$$

$$\Rightarrow f(i, t^c) = \frac{|CE|}{p_i} + f^*$$

Since $p_j > p_i$,

$$\begin{aligned}
-\frac{1}{p_j} &> -\frac{1}{p_i} \\
\Rightarrow f^* - \frac{|AC|}{p_j} &< f^* - \frac{|AC|}{p_i} \\
\Rightarrow f(j, t^c) &> f(i, t^c)
\end{aligned}$$

So if $f(j, t^c) < 0$ is given, then $f(i, t^c) < f(j, t^c) < 0$

After the proof of this auxiliary theorem, Theorem 4.1. is developed as follows.

Theorem 4.1. *Given two jobs i and j with $p_i \leq p_j$, both ready at time t^c , the given rules minimize the tardiness.*

1. For $p_i \neq p_j$, $t^* > t^c$,

if $f(i, t^c) \geq 0$ and $f(j, t^c) \geq 0$, then use **EDD**,
if $f(i, t^c) \geq 0$ and $f(j, t^c) < 0$ then if $d_i - p_j \geq t^c$ $j \prec i$,
else $i \prec j$,
if $f(i, t^c) < 0$ and $f(j, t^c) < 0$, then use **SPT**.

2. For $p_i \neq p_j$, $t^* \leq t^c$, use **SPT**.

3. For $p_i = p_j$, use **EDD**.

Proof:

Case 1. For $p_i \neq p_j$, $t^* > t^c$, there are three possibilities:

1.1. $f(i, t^c) \geq 0$ and $f(j, t^c) \geq 0$,

1.2. $f(i, t^c) \geq 0$ and $f(j, t^c) < 0$,

1.3. $f(i, t^c) < 0$ and $f(j, t^c) < 0$.

The fourth combination,

$$f(i, t^c) < 0 \text{ and } f(j, t^c) \geq 0$$

is an infeasible case which is proved above in Theorem A.1.

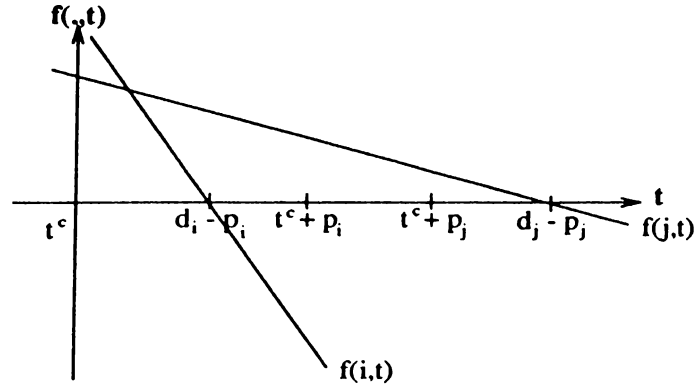


Figure A.3. Case 1.1.ii for $d_j - p_j > d_i - p_i$

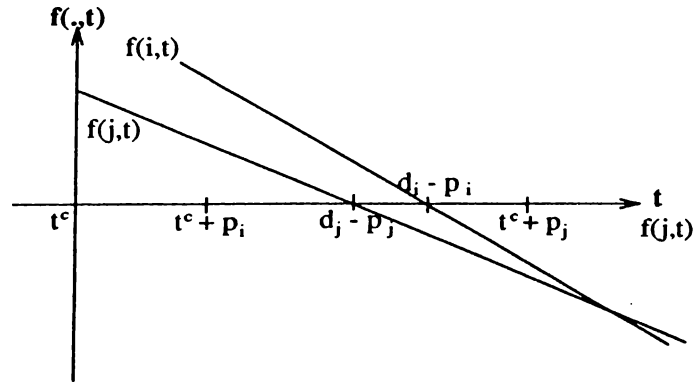


Figure A.4. Case 1.1.ii for $d_j - p_j < d_i - p_i$

Case 1.1. $f(i, t^c) \geq 0$ and $f(j, t^c) \geq 0$

For this case, $T_{i \prec j}^i = 0 \wedge T_{j \prec i}^j = 0$

i. $t^c + p_i \leq d_j - p_j \wedge t^c + p_j \leq d_i - p_i$

$$T_{i \prec j}^j = 0 \wedge T_{j \prec i}^i = 0$$

$$T_{i \prec j} = T_{j \prec i} = 0$$

\Rightarrow Indifferent between i and j for tardiness.

ii. $t^c + p_i \leq d_j - p_j \wedge t^c + p_j > d_i - p_i$

This case is possible when either $d_j - p_j > d_i - p_i$

or $d_i - p_i > d_j - p_j$

ii.a $d_j - p_j > d_i - p_i$

This case is illustrated in Figure A.3.

ii.b $d_j - p_j < d_i - p_i$

This case is illustrated in Figure A.4.

In both cases, $T_{i \prec j} = 0$

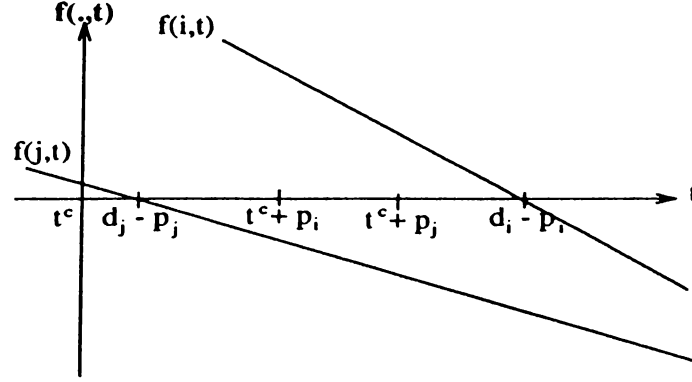


Figure A.5. Case 1.1.iii

$$T_{j \prec i} = T_{j \prec i}^i = t^c + p_j - (d_i - p_i)$$

$$\Rightarrow i \prec j$$

$$\text{Since } t^c + p_i \leq d_j - p_j \wedge t^c + p_j > d_i - p_i,$$

$$t^c + p_i + p_j \leq d_j \wedge t^c + p_i + p_j > d_i$$

$$\Rightarrow d_i < d_j, \text{ that is EDD.}$$

$$\text{iii. } t^c + p_i > d_j - p_j \wedge t^c + p_j \leq d_i - p_i$$

This case is feasible only when $d_j - p_j < d_i - p_i$
It is illustrated in Figure A.5.

$$T_{i \prec j} = T_{i \prec j}^j = t^c + p_i - (d_j - p_i)$$

$$T_{j \prec i}^i = 0$$

$$\Rightarrow j \prec i$$

$$\text{Since } t^c + p_i > d_j - p_j \wedge t^c + p_j \leq d_i - p_i,$$

$$t^c + p_i + p_j > d_j \wedge t^c + p_i + p_j \leq d_i$$

$$\Rightarrow d_j < d_i, \text{ that is EDD.}$$

$$\text{iv. } t^c + p_i > d_j - p_j \wedge t^c + p_j > d_i - p_i$$

$$T_{i \prec j} = T_{i \prec j}^j$$

$$T_{j \prec i} = T_{j \prec i}^i$$

$$T_{j \prec i} - T_{i \prec j} = T_{j \prec i}^i - T_{i \prec j}^j$$

$$= t^c + p_j - (d_i - p_i) - (t^c + p_i - (d_j - p_j))$$

$$= d_j - d_i$$

\Rightarrow that is EDD.

Case 1.2. $f(i, t^c) \geq 0$ and $f(j, t^c) < 0$

For this case, $T_{i \prec j}^i = 0 \wedge T_{i \prec j}^j > 0 \wedge T_{j \prec i}^j > 0$

$$\text{i. } t^c + p_j \leq d_i - p_i$$

$$T_{j \prec i}^i = 0$$

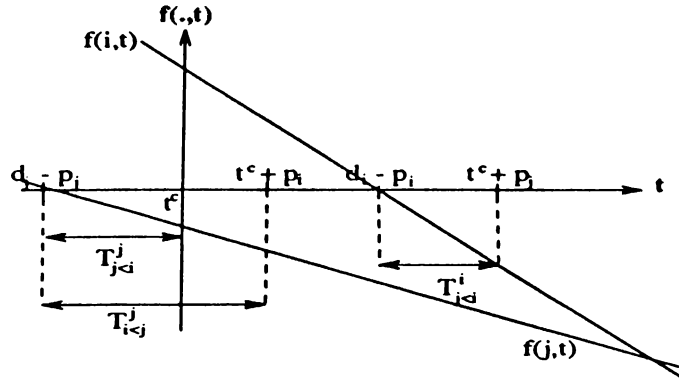


Figure A.6. Case 1.2.ii

$$\begin{aligned}
 T_{j<i} &= T_{j<i}^j \quad \wedge \quad T_{i<j} = T_{i<j}^j \\
 T_{j<i} - T_{i<j} &= T_{j<i}^j - T_{i<j}^j \\
 &= t^c - (d_j - p_j) - (t^c + p_i - (d_j - p_j)) \\
 &= -p_i < 0
 \end{aligned}$$

$$\Rightarrow j < i$$

ii. $t^c + p_j > d_i - p_i$

$$\begin{aligned}
 T_{j<i}^i &> 0 \\
 T_{j<i} &= T_{j<i}^j + T_{j<i}^i \quad \wedge \quad T_{i<j} = T_{i<j}^j \\
 T_{j<i} - T_{i<j} &= T_{j<i}^i + T_{j<i}^j - T_{i<j}^j \\
 &= t^c + p_j - (d_i - p_i) + t^c - (d_j - p_j) \\
 &\quad - (t^c + p_i - (d_j - p_j)) \\
 &= t^c + p_j - d_i
 \end{aligned}$$

This case is illustrated in Figure A.6.

If $t^c > d_i - p_i$, then $i < j$.

If $t^c \leq d_i - p_i$, then $j < i$.

Case 1.3. $f(i, t^c) < 0$ and $f(j, t^c) < 0$

This case is illustrated in Figure A.7.

$$\begin{aligned}
 T_{i<j} &= T_{i<j}^i - T_{i<j}^j \\
 T_{j<i} &= T_{j<i}^i - T_{j<i}^j \\
 T_{j<i} - T_{i<j} &= T_{j<i}^i + T_{j<i}^j - (T_{i<j}^i + T_{i<j}^j) \\
 &= t^c + p_j - (d_i - p_i) + t^c - (d_j - p_j) \\
 &\quad - (t^c - (d_i - p_i) + t^c + p_i - (d_j - p_j)) \\
 &= p_j - p_i > 0
 \end{aligned}$$

$$\Rightarrow i < j$$

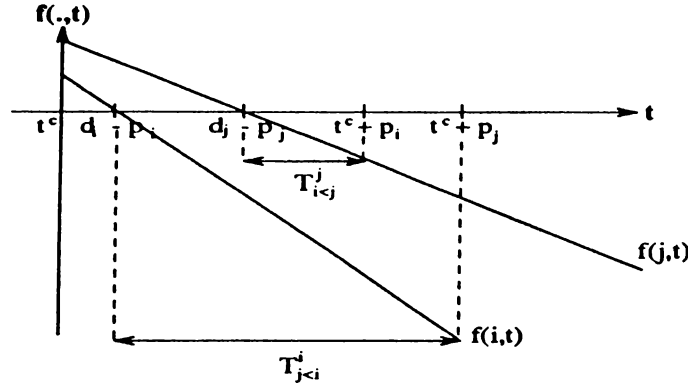


Figure A.7. Case 2.1.iv

Case 2. For $p_i \neq p_j$, $t^* \leq t^c$, there are three possible cases:

- 2.1. $f(i, t^c) \geq 0$ and $f(j, t^c) \geq 0$,
- 2.2. $f(i, t^c) \geq 0$ and $f(j, t^c) < 0$,
- 2.3. $f(i, t^c) < 0$ and $f(j, t^c) < 0$.

Case 2.1. $f(i, t^c) \geq 0$ and $f(j, t^c) \geq 0$

For this case, $T_{i < j}^i = 0 \wedge T_{j < i}^j = 0$

i. $t^c + p_i \leq d_j - p_j \wedge t^c + p_j \leq d_i - p_i$

$$\Rightarrow T_{i < j}^j = 0 \wedge T_{j < i}^i = 0$$

$$\Rightarrow T_{i < j} = T_{j < i} = 0$$

\Rightarrow Indifferent between i and j for tardiness.

ii. $t^c + p_i \leq d_j - p_j \wedge t^c + p_j > d_i - p_i$

$$\Rightarrow T_{i < j}^j = 0 \wedge T_{j < i}^i > 0$$

$$\Rightarrow T_{i < j} = 0 \wedge T_{j < i} > 0$$

$$\Rightarrow i < j$$

iii. $t^c + p_i > d_j - p_j \wedge t^c + p_j \leq d_i - p_i$

This case is infeasible.

$$d_i - p_i < d_j - p_j$$

$$t^c + p_i > d_j - p_j \Rightarrow t^c + p_i > d_i - p_i$$

$$\Rightarrow t^c + p_i > t^c + p_j$$

$$\Rightarrow p_i > p_j \text{ which is a contradiction.}$$

iv. $t^c + p_i > d_j - p_j \wedge t^c + p_j > d_i - p_i$

This case is illustrated in Figure A.8.

$$T_{i < j} = T_{i < j}^j$$

$$T_{j < i} = T_{j < i}^i$$

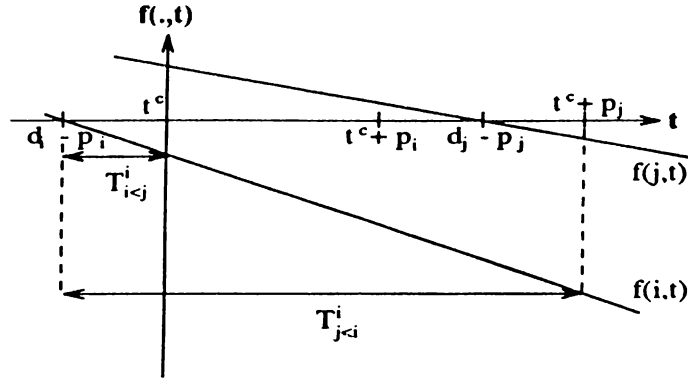


Figure A.8. Case 2.2.i

$$\begin{aligned}
 T_{j < i} - T_{i < j} &= T_{j < i}^i - T_{i < j}^j \\
 &= t^c + p_j - (d_i - p_i) - (t^c + p_i - (d_j - p_j)) \\
 &= d_j - d_i > d_j - d_i + (p_i - p_j) \text{ since } p_i - p_j < 0 \\
 &> d_j - p_j - (d_i - p_i) > 0
 \end{aligned}$$

$$\Rightarrow i < j$$

$$\Rightarrow p_i < p_j, \text{ that is SPT.}$$

Case 2.2. $f(i, t^c) < 0$ and $f(j, t^c) \geq 0$

For this case, $T_{j < i}^j = 0 \wedge T_{i < j}^i > 0 \wedge T_{j < i}^i > 0$

i. $t^c + p_i \leq d_j - p_j$

This case is illustrated in Figure A.9.

$$T_{i < j} = T_{i < j}^i$$

$$T_{j < i} = T_{j < i}^i$$

$$\begin{aligned}
 T_{j < i} - T_{i < j} &= T_{j < i}^i - T_{i < j}^i \\
 &= t^c + p_j - (d_i - p_i) \\
 &\quad - (t^c + p_i - (d_j - p_j)) \\
 &= p_j > 0
 \end{aligned}$$

$$\Rightarrow i < j$$

$$\Rightarrow p_i < p_j, \text{ that is SPT.}$$

ii. $t^c + p_i > d_j - p_j$

This case is illustrated in Figure A.10. $T_{i < j} = T_{i < j}^i + T_{i < j}^j$

$$T_{j < i} = T_{j < i}^i$$

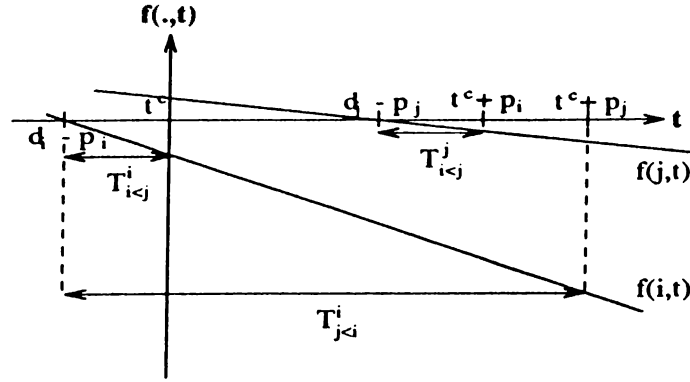


Figure A.9. Case 2.2.ii

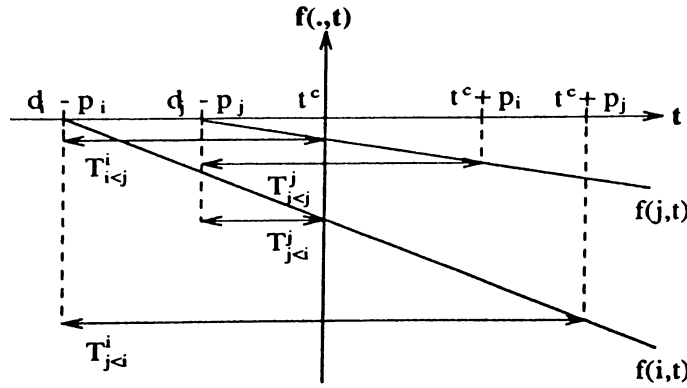


Figure A.10. Case 2.3.i

$$\begin{aligned}
 T_{j < i} - T_{i < j} &= T_{j < i}^i - (T_{i < j}^i + T_{i < j}^j) \\
 &= t^c + p_j - (d_i - p_i) \\
 &\quad - (t^c - (d_i - p_i) + t^c + p_i - (d_j - p_j)) \\
 &= d_j - t^c - p_i \\
 &> d_j - t^c - p_j > 0 \text{ since } p_i < p_j
 \end{aligned}$$

$$\Rightarrow i < j$$

$$\Rightarrow p_i < p_j, \text{ that is SPT.}$$

Case 2.3. $f(i, t^c) < 0$ and $f(j, t^c) < 0$

For this case, $T_{j < i}^j > 0 \wedge T_{i < j}^j > 0$

$$T_{i < j}^i > 0 \wedge T_{j < i}^i > 0$$

$$T_{i < j} = T_{i < j}^i + T_{i < j}^j$$

$$T_{j < i} = T_{j < i}^i + T_{j < i}^j$$

i. $t^* \leq d_i - p_i$

This case is illustrated in Figure A.11.

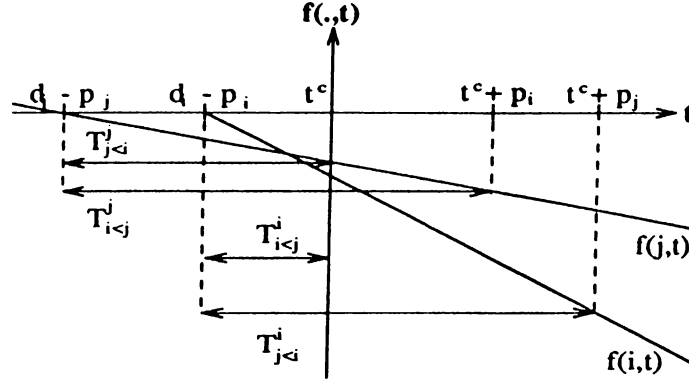


Figure A.11. Case 2.3.ii

$$\begin{aligned}
 T_{j<i} - T_{i<j} &= T_{j<i}^i + T_{j<i}^j - (T_{i<j}^i + T_{i<j}^j) \\
 &= t^c + p_j - (d_i - p_i) + t^c - (d_j - p_j) \\
 &\quad - (t^c - (d_i - p_i) + t^c + p_i - (d_j - p_j)) \\
 &= p_j - p_i > 0
 \end{aligned}$$

$$\Rightarrow i < j$$

$$\Rightarrow p_i < p_j, \text{ that is SPT.}$$

ii. $d_i - p_i < t_* < t_c$

This case is illustrated in Figure A.12.

$$\begin{aligned}
 T_{j<i} - T_{i<j} &= T_{j<i}^i + T_{j<i}^j - (T_{i<j}^i + T_{i<j}^j) \\
 &= t^c + p_j - (d_i - p_i) + t^c - (d_j - p_j) \\
 &\quad - (t^c - (d_i - p_i) + t^c + p_i - (d_j - p_j)) \\
 &= p_j - p_i > 0
 \end{aligned}$$

$$\Rightarrow i < j$$

$$\Rightarrow p_i < p_j, \text{ that is SPT.}$$

A.2 Transitivity Property for the Dominance Rule

In the case of more than two job choices at a given time, the Dominance Rule can be applicable only if it has transitive property. Its existence is checked by Theorem A.2.

Theorem A.2. For any three jobs i, j and k ready at the current time, the

following proposal is true:

If $i \prec j$ and $j \prec k$, then $i \prec k$.

Proof: When a precedence relation is given between two jobs i and j , to conclude whether which case of the Dominance Rule is applied, the intersection point of the $f(i, t)$ and $f(j, t)$ functions, t^* , should be determined. For three jobs, there are three intersection points, $t_{i,j}^*$, $t_{j,k}^*$ and $t_{i,k}^*$. There are two possible cases for an intersection point, it is either greater than t^c or equal or smaller, which means 2^3 different combinations to consider. For each job, $f(\cdot, t^c)$ values can be negative or positive. Also there are 2^3 different combinations which means that each case has 8 subcases. Here, only the case where,

$$t_{i,j}^* > t^c, t_{j,k}^* > t^c \text{ and } t_{i,k}^* > t^c$$

is considered with its 32 subcases. The proof for other cases are quite similar.

Case 1. $f(i, t^c) \geq 0, f(j, t^c) \geq 0$ and $f(k, t^c) \geq 0$

EDD is the valid rule for all pairs.

$$\begin{aligned} i \prec j \wedge j \prec k &\Rightarrow d_i \prec d_j \wedge d_j \prec d_k \\ &\Rightarrow d_i \prec d_k \\ &\Rightarrow i \prec k \end{aligned}$$

Case 2. $f(i, t^c) \geq 0, f(j, t^c) \geq 0$ and $f(k, t^c) < 0$

For i and j EDD is valid, j, k and i, k pairs are sequenced due to the case 1.2 of Theorem 4.1.

$$\begin{aligned} i \prec j \wedge j \prec k &\Rightarrow d_i \prec d_j \wedge t^c > d_j - p_k \\ &\Rightarrow t^c > d_i - p_k \\ &\Rightarrow i \prec k \end{aligned}$$

Case 3. $f(i, t^c) \geq 0, f(j, t^c) < 0$ and $f(k, t^c) < 0$

For j and k SPT is valid, i, j and i, k pairs are sequenced due to the case 1.2 of Theorem 4.1.

$$\begin{aligned} i \prec j \wedge j \prec k &\Rightarrow t^c > d_i - p_j \wedge p_i < p_k \\ &\Rightarrow t^c > d_i - p_k \\ &\Rightarrow i \prec k \end{aligned}$$

Case 4. $f(i, t^c) < 0, f(j, t^c) < 0$ and $f(k, t^c) < 0$

SPT is the valid rule for all pairs.

$$\begin{aligned} i \prec j \wedge j \prec k &\Rightarrow p_i < p_j \wedge p_i < p_k \\ &\Rightarrow p_i < p_k \\ &\Rightarrow i \prec k \end{aligned}$$

Case 5. $f(i, t^c) \geq 0, f(j, t^c) < 0$ and $f(k, t^c) \geq 0$

i, j and j, k are sequenced due to the case 1.2 of Theorem 4.1. *EDD* is valid for i, k pair.

$$\begin{aligned} i \prec j \wedge j \prec k &\Rightarrow t^c > d_i - p_j \wedge t^c > d_k - p_j \\ &\Rightarrow d_i < d_k \\ &\Rightarrow i \prec k \end{aligned}$$

Case 6. $f(i, t^c) < 0, f(j, t^c) \geq 0$ and $f(k, t^c) \geq 0$

i, j and i, k are sequenced due to the case 1.2 of Theorem 4.1. *EDD* is valid for j, k pair.

$$\begin{aligned} i \prec j \wedge j \prec k &\Rightarrow t^c > d_j - p_i \wedge d_j < d_k \\ &\Rightarrow t^c < d_k - p_i \\ &\Rightarrow i \prec k \end{aligned}$$

Case 7. $f(i, t^c) < 0, f(j, t^c) \geq 0$ and $f(k, t^c) < 0$

i, j and j, k are sequenced due to the case 1.2 of Theorem 4.1. *SPT* is valid for i, k pair.

$$\begin{aligned} i \prec j \wedge j \prec k &\Rightarrow t^c \leq d_j - p_i \wedge d_j < d_k \\ &\Rightarrow t^c < d_k - p_i \\ &\Rightarrow i \prec k \end{aligned}$$

Case 8. $f(i, t^c) < 0, f(j, t^c) < 0$ and $f(k, t^c) \geq 0$

i, k and j, k are sequenced due to the case 1.2 of Theorem 4.1. *SPT* is valid for i, j pair.

$$\begin{aligned} i \prec j \wedge j \prec k &\Rightarrow p_i < p_j \wedge t^c \leq d_k - p_j \\ &\Rightarrow t^c < d_k - p_i \\ &\Rightarrow i \prec k \end{aligned}$$

Bibliography

- [Abraham et al. 85] Chacko Abraham, Brenda Dietrich, Stephen Graves, Candace Yano, 'A Research Agenda for Models to Plan and Model Manufacturing Systems,' Technical Report, Dept. of Industrial and Operations Eng., The Univ. of Michigan (July 1985).
- [Adams et al.87] J. Adams, E. Balas, D. Zawack, 'The Shifting Bottleneck Procedure for Job Shop Scheduling,' *Management Science*, 34, 391-401 (1988).
- [Baker 74] Kenneth R. Baker, *Introduction to Sequencing and Scheduling*, John Wiley and Sons (1974).
- [Bean 84] James C. Bean, 'A Lagrangian Algorithm for the Multiple Choice Integer Program,' *Operations Research*, 32, 1185-1193 (1984).
- [Bean&Birge 85] James C. Bean, John R. Birge, 'Match-up Real-Time Scheduling,' Technical Report 85-22, Dept. of Industrial and Operations Eng., The Univ. of Michigan (June 1985).
- [Bean et al. 91] James C. Bean, John R. Birge, John Mittenehal, 'Match-up Scheduling with Multiple Resources, Release Dates and Distruption,' *Operations Research*, Vol 39, No:3, pp 470-483 (May-June 1991).
- [Biefield&Cooper 90] Erich Biefield, Lynne Cooper, 'Operations Mission Planner,' JPL Publication 90-16 (March 1990).

- [Birge 85] John R. Birge, 'Real-Time Adaptive Scheduling in Flexible Manufacturing Systems,' Technical Report 85-26, Dept. of Industrial and Operations Eng., The Univ. of Michigan (August 1985).
- [Conway 91] Richard Conway, 'Problems and Implications of Continuous-Time Scheduling,' Production and Operations Management Society Annual Meeting (1991).
- [Davis&Jones 88] Wayne J. Davis, Albert T. Jones, 'A Real-Time Production Scheduler for a Stochastic Manufacturing Environment,' *Int. J. Computer Integrated Manufacturing*, Vol 1, No:2, pp 101-112 (1988).
- [Davis&Jones 89] Wayne J. Davis, Albert T. Jones, 'Using On-Line Simulation in Production Scheduling,' Conference Proceedings: NSF Design and Manufacturing Systems Grantees Conference, Advances in Design and Manufacturing Systems Tempe, Arizona (Jan 8-12, 1990).
- [Du&Leung 90] J. Du, J.Y. Leung, 'Minimizing Total Tardiness on One Machine is NP-hard,' *Mathematics of Operations Research*, 15 (1990).
- [Dudek et al. 92] R. A. Dudek, S. S. Panwalkar, M. L. Smith, 'The Lessons of Flow-shop Scheduling Research,' *Operations Research*, Vol 40, No:1, pp 1-7 (Jan-Feb 1992).
- [Erschler 89] Jacques Erschler, Francois Roubellat, 'An Approach to Solve Workshop Real-Time Scheduling Problems,' NATO ASI Series, Vol F53, Advanced Information Technologies for Industrial Material Flow Systems (1989).
- [Fox 90] Mark S. Fox, 'Constraint-Guided Scheduling, A Short History of Research at CMU,' *Computers in Industry*, 14, pp 79-88 (1990).
- [Graves 81] Stephen C. Graves, 'A Review of Production Scheduling,' *Operations Research*, Vol 29, No:4, pp 646 - 675 (July-Aug 1981).

- [Kan 76] Rinnooy Kan, A. H. G, *Machine Scheduling Problems: Classification, Complexity and Computation*, Nijhoff (1976).
- [Lawler 89] Eugene L. Lawler, Jon Karel Lenstra, Rinnooy Kan, David Shmoys, 'Sequencing and Scheduling: Algorithms and Complexity' Report BS-R8909 Center for Mathematics and Computer Science, Netherland (1989).
- [McKay 88] Kenneth N. McKay, Frank R. Safayeni, John A. Buza-cott, 'Job Shop Scheduling Theory: What is Relevant?,' *Interfaces*, 18, 4, pp 84 - 90 (July-Aug 1988).
- [McKenzie 76] Lionel W. McKenzie, 'Turnpike Theory,' *Econometrica*, Vol 44, No:5, pp 841-866 (Sep 1976).
- [Montgomery] Douglas C. Montgomery, *Design and Analysis of Experiments*, Third Edition, Wiley (1991).
- [Morton et al.84] T.E. Morton, R.V. Rachamadugu, A.P.J. Vepsalainen, 'Accurate Myopic Heuristics for Tardiness Scheduling,' Working Paper 36-83-84, GSIA, Carnegie Mellon Univ., (1984).
- [Morton 92] Thomas E. Morton, *Heuristic Scheduling Systems With Applications to Production Systems and Project Management*, unpublished manuscript (July 1992).
- [Ow&Smith 88] Peng Si Ow, Stephen F. Smith, 'Viewing Scheduling as an Opportunistic Problem Solving Process,' *Annals of Operations Research*, 12, pp 85 - 108 (1988).
- [Odrey&Wilson 87] Nicholas G. Odrey, George R. Wilson, 'An On-Line Control Structure for Flexible Manufacturing Systems,' 14th Conference on Production Research and Technology, University of Michigan (Oct 6-9, 1987).
- [Odrey&Wilson 90] Nicholas G. Odrey, George R. Wilson, 'Hierarchical Planning and Control for a Flexible Manufacturing

- Shop,' Conference Proceedings: NSF Design and Manufacturing Systems Grantees Conference, Advances in Design and Manufacturing Systems Tempe, Arizona (Jan 8-12, 1990).
- [Ow et al. 88] Peng Si Ow, Stephen F. Smith, Alfred Thiriez, 'Reactive Plan Revision,' *AAAI* (1988).
- [Parunak 87] H. Van Dyke Parunak, 'Why Scheduling Is Hard? (and How to Do It Anyway?),' Technical Report DSFG 87 - 9 (1987).
- [Rodammer] Frederick A. Rodammer, K. Preston White, 'Design Requirements For a Real-Time Production Scheduling Decision Aid,' Real-Time Optimization in Automated Manufacturing Facilities, NBS Special Publication 724, 401-422 (1986).
- [Pinedo 83] M. L. Pinedo, 'Stochastic Scheduling With Release Dates and Due Dates,' *Operation Research*, Vol 31, pp 559-572 (1983).
- [Pape&Smith 87] Claude Le Pape, Stephen F. Smith, 'Management of Temporal Constraints for Factory Scheduling,' Technical Report CMU-RI-TR-87-13, Carnegie Mellon Univ. (1987).
- [Smith 87] Stephen F. Smith, 'A Constraint-Based Framework for Reactive Management of Factory Schedules,' Proceedings Internal Conference on Expert Systems and the Leading Edge in Production Planning and Control, Charleston, South Caroline (May 1987).
- [Smith 92] Stephen F. Smith, 'OPIS: A Methodology and Architecture for Reactive Scheduling,' Technical Report, The Robotics Institute, Carnegie Mellon Univ. (Sept 1992).
- [Smith et al. 90] Stephen F. Smith, Peng Si Ow, Nicda Muscettola, Jean-Yves Potvin, Dirk C. Matthy, 'An Integrated Framework for Generating and Revising Factory Schedules,' *J. of*

- the Operational Research Society*, Vol 41, No 6, 539-552 (1990).
- [Saleh 88] Abdol Saleh, 'Real-Time Control of a Flexible Manufacturing Cell,' PhD Dissertation, Lehigh University (1988).
- [Saleh et al. 91] A. Saleh, N.G. Odrey, G.R. Wilson, 'Design and Algorithmic Implementation of a Real-Time Controller for a Manufacturing Cell,' PED, Vol 53, Design, Analysis and Control of Manufacturing Cells ASME (1991).
- [Walden et al. 92] Eugene Walden, C. V. Ravishankar, 'Algorithms for Real-Time Scheduling Problems,' Technical Report CSE-TR-92-91, Computer Science and Engineering Division, The Univ. of Michigan (1991).
- [Winston] Wayne L. Winston, *Operations Research: Applications and Algorithms*, PWS-KENT Publishing Co., Chp 18, Turnpike Theory, pp 796-797 (1987).
- [Wu&Wysk 89] Szu Yung David Wu, Richard A. Wysk, 'An Application of Discrete Event Simulation to On-line Control and Scheduling in Flexible Manufacturing,' *International Journal of Production Research*, Vol 27, No:9, 1989, pp 1603-1623 (1989).

VITAE

Elif Görgülü is born in 1969 in Bursa. She is graduated from Ankara Fen Lisesi in 1987. She received B.S. and M.S. degrees in Industrial Engineering from Bilkent University. She is currently a research assistant in Bilkent University. Her research interests include scheduling, production planning and inventory control. She is a member of ORSA, TIMS and IIE.