SCHEDULING PREVENTIVE MAINTENANCE ON A SINGLE MACHINE: A MACHINING CONDITIONS BASED APPROACH

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

Sinan Gürel

December, 2002

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. M. 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. Meral Azizoğlu

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.

Assist. Prof. Alper Şen

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet B. Baray
Director of the Institute

ABSTRACT

SCHEDULING PREVENTIVE MAINTENANCE ON A SINGLE MACHINE: A MACHINING CONDITIONS BASED APPROACH

Sinan Gürel
M.S. in Industrial Engineering
Supervisor: Assoc. Prof. M. Selim Aktürk
December, 2002

As the complexity of the production systems increase, condition based maintenance becomes a more preferred preventive maintenance (PM) approach. On CNC cutting machines, machining conditions (machining speed and feed rate) selection affects the machine's PM need. Higher production rates result in higher wear and failure rates. So, when determining machining conditions for an operation, we need to consider the PM cost in addition to the manufacturing costs. We first define a relationship between processing time of a job and PM need of the machine. We propose a machining condition selection method which minimizes PM and manufacturing costs simultaneously.

When scheduling a set of jobs, machining conditions decision for each job affects performance measure of the schedule. The minimization of total completion time with machine availability constraint was shown to be an NP-hard problem. In our problem we have nonlinear relationship between the PM need of the machine and the processing times of the jobs. For a given sequence of jobs, decreasing processing times of jobs decreases processing time effect on completion times of jobs and increases PM visit effect on completion times of jobs due to more PM visits. There is a tradeoff which should be considered when selecting machining conditions, equivalently processing time for each job. We provide optimality properties for the problem. We design a search algorithm to determine processing times for the jobs and their schedule on a single CNC machine simultaneously.

Keywords: Scheduling, Preventive Maintenance, Condition Based Maintenance, Total Completion Time, Machining Conditions, Controllable Processing Time, Heuristics.

ÖZET

TEK MAKİNADA KORUYUCU BAKIM ÇİZELGELEME: İMALAT KOŞULLARI TEMELLİ YAKLAŞIM

Sinan Gürel
Endüstri Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Doç. Dr. M. Selim Aktürk
Aralık, 2002

Üretim sistemleri geliştikçe ve karmaşıklaştıkça durum bazlı bakım daha çok tercih edilen bir koruyucu bakım yöntemi haline gelmektedir. CNC takım tezgahlarında seçilen imalat koşulları makinanın bakım ihtiyacını etkilemektedir. Makinanın üretim hızını arttırdığımızda aşınma ve arızalar artmaktadır. Bu yüzden, imalat koşullarını belirlerken üretim maliyetleri yanında koruyucu bakım maliyetleri de dikkate alınmalıdır. Bu çalışmada ilk önce bir iş için üretim zamanı ile koruyucu bakım maliyetini ilişkilendiriyoruz. Üretim ve koruyucu bakım maliyetleri toplamını enazlayan imalat koşulları belirleme metodu öneriyoruz.

Bir makinada işleri çizelgelerken, işlerin seçilen imalat koşulları çizelgenin performansını etkiler. Makinanın çalışmadığı zaman aralıkları sözkonusu iken toplam iş bitim zamanını enazlama probleminin NP-zor olduğu biliniyor. Bu çalışmada aynı zamanda işlem zamanıyla koruyucu bakım ihtiyacı arasındaki doğrusal olmayan ilişki de gözönünde bulunduruluyor. Bir çizelgedeki işlerin işlem zamanlarını azalttığımızda makinanın koruyucu bakım ihtiyacı artıyor. Bu da toplam iş bitim zamanı üzerindeki işlem zamanı etkisini azaltmakla beraber artan koruyucu bakım işleri nedeniyle koruyucu bakım etkisini arttırıyor. Bu yüzden işlerin imalat koşullarını ya da işlem zamanlarını belirlerken bu ters ilişkiyi dikkate almamız gerekiyor. Bu çalışmada, sözü edilen problem için gerekli enazlama kriterleri sunuluyor. Bu kriterler ışığında işlerin işlem sürelerini belirleyen ve aynı zamanda işleri çizelgeleyen bir tarama algoritması öneriliyor.

Anahtar sözcükler: Çizelgeleme, Koruyucu Bakım, Durum Bazlı Bakım, Toplam İş Bitim Süresi, İmalat Koşulları, Değişken İşlem Zamanı, Sezgisel Algoritmalar.

Acknowledgement

I would like to express my gratitude to Assoc. Prof. M. Selim Aktürk for his valuable and perpetual guidance and encouragement during my study. His supervising with patience and interest made this thesis possible.

I am grateful to Assoc. Prof. Meral Azizoğlu and Assist. Prof. Alper Şen for reading and reviewing this thesis and their valuable comments and suggestions.

I would like to thank to my family for their support to bring this thesis to an end.

Last but not the least, I dedicate this thesis to my fiancee, Yeşim Yurttutan, who always supported me and listened patiently to all my endless monologues on this study.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Preventive maintenance (PM) is a set of operations which is designed to preserve and increase equipment reliability. A correctly applied PM program provides a significant increase in production capability. An ideal PM program prevents failure of all equipment. Some typical PM operations for a CNC cutting machine are cleaning units of machine, refilling lubrication system, refilling cutting fluid, various inspections, testing machine movements, and so on.

As the needs of the industry have changed, different PM policies have arose. Traditionally, preventive maintenance visits to machines are made periodically (daily, weekly, etc). In this approach, mainly, PM is applied after a predefined time period since the previous PM visit. Of course some variations of this idea have been used in practice. For instance, if a failure occurs on a machine, besides repairing, PM operations are also applied to the machine instead of waiting for the planned PM time. But, the idea is not to allow machine to stay without being visited for PM longer than the predefined time.

An alternative PM approach is measuring total operating time (or output, a parameter which gives an idea for the work done by the machine) and maintaining the machine after each predefined operating time or operating parameter. This approach, which uses measures like operating time or some parameter related to work done by machine, is based on more realistic approach that a machine or

a system needs more maintenance as it does more work. Although a machine staying idle also needs PM after some time, this approach is a good one for the machines running most of the time.

Two approaches mentioned above are based on the assumption and the fact that machines or systems deteriorate or lose smooth operating conditions and operation quality by time. Another approach which has completely different philosophy is monitoring some condition parameters (vibration, temperature, etc.) of the machine and when these parameters are below or above certain levels, maintaining the machine. This method is just based on monitoring the machine continuously or in discrete time points. This policy, in fact, what we call predictive maintenance is a preventive maintenance approach based on observing the status of the machine, so that, it predicts when machine is close to a failure and takes action against failure. This is a condition driven PM program.

We can see examples where both or some of these policies or variants of these policies are implemented in industry. Usually, which policy to be implemented is a decision affected by the needs and maintenance capabilities of a firm. Equipment of a firm may demand more sensitive and effective PM approach depending on its importance for production system and its technical properties. Periodic maintenance is easier to manage than measurement based policies, where measurement based policies are more effective in terms of maintenance and cost.

Surely, we can say that development of competition and technology requires firms to work more efficiently, which makes PM a more important tool for production. For example, in energy plants energy generation interruptions must be avoided because each hour that they do not run plant they lose money. They prefer to monitor the condition of their critical equipment continuously, so that they realize approaching failure and take action against breakdowns. They use computerized systems for condition monitoring and observing the equipment and to store data about equipment. So, PM being a cost item for a firm, in fact, reduces costs of firm which come from losses due to down-time and equipment deterioration.

Meanwhile, a firm is willing to make more intensive maintenance to a machine

if and only if that machine's contribution to the system and its economic value is high enough. So, amount of PM to be applied is a decision which effects the gain of PM. Higher PM amount does not always mean higher gain.

A firm's maintenance management system's capabilities affect which policy to apply. If a firm has a maintenance system which is able to collect measurements from machines and process them to generate work orders and manage the work orders, then that firm can use any policy. If they do not create a database to store their equipment and maintenance data and do not make analysis of this data, they are completely unaware of what is going on.

In this study, we are concerned with CNC cutting machines in a flexible manufacturing environment. These systems are controlled by computers. We can control tool changes, job part changes and cutting parameters automatically on CNC machines. So, they are expensive, and difficult to operate and control. They require careful decision making and scheduling of operations, tool changes and maintenance to maximize the utilization of the system.

Here, different then the traditional policies, we are interested with the effect of the machining parameters (machining speed and feed rate) on a machine's PM need and introduce a more sensitive PM application. We know that higher production rate for a cutting machine would result higher deterioration and higher PM need. Here, we will try to characterize the relationship between machining conditions and PM need for a CNC cutting machine. We will introduce PM cost as a function of processing time and provide a method to minimize manufacturing and PM cost for a single machining operation.

Motivated by and using the relationship between machining conditions and PM, we are dealing with the total completion time problem for a single machine with PM intervals. The current literature mostly ignores the availability constraint for a manufacturing environment when dealing with scheduling problems. In practice we could have an availability constraint for any machine due to set up, PM, tool change, etc. Also, scheduling jobs and PM visits together are important issue for maintenance team scheduling problem for a system with limited maintenance team resource.

Properties of the total completion problem for a single machine with a PM constraint and processing time variability will be presented as well as an effective search algorithm, which makes sequencing decisions and machining conditions decisions at the same time.

In Chapter 2, we present an overview of the current literature. It covers a review of studies on machining conditions selection, machine scheduling with processing time variability as well as machine scheduling with an availability constraint. In Chapter 3, we discuss the PM and machining conditions relationship. Single machining operation problem, including PM cost, will be defined and a method to solve this problem will be provided in the same chapter. Then, in Chapter 4 we introduce the total completion time minimization problem motivated by our findings on PM in Chapter 3. We provide definition of the problem and present some optimality properties for the problem. In Chapter 5, we propose heuristic algorithms for the problem. A subproblem definition and three heuristics proposed as well as a local search algorithm based on this subproblem are given. Then, in Chapter 6 we provide a numerical example to discuss features of the algorithms. In Chapter 7, we present the experimental design and discuss the results. Finally, we conclude with Chapter 8 where we give our final remarks and future research directions.

Chapter 2

Literature Review

In the literature both preventive maintenance management and scheduling are studied extensively. There are some studies which consider the interaction between these two topics. Preventive Maintenance(PM) was concerned in scheduling problems as a constraint of machine availability which makes some of them too hard.

In this study, besides problem of scheduling PM and jobs on a machine we are also dealing with well-known machining conditions optimization problem for single machining operation which was not considered with PM before. We will combine PM cost of a CNC machine with the machining conditions optimization problem. There are studies that concern both machining conditions optimization problem and scheduling problem at the same time.

Here, studies from the literature will be presented to give an idea about the theoretical background of the related topics;such as preventive maintenance, machining conditions selection and scheduling.

2.1 Preventive Maintenance

As the complexity of the manufacturing systems increase; costs of these systems become higher and their PM applications become more critical, in order to keep their asset value by keeping them productive as long as possible. So, PM becomes a more important and a more sophisticated activity for industry, which also motivates much more study on maintenance issues as we can see in the literature. In this section we will give a literature review on PM itself, its growth and development as well as recently published studies which cover different points of views about PM applications.

Having been researched since 1950s and 1960s, there is a rich literature on development and implementation of mathematical models of preventive maintenance planning for different systems. Important surveys influenced the field are by McCall [33], Pierskalla and Voelker [41], Sherif and Smith [44], and Valdez-Flores and Feldman [48]. Also Barlow and Hunter [9] include an in-depth mathematical analysis of preventive maintenance. These studies present various approaches for modeling preventive maintenance.

A survey by Wang [51] summarizes, classifies, and compares various maintenance policies in the literature for both single-unit and multi-unit systems. Different PM policies like age-dependent, periodic, failure limit, sequential and repair limit are described and studies concerning these policies and extensive forms of these policies are classified and compared. Age dependent policy requires a unit to be preventive maintained at some predetermined age T or repaired at failure. Periodic PM policy is to apply PM to equipment at fixed time intervals (kT for $k = 1, 2, ...$) independent of failures on the equipment. According to Wang [51], the most attention receiving PM policies in the literature are age-dependent and periodic PM policies about which many papers are published. Some examples are: McCall [33], Barlow and Proshan [10], Pierskalla and Voelker [41], Osaki and Nakagawa [37], Sherif and Smith [44], Jardine [24], Valdez-Flores and Feldman [48] and Pham and Wang [40].

A smooth explanation and classification of maintenance policies (corrective,

preventive, mixed maintenance) and a suggestion of a model to develop a maintenance strategy in a manufacturing system using stochastic chains were provided by Simeu-Abazi and Sassine [45].

As preventive maintenance idea was accepted and implemented by industry as a tool to reduce failures and unplanned down-time, many companies set up time-based preventive maintenance programs for their equipment. As PM idea developed, as well as time-based PM, condition based approaches which monitor the actual status of the equipment and predict failures based on this data were introduced. Nolden [35] discusses predictive maintenance programs that use sophisticated devices (vibration monitoring, wear particle analyzers, etc.) to predict failures and problems while machine is running. Jardine [23] reviews strategies for implementing smart condition monitoring decisions and focuses identifying the key risk factors among the signals that are obtained during condition monitoring that should be used to predict health of equipment. He also blends economic considerations with risk estimate to establish optimal condition based maintenance decisions. Banjevic et al. [8] present a control-limit policy and software for condition-based maintenance optimization. Mitchell [34] surveys the history of condition monitoring and condition-based maintenance; how condition-based maintenance concept developed in time.

Since 1980s computers have been used widely in maintenance activities. But extensive use is just for the purpose of providing information for management rather than improving maintenance approach. Softwares that keep track of PM schedules and work reports are being widely used. Computers are also used for condition monitoring purposes. But, still their decision making capability is limited.

Besides static maintenance models and condition-based maintenance models, alternative approaches were generated to cope with the dynamic and uncertain nature of manufacturing systems. Meanwhile, different approaches are motivated by complex natures of manufacturing systems. Rather than generalized rough cut PM rules we are more likely to deal with PM issue of each unique system individually.

Gopalakrishnan et al. [18] studied a PM task scheduling model based on historical failure and PM data to cope with dynamic nature of operating environment. They considered prioritizing PM tasks, task rescheduling and multi-skilled workforce availability constraint in their model. Also Eleftherios et al. [22] worked on a throughput dependent periodic maintenance policy for a general production unit using a Markov decision model. It was based on the idea that higher throughput rates result higher deterioration of production system.

In this study different than the studies in the literature, we are relating machining conditions (machining speed and feed rate) on a CNC turning machine to PM need of the machine, so that we can evaluate how a job with certain machining conditions add to PM need of the machine. The suggested model also incorporates PM costs into single machining operation problem in which we are looking for machining conditions for minimum production cost, so in our model we are able to figure out how machining parameter (machining speed and feed rate) decisions can affect PM decisions.

2.2 Machining Conditions

Machining parameters selection for turning operation is a problem which has been studied for a long time. For turning process, machine parameters are machining speed, feed rate and depth of cut. These parameters identify the cost or production rate for an operation. For turning process major constraints that parameter selections must satisfy are surface roughness required for the part being machined and the maximum machine power that could be applied by machine, as stated by Bhattacharya [12].

The tool life equation developed by Taylor [46] which defines the relationship between tool life and machining parameters mentioned above is used for both determining tooling cost which occurs due to loss of tool life by an operation and to determine the tool life constraint for the problem if we have a restriction that an operation must be performed within the predefined tool life.

Different objectives like minimizing production cost, maximizing output production rate or maximizing profit rate have been studied. Mathematical models and solution methods for different objective functions of machine parameter selection problem for turning operation were included in Hitomi [21].

Malakooti and Deviprasad [32] formulated machine parameter selection problem as a multiple objective decision making problem. Three conflicting objectives of minimize total cost, minimize production time and minimize surface roughness were considered and a heuristic approach was discussed. Also in this paper there is a list of seminal studies for machine parameter selection problem area.

Ermer and Kromordihardjo [16] suggested the combination of separable programming and geometric programming for the conversion of the machine parameter optimization model to a linear programming formulation. An analytical tool for the selection of machine parameters in turning to minimize total cost of machining and tooling was provided by Gopalakrishnan and Al-Khayyal [17]. The method they provided is based on geometric programming and it uses the complementary slackness conditions to solve the problem.

Machine parameter selection problem was considered with tool allocation problem at the same time by Akturk and Avci [3]. They proposed a solution procedure to tool allocation and machine parameter selection problem where tool availability was considered. They considered tool life constraint in a geometric programming model and also took some non-machining cost terms like tool replacement and loading times into account.

Akturk and Onen [7] proposed an algorithm to solve lot sizing, tool allocation and machining parameters selection problems simultaneously to minimize total production cost. Akturk [2] developed an exact approach for finding minimum production cost machining parameters and tool allocation simultaneously where there are alternative tools for each operation.

In this study, we are trying to formulate PM cost for a turning operation as a function of machining parameters: machining speed and feed rate. When deciding machining parameters, PM will be a cost factor for us. Also this characterization

will lead us to make extensions for PM scheduling.

2.3 Scheduling

Specifically, machine scheduling is making the decision of which resources will be assigned to which jobs in which sequence and at what time so as to optimize some performance measures. Although efforts on scheduling have been spent since the beginning of the century, after 1950s scheduling publications have been appearing in operations research literature. Especially after 1980s, as computers being widely used in industry which made findings about scheduling to be applicable in practice, scheduling have become an area of research in many different directions with increasing attention paid, as stated by Pinedo [42].

In the literature mostly it is assumed that processing times of jobs are fixed and machine availability is not a constraint. However considerable attention has been paid for these two factors which made scheduling problems more interesting and sometimes more difficult. Our problem is considering both machine availability as a result of PM operations and controllable processing times as a practical issue especially for CNC machining environment.

2.3.1 Controllable Processing Times

Processing time control is an important dimension of research in scheduling, because in a real manufacturing environment, generally, we can control the processing times of the jobs within some technical constraints. Having the control on processing times we have to consider this in our scheduling decisions. Also, at this point, production cost can also be considered since processing a job in a shorter time requires some additional resources to be used which increases cost. So, scheduling problems with controllable processing times may consider cost performance as well as scheduling performance measures.

Vickson [49, 50] consider the problem of single-machine sequencing in which

processing times can be chosen from given sets of feasible times. The cost of performing each job is a linear function of its processing time, and the schedule cost is equal to the sum of two costs: total processing cost and a cost associated with the job completion times. The objective is to find a job sequence and job processing times that minimize the schedule cost. The survey of Nowicki and Zdrzalka [36] lists results for sequencing problems with controllable processing times up to 1990.

Panwalkar and Rajagopalan [39] consider a single-machine sequencing problem in which job processing times are controllable variables with linear costs. Where all jobs have a common due date, the objective is to find optimal processing times, an optimal due date and an optimal sequence which will minimize a cost function containing earliness cost, tardiness cost and total processing cost. They provide some optimality properties for the given problem.

Trick [47] also examines scheduling problems where not only assignment of jobs to machines but also the machining time that jobs use on a machine is decided. He discusses models and provides algorithms for single capacitated machines and multiple capacitated machine for cost and makespan minimization problems. His approach illustrates the exploitation of underlying network structure in combinatorial optimization problems.

Daniels et al. [15] consider a manufacturing system where there are parallel cells of single machines. Each job's processing time in a cell depends on the resource allocated to that cell. They provide formulations, complexity information, important characteristics of optimal solutions, and develop optimal and heuristic solution approaches for two versions of problem. In one version a resource allocation is done and applied through whole production horizon whereas in the other version resource allocation can be changed dynamically at any time during production.

Cheng et al. [14] study scheduling n independent and simultaneously available jobs on m parallel machines where job processing times can be compressed through incurring additional cost of convex function of compression amount.

They consider total compression cost plus total flow time and total compression cost plus sum of earliness and tardiness costs and provide methods to solve them as assignment problems.

Karabati and Kouvelis [25] discuss simultaneous scheduling and optimal processing time decision problem for a multi-product, deterministic flow line operated under a cyclic scheduling approach. They provide a solution method for the processing time decision subproblem first and then develop an iterative procedure including this method to solve both problems at the same time.

2.3.2 Machine Availability

Scheduling with machine availability constraint have been studied for many different machine environments and different objectives. Many results have been achieved. Adiri et al. [1] studied minimizing flow time on a single machine with a single breakdown. They showed that for the case of single breakdown with concave distribution the shortest processing time (SPT) rule minimizes flow time. They also showed that SPT minimizes flow time under multiple breakdowns with exponential distribution. For the deterministic case in which the breakdown times are known, it is proved that finding a schedule which has a flow time less than a given value is NP-complete. For deterministic case they provided a worst-case lower bound of 1/4 for the SPT rule. Deterministic case is also studied by Lee and Liman [29] and they provided a shorter NP-completeness proof as well as they found a new worst-case bound of 2/7 which is tight for the SPT rule.

Lee [31] studies deterministic case of availability in scheduling for various objectives (makespan, total weighted completion time, tardiness and number of tardy jobs) and various machine environments (single and parallel machines). For each case he provided either polynomial optimal algorithm or NP-hardness proof and dynamic programming or heuristic approaches. Lee et al. [26] survey studies on machine scheduling with availability constraints up to 1997.

Qi et al. [43] study total completion time of jobs on a single machine with

PM where maintenance times are decided according to the rule that maintenance must be done after at most a certain time of operation. In previous studies for deterministic cases maintenance intervals are assumed to be known beforehand. The problem is proved to be NP-hard in strong sense and heuristics and a branch and bound algorithm are proposed. A similar study for the same problem of tool change version instead of maintenance is conducted by Akturk et al. [4].

Graves and Lee [20] discuss scheduling semi-resumable jobs and maintenance on a single machine with total weighted completion and minimum and maximum lateness concerns. In this study time for maintenance is a decision variable, too.

Lee and Leon [28] and Lee and Lin [30] study scheduling problems with rate modifying activities. Lee and Leon [28] consider the decisions of when to schedule the rate modifying activity and sequence of jobs to optimize various objectives. Lee and Lin [30] consider scheduling problems with rate modifying activity with random breakdowns.

Lee and Chen [27] consider machine availability constraint for parallel machine environment to minimize total weighted completion time. Two cases, one of which assumes two machines can be maintained at the same time and the other assumes the opposite, were discussed.

In this study, we propose a processing time dependent PM index for each job. The sum of indices between two PM visits will not be greater than 1. So, with this new PM strategy we try to minimize total completion time of a given instance.

2.4 Conclusion

In the literature preventive maintenance, machining conditions and scheduling problems have been studied extensively. Considerable amount of study were made relating the areas above. In the literature effects of machining parameters' on PM were not taken into account, although a simple real life event, the conditions that

we run a machine affect its maintenance needs.

In this study, we relate PM with machining conditions of turning operation and derive a new PM policy. Meanwhile, we introduce PM cost into machining conditions optimization problem so as to consider PM as a cost factor when deciding machining parameters for a job.

There are studies on single-machine scheduling with availability constraint. This study is different from them in a way that we use the relationship between PM and machining conditions in a scheduling environment. We consider both PM management, machining conditions selection and total completion problems on a single machine at the same time.

In industry, usually PM operations are managed by maintenance management softwares like MAXIMO. PM plans are generated and monitored by such programs. Rather than decision making, such programs handle data for the maintenance system. They can also be equipped with tools for decision making. In this study, we aimed to provide theoretical basis for a potential machine scheduling tool for such programs. So that they can be used for optimization purposes as well as an information system.

In this chapter we provided a review of studies related to our study. We emphasized the similarities and differences of our study with the existing studies in the literature. So, we hopefully make the readers able to see where this work falls in the literature.

In the next chapter, we study PM and machining conditions relationship. We introduce the mathematical model for the machining conditions selection problem considering the PM cost. A solution procedure for the problem is also provided.

Chapter 3

PM and Machining Conditions

A CNC cutting machine simply converts electrical energy to mechanical energy which is used to remove some volume of material on the job. During this conversion and application of forces on job, machine elements are subjected to forces. Also, this energy conversion results some energy loss on machine. Under these forces and temperature; fatigue, wear, overheat and etc. take place on machine. The result is the performance loss (gradual failure) of the machine and increased risk of sudden failures.

As we change machining conditions in the way that machine is due to higher load (in terms of power) we can expect the rate of gradual failure on machine is higher and the likelihood of breakdown is higher during a cutting operation.

The idea is explained with a simple but impressive car race example in Black [13]. Suppose you are preparing for a car race. Instead of running the car at 200 kph, you run it at 100 kph, just sufficient to get you where you need to go on time with no waiting when you get there. The car will run much longer before it breaks down. The race is going to the steady and consistent, not the fastest. For the long term we are more likely to operate our machine at medium conditions rather than running it at highest possible rate. So, when scheduling PM visits on a machine we can take machining conditions into consideration as well as machining duration. CNC machines, having the ability to control machining

conditions exactly and being expensive and sensitive, will be considered in this study.

3.1 Outline of a PM Visit in a CNC Environment

In this section, we present which operations take place in a PM visit of a CNC cutting machine. In a typical PM visit for a CNC cutting machine following steps are taken:

- Clean chipping, dust, impurities on machine's various elements (table, machine bed, slides, axes, etc.)
- Clean switches and externally exposed electric devices.
- Check oil level in lubrication system tank. Add, if necessary.
- Check hydraulic working pressure and air inlet pressure.
- Inspect any leakages and if exists take action.
- Inspect cutting fluid tank, remove foreign bodies. Add cutting fluid if necessary.
- Check for any detached parts, loose fasteners or damaged connection points.
- Change air and oil filters if necessary.
- Test machine movements and functions by test program.

The operations above are somewhat generalized and can be applied to every cutting machine environment. As we run the machine at higher power levels we have to make cleaning and inspection operations more frequently. It is more likely to face detached parts and loose fasteners. We need frequent cutting fluid changes and so on. We would face mechanical problems as well as job quality

losses if these operations were not performed. Each PM operation requires man hour and spare parts or material. We need considerable cost and down-time to complete these operations.

3.2 Machining Conditions and PM Cost

As we increase the production rate of a machine by changing machining conditions, or equivalently decreasing processing times, more power is applied on machine. Higher power level as a result of higher production rate increases machine's PM need. Therefore, in this study, we will use production rate of the machine to make decisions about PM.

Given the fact that production rate is a factor on maintenance decisions, we can derive PM rules as follows, if you run the machine at production rates of $r_1 < r_2 < ... < r_n$ then, apply PM to the machine after $T_1 > T_2 > ... > T_n$ hours of operation, respectively. This approach is based on the fact that if you run the machine with certain production rate, after a certain length of time you have to maintain it. For higher production rates you have to visit the machine more frequently. On CNC machines, generally many different types of parts are machined each of which has different processing times. So, the problem is that we do not operate the machine at a constant production rate for all jobs. Then, we have to find a way to overcome the problem of estimating how much a job constitutes to maintenance need of a machine.

First, we estimate the PM cost of a job on a CNC turning machine, then we will consider PM cost of a job in single machining operation problem (SMOP).

3.2.1 PM Cost Function of Production Rate

In this study, PM cost function of a machine for a time period will be used to make PM decisions. This function determines PM cost of a machine over a time period for a given production rate. Of course we do not have such a predefined

function for a system. But, for a machine, by analyzing PM cost figures for different production levels on past data, we can estimate a function and use it for PM decisions. As long as we have a record of PM activities, their costs and corresponding production levels for same time slots, it is not difficult to estimate such a function. A typical maintenance management software like MAXIMO, can provide the historical cost data for PM visits. By doing statistical analysis of past data for maintenance, we can derive a function for PM cost versus production rate. Here, we will use a hypothetical form of this function which is based on real life assumptions.

For any production system, we know that as we increase the production rate we run the machine at higher power level. As we discussed before, applying higher power results with more gradual wear and breakdowns. So, for a given time period if we increase the rate of production we have to make more PM visits to the machine. This means we have to use more spare parts, more labour hours which means paying more maintenance cost. So, our cost function is an increasing function of production rate.

We can easily argue that as we increase production rate, machine becomes more fragile and increments of production rate causes higher deterioration and higher failure risks. At higher production rates marginal increase in PM cost is higher. So, this function is a continuous and strictly convex function.

To decide the form of such a function we also have to answer the question that whether there exists PM cost for an idle machine. If a machine is idle, we have to maintain it so as to keep it ready to operate whenever necessary and to prevent wear by time. A machine must be cleaned and checked at certain frequency, so that it is not left to wear out on shop floor. So, for a given time period we have a fixed minimum PM cost when production rate is zero. The shape of the function described above and to be used in this study is given in Figure 3.1.

In this study we assume that for a given time period T, for a CNC machine we have a PM cost function form as below:

PM Cost $= A + Bx^k$

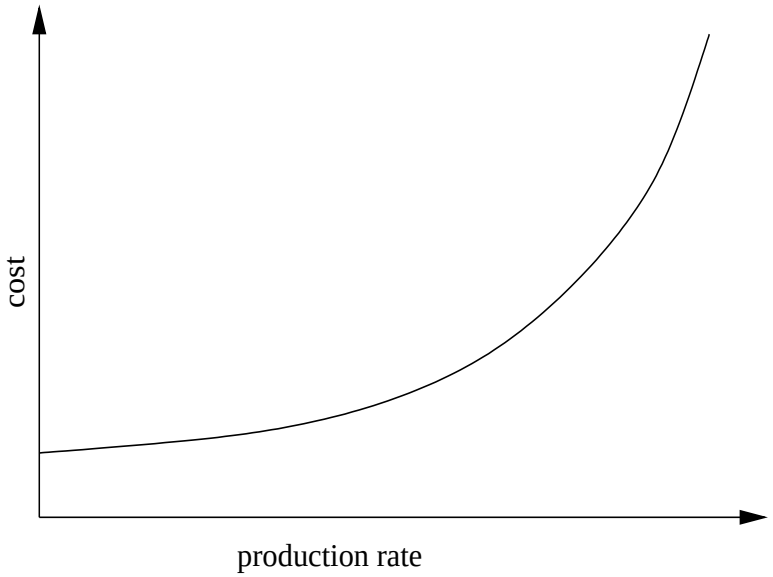

Figure 3.1: PM Cost vs. Production Rate

Here, x is the production rate of the machine (part/min) and A is the PM cost of the machine when it is idle for a time period of T(length of time period that the function is defined for). A, B and k are dependent on T and on cost of a PM visit, C_{pm}. The function is increasing and convex, so we must have $B > 0$ and $k \geq 1$.

Although PM visits might have different durations due to the conditions of the machine at the moment, since a PM visit includes certain operations and checks we can assume that each visit has the same cost and the same duration.

For a CNC turning machine, given the parameters for a job and machining conditions, we are able to calculate processing time, t_m, for the job. So, the arguments that we used above for the relationship between PM cost and production rate can be easily extended to the relationship between PM cost and processing time of a job. If we express production rate as part per minute then we can express production rate of the machine for job type as $x = 1/t_m$. The relationship between PM cost and t_m for a time period of T can be seen in Figure 3.2.

Given the PM cost function of production rate for a CNC turning machine, we

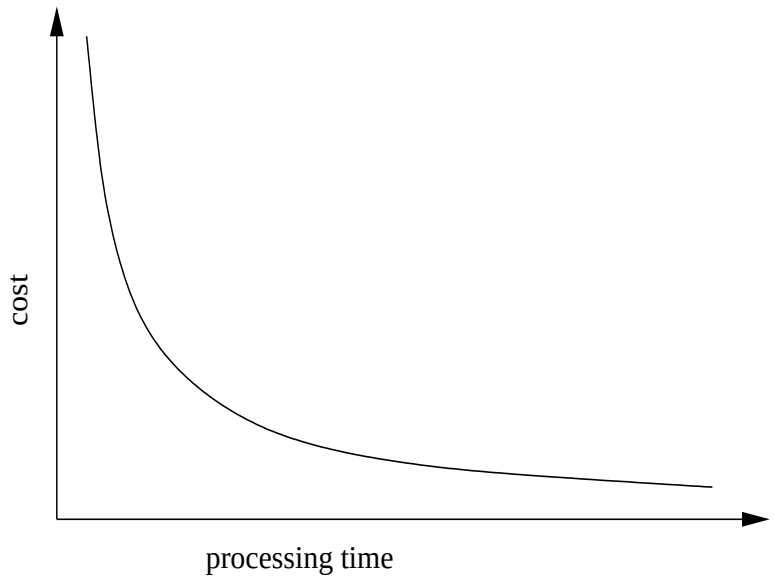

Figure 3.2: PM Cost vs. Processing Time

can express it in terms of processing time which is a function of machining conditions of the job, cutting speed and feed rate. In the literature, single machining operation problem (SMOP) is studied. SMOP focuses on manufacturing costs of CNC turning machines and decides machining conditions for a job. Now, we can also consider PM cost of a job in SMOP when deciding machining conditions, because we are able to handle PM cost of a job in terms of machining conditions.

3.2.2 Calculating PM cost of a job

Suppose we have the PM cost function for a time period of T as below:

PM Cost $= A + Bx^k$

Assuming that we run the machine to operate jobs at processing time of t_m, we can calculate PM cost per job as below:

PM Cost per job $= \frac{At_m^k + B}{Tt_m^{(k-1)}}$ where $x = 1/t_m$.

Another way to find PM cost per job is using the PM interval length. PM

interval length is the maximum time period that the machine can run without PM. Knowing the cost of a PM visit and processing time of the job, we can express PM cost per job as below:

PM Cost per job $= C_{pm}\frac{t_m}{p(t_m)}$

where $p(t_m)$ is the PM interval length for a given processing time (or equivalently, for given production rate). As the cost of PM is a function of processing time, PM interval length is also a function of processing time. By using the equality of two equations above we can derive expression for $p(t_m)$, whose shape is represented in Figure 3.3.

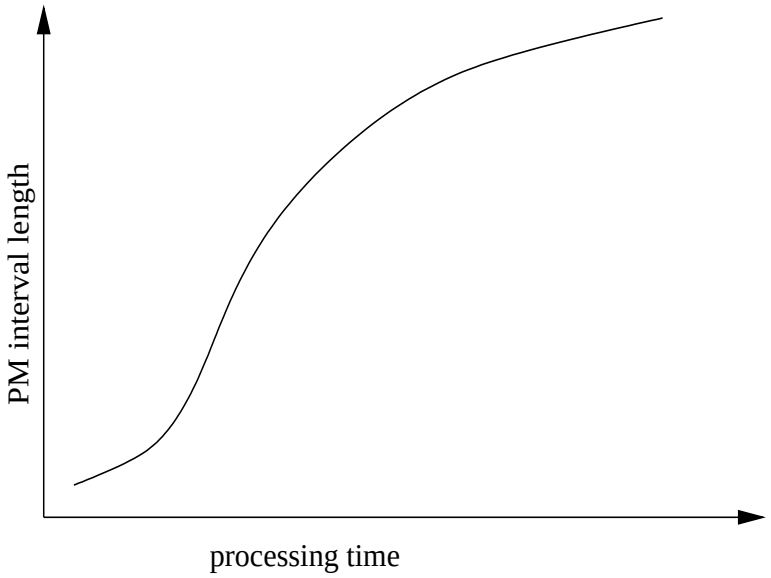

Figure 3.3: PM Interval Length vs. Processing Time

In practice we do not operate all jobs at the same production rate. Each job's machining conditions are decided individually. So, when producing jobs with different processing times we need an objective measure to decide each job's contribution to PM need. The term $\frac{t_m}{p(t_m)}$ is in fact such a measure for us. It gives the ratio of the processing time to corresponding PM interval length. This is a value between 0 and 1, and gives a measure about what portion of a PM visit cost is due to this job. The general expression for PM index of a job described above is as follows;

$$\text{PM index} = \frac{At_m^k + B}{C_{pm} T t_m^{(k-1)}}$$

Figure 3.4 shows the shape of PM index as a function of processing time. We can observe that the index forces us to run the machine at a certain production rate to minimize the PM cost per job. This is because even we keep the machine idle, we have to pay for its PM in order to keep it ready to operate. Also, this shape explains that at higher production rates, equivalently saying lower processing time levels, our PM index sharply increases which means PM cost sharply increases.

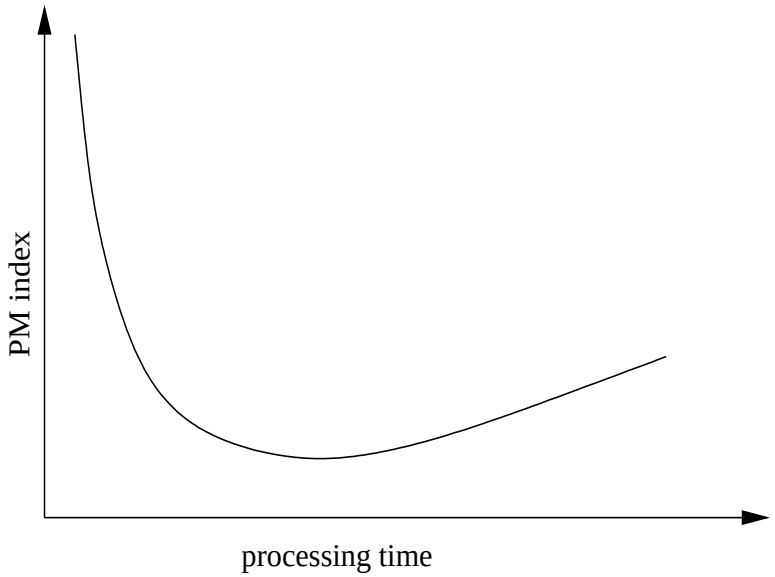

Figure 3.4: PM Index vs. Processing Time

For a set of jobs which are to be processed sequentially, we can easily calculate PM index for each job and sum up all indices to schedule a PM. We put a PM visit when sum of indices of jobs since the last PM visit reaches to 1.

Having defined the PM cost of a job as a function of its processing time we will discuss mathematical model of single machining operation problem in the next section. We will cover PM cost as a factor when selecting machining conditions for a job on CNC turning machine. Now, we can introduce the notation used throughout the study:

Parameters:

α_j, β_j, γ_j : speed, feed, depth of cut exponents for tool j

K_j : Taylor's tool life constant for tool j

C_m, b, c, e : specific coefficients and exponents of the machine power constraint

C_o : operating cost of the CNC machine (\$/min)

C_s, g, h, l : specific coefficients and exponents of the surface roughness constraint

C_{t_j} : cost of the tool j (\$/tool)

D_i : diameter of the generated surface for the operation i (in)

d_i : depth of cut for operation i (in)

H : maximum available machine power for all operations (hp)

L_i : length of the generated surface for the operation i (in)

p_{ij} : number of times that an operation i can be performed by a tool type j

S_i : maximum allowable surface roughness for the operation i (μin)

C_{pm} : cost of preventive maintenance visit (\$/visit)

T_{PM} : duration of a PM visit to the machine (min.)

Decision Variables:

v_{ij} : cutting speed for operation i using tool j (fpm)

f_{ij} : feed rate for operation i using tool j (ipr)

$t_{m_{ij}}$: processing time of operation i using tool j (min)

T_{ij} : tool life of tool j in operation i (min)

U_{ij} : usage rate of tool j in operation i

P_{ij} : PM index for operation i when tool j is used

C_i : completion time for job i

3.3 Single Machining Operation Problem (SMOP)

In preceding sections we have discussed the PM cost and machining conditions relationship and derived mathematical expressions for PM cost of a job and PM index for a job. For a specified machining environment and machining conditions for a particular job, we are able to calculate PM index and PM cost for that job. Here, we will use this established relationship in SMOP. It is important to point out here, that, to the best of our knowledge, any maintenance related cost has not been considered in SMOP before. Having PM cost included in SMOP, we provide mathematical model for SMOP and prove an optimality condition for the problem. Finally, we present an algorithm to solve the problem.

For a given job and operation, we have three cost terms that we consider in SMOP. They are machining cost, tooling cost and PM cost. Machining cost is the function of processing time. Tooling cost is the function of tool usage and PM cost is the function of PM index for a job. We can control processing time, tool usage and PM index for a job by controlling the machining speed and feed rate of the cutting operation on a CNC turning machine. We can set machining speed and feed rate within the limitations (constraints) of machine, tool and job. The constraints that we have to satisfy while setting speed and feed rate are Machine Power, Surface Roughness and Tool Life Constraints. Given the machining conditions and parameters we can calculate processing time for a turning operation as discussed in Gorczyca [19]:

$$t_{m_{ij}} = \frac{\pi D_i L_i}{12 v_{ij} f_{ij}}$$

Tool usage can be calculated by using the processing time equation above and tool life equation of Taylor [46] as below:

$$U_{ij} = \frac{t_{m_{ij}}}{T_{ij}} = \frac{(\pi D_i L_i)/(12 v_{ij} f_{ij})}{K_j/(v_{ij}^{\alpha_j} f_{ij}^{\beta_j} d_i^{\gamma_j})}$$

As discussed before, PM index for a job can be calculated as below:

$$P_{ij} = \frac{A t_{m_{ij}}^k + B}{C_{pm} T t_{m_{ij}}^{(k-1)}}$$

which in terms of v_{ij} and f_{ij} can be expressed as:

$$P_{ij} = \frac{A\pi D_i L_i}{12T} v_{ij}^{-1} f_{ij}^{-1} + \frac{B}{T} \left[\frac{12}{\pi D_i L_i}\right]^{(k-1)} v_{ij}^{(k-1)} f_{ij}^{(k-1)}$$

The objective function for SMOP is as below:

$$M_{ij} = C_o t_{m_{ij}} + C_t U_{ij} + C_{pm} P_{ij}$$

As mentioned above we have three constraints for SMOP. The first one is tool life constraint which limits tool usage for an operation. For instance for our original scheduling problem which will be discussed in following chapters we have the condition that an operation should be completed by at most one tool due to surface quality. So, tool life constraint for our scheduling problem will limit tool usage for an operation by 1. Another constraint is the machine power constraint. Each cutting machine has a maximum power level that it can run at. This cutting power is a function of v and f. So, machine power is a constraint when deciding machining conditions. The last constraint is the surface roughness constraint. Machining conditions affect surface roughness of a job. Increased machining speed decreases surface roughness whereas increased feed rate increases surface roughness. As each job to be machined has a maximum allowable surface roughness level then maximum allowable surface roughness is a constraint for machining conditions decision.

Having explained the constraints and the objective function of the problem, we provide the geometric programming (GP) model for the problem below:

$$\min\ M_{ij} = C_1 v_{ij}^{-1} f_{ij}^{-1} + C_2 v_{ij}^{(\alpha_j - 1)} f_{ij}^{(\beta_j - 1)} + C_3 v_{ij}^{(k-1)} f_{ij}^{(k-1)}$$

Subject to:

$$\begin{aligned}
&\text{(Tool Life Constraint)} \\
C_t' v_{ij}^{(\alpha_j - 1)} f_{ij}^{(\beta_j - 1)} &\leq 1 \\
&\text{(Machine Power Constraint)} \\
C_m' v_{ij}^{b} f_{ij}^{c} &\leq 1 \\
&\text{(Surface Roughness Constraint)} \\
C_s' v_{ij}^{g} f_{ij}^{h} &\leq 1 \\
v_{ij}, f_{ij} &> 0
\end{aligned}$$

where $C_1 = \frac{\pi D_i L_i C_o}{12} + \frac{A\pi D_i L_i}{12T}$, $C_2 = \frac{\pi D_i L_i d_i^{\gamma_j} C_{t_j}}{12K_j}$, $C_3 = \frac{B}{T}\left[\frac{12}{\pi D_i L_i}\right]^{(k-1)}$, $C'_t = \frac{\pi D_i L_i d_i^{\gamma_j} p_{ij}}{12K_j}$, $C'_m = \frac{C_m d_i^e}{H}$ and $C'_s = \frac{C_s d_i^l}{S_i}$.

In Figure 3.5, an example of feasible region for SMOP can be seen. The region under the constraints is the feasible region where we can select machining speed and feed rate.

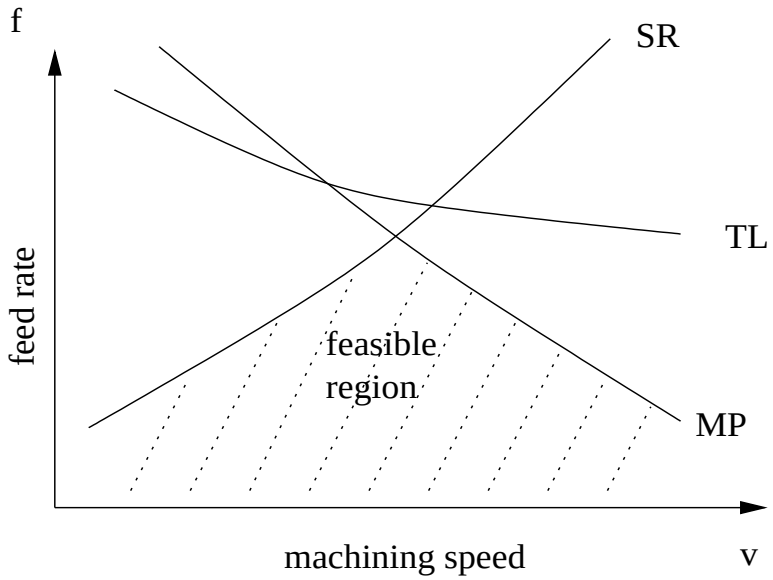

Figure 3.5: A typical feasible region for SMOP

Given the GP model above we will derive conclusions about optimal solution to the problem by using duality and complementary slackness properties.

(Dual GP) $\max \left(\frac{C_1}{Y_1}\right)^{Y_1} \left(\frac{C_2}{Y_2}\right)^{Y_2} \left(\frac{C_3}{Y_3}\right)^{Y_3} (C'_t)^{Y_4}(C'_m)^{Y_5}(C'_s)^{Y_6}$

$$
\begin{aligned}
Subject\ to: \qquad Y_1 + Y_2 + Y_3 &= 1 && (3.1)\\
-Y_1 + (\alpha_j - 1)Y_2 + (k-1)Y_3 + (\alpha_j - 1)Y_4 + bY_5 + gY_6 &= 0 && (3.2)\\
-Y_1 + (\beta_j - 1)Y_2 + (k-1)Y_3 + (\beta_j - 1)Y_4 + cY_5 + hY_6 &= 0 && (3.3)\\
Y_1, Y_2, Y_3, Y_4, Y_5, Y_6 &\geq 0
\end{aligned}
$$

Theorem 1 *The surface roughness constraint must be tight at the optimal solution.*

Proof: Suppose that at optimal solution, surface roughness constraint is loose. Then, the corresponding dual variable for the surface roughness constraint $Y_6 = 0$ due to complementary slackness. When $Y_6 = 0$, constraints 3.2 and 3.3 can not be satisfied at the same time when all dual variables are nonnegative, since $\alpha_j > \beta_j$ and $b > c$ due to Gorczyca [19]. However, in dual problem all variables should be nonnegative. Then, when $Y_6 = 0$, dual problem is infeasible. This proves that at optimality the surface roughness constraint must be tight.

Having shown that surface roughness constraint is tight at the optimal solution for SMOP, the next step is to see how objective function and its terms behave on the surface roughness constraint. In Figure 3.6, the behaviors of machining cost, tooling cost and PM cost for a job as well as total cost as functions of machining speed can be seen. We know that machining cost, tooling cost and PM cost functions are convex. So, being sum of convex functions, our objective function is also convex which can be seen in Figure 3.6.

In Figure 3.5, suppose that we are moving on the surface roughness constraint in the direction that both v and f increases. If v and f increase then t_m decreases, so machining cost decreases, PM cost behaves as PM index in Figure 3.4 and tooling cost increases because U_{ij} increases since $\alpha, \beta > 1$. As we know the behavior of our objective function along the surface roughness constraint as a function of machining speed we will design an algorithm to solve the problem above.

3.3.1 Algorithm for Solving SMOP

By Theorem 1, we know that the surface roughness constraint is always tight at optimality. By using this we can get a one dimensional cost function on the surface roughness constraint since we can derive an analytical expression of one variable as a function of other.

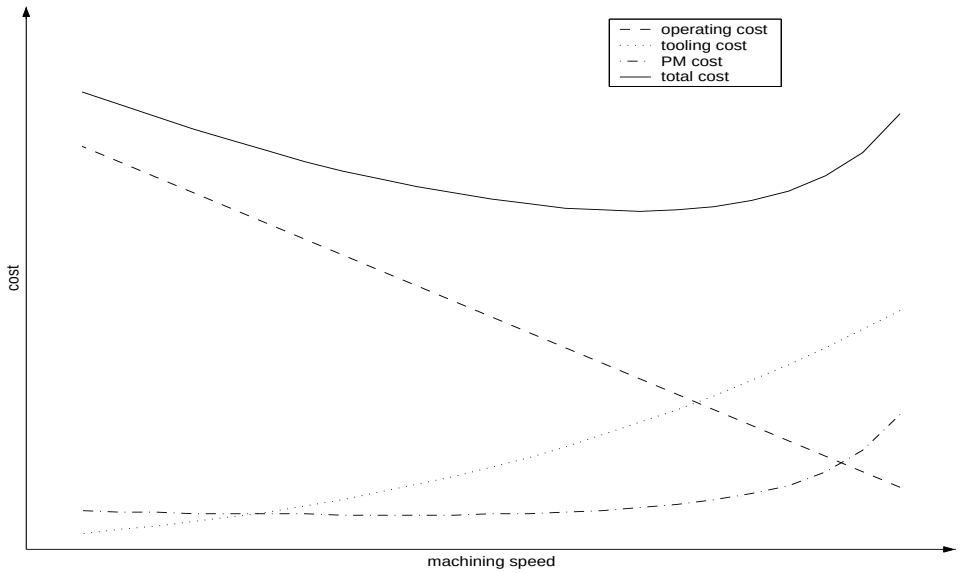

Figure 3.6: Costs for SMOP vs. Machining Speed along Surface Roughness Constraint

Let $f_{ij} = \left(\frac{1}{C'_s v_{ij}^g}\right)^{\frac{1}{h}}$, then the objective function becomes:

$$M_{ij} = C_1(C'_s)^{(-1/h)} v_{ij}^{\frac{g-h}{h}} + C_2(C'_s)^{\frac{-(\beta_j-1)}{h}} v_{ij}^{\frac{h(\alpha_j-1)-g(\beta_j-1)}{h}} + C_3(C'_s)^{\frac{-(k-1)}{h}} v_{ij}^{\frac{(h-g)(k-1)}{h}}.$$

Objective function above is convex but, we can not find an explicit closed form expression for optimal v_{ij} when we take a derivative of function and look for roots. So, this function will be solved for minimum by using Golden Section Algorithm (GSA) which is one of the most effective search algorithms for finding the optimum of one dimensional convex functions.

The GSA first takes a search interval on the domain of the function. Then by iteratively shortening this interval, it finds an interval of allowable length including optimal point. More detail can be found in Bazaraa et al. [11].

To apply GSA, we will first find an upper limit and a lower limit for optimal machining speed v_{opt}. The GSA will be applied on the interval between these two points. These limits are to be decided among three points, v_{mintm} which

minimizes processing time on the surface roughness constraint for the job, v_p which minimizes PM cost on the surface roughness constraint for the job and v_{m+t} which minimizes sum of machining and tooling costs on the surface roughness constraint. The algorithm will find those three candidate points. Then, it will decide the appropriate upper and lower limits to apply GSA. So, the proposed algorithm for solving SMOP is below:

Step 1. Find v, f where both surface roughness and machine power constraints are tight. By using equalities for both constraints, corresponding expressions for v and f are given below:
$v_{ij} = \left(\frac{1}{C'_s f_{ij}^h}\right)^{\frac{1}{g}}$ and $f_{ij} = [C'^g_m C'^{-b}_s]^{\frac{1}{bh-gc}}$
Assign $v1 = v_{ij}$.

Step 2. Find v, f where both surface roughness and tool life constraints are tight. $v_{ij} = \left(\frac{1}{C'_s f_{ij}^h}\right)^{\frac{1}{g}}$ and $f_{ij} = \left[C'^{\alpha-1}_s C'^{-g}_t\right]^{\frac{1}{g(\beta-1)-h(\alpha-1)}}$
Assign $v2 = v_{ij}$.

Step 3. Decide minimum processing time, v_{mintm} which is equal to $min(v1, v2)$.

Step 4. Relax tool life and machine power constraints and find v, f which minimize PM index on the surface roughness constraint. Processing time level which minimizes PM index is:
$tm' = \left(\frac{B(k-1)}{A}\right)^{(1/k)}$
As surface roughness constraint is tight:
$f_{ij} = \left(\frac{1}{C'_s v_{ij}^g}\right)^{\frac{1}{h}}$
Substituting into processing time expression,
$v_{ij} = \left[\left(\frac{B(k-1)}{A}\right)^{\frac{1}{k}} \left(\frac{12(C'_s)^{(-1/h)}}{\pi D_i L_i}\right)\right]^{\frac{h}{g-h}}$
Assign $v_p = v_{ij}$.

Step 5. Relax tool life and machine power constraints and find v, f which minimize machining plus tooling cost on the surface roughness constraint. The surface roughness constraint being tight we have:
$f_{ij} = \left(\frac{1}{C'_s v_{ij}^g}\right)^{\frac{1}{h}}$

By using expression for f, we can derive objective function on the surface roughness constraint and taking derivative for v, optimal v is as below:
$v_{ij} = \left[\frac{C_1}{C_2}(C_s')^{(\beta_j/h)}\frac{h-g}{h(\alpha_j-1)-g(\beta_j-1)}\right]^{\frac{h}{h\alpha_j-g\beta_j}}$
Assign $v_{m+t} = v_{ij}$.

Step 6. If $v_{mintm} \leq min(v_p, v_{m+t})$ then $v_{opt} = v_{mintm}$,(Case 1 and Case 2 in Figure 3.9), go to Step 8.
Else, go to Step 7.

Step 7. Find interval of machining speed (v_u: upper limit and v_l: lower limit) to apply GSA as below:
If $v_p < v_{m+t}$ then $v_l = v_p$ and $v_u = v_{m+t}$,(Case 4 and Case 6 in Figure 3.9),
else $v_l = v_{m+t}$ and $v_u = v_p$, (Case 3 and Case 5 in Figure 3.9).
If $v_l < v_{mintm} < v_u$ then $v_u = v_{mintm}$, (Case 5 and Case 6 in Figure 3.9).

Step 7.1. (Apply Golden Section Algorithm) Choose an allowable final length of uncertainty $l_v > 0$. Let $[v_l, v_u]$ be the initial interval of uncertainty, and let $\lambda_1 = v_l + (1-\eta)(v_u - v_l)$ and $\mu_1 = v_l + \eta(v_u - v_l)$, where $\eta = 0.618$. Evaluate $\Theta(\lambda_1)$ and $\Theta(\mu_1)$, let $k = 1$, and go to Step 7.2.1.

Step 7.2.1. If $v_u - v_l < l_v$, optimal solution lies in the interval $[v_l, v_u]$, $v_{opt} = (v_l + v_u)/2$ and go to Step 8. Otherwise, if $\Theta(\lambda_k) > \Theta(\mu_k)$, go to Step 7.2.2; and if $\Theta(\lambda_k) \leq \Theta(\mu_k)$, go to Step 7.2.3.

Step 7.2.2. Let $v_l = \lambda_k$. Furthermore, let $\lambda_{k+1} = \mu_k$, and let $\mu_{k+1} = v_l + \eta(v_u - v_l)$. Evaluate $\Theta(\mu_{k+1})$ and go to Step 7.2.4.

Step 7.2.3 Let $v_u = \mu_k$. Furthermore, let $\mu_{k+1} = \lambda_k$, and let $\lambda_{k+1} = v_l + (1-\eta)(v_u - v_l)$. Evaluate $\Theta(\lambda_{k+1})$ and go to Step 7.2.4.

Step 7.2.4. Replace k by $k+1$ and go to Step 7.2.1.

Step 8. Having found the optimal machining speed, corresponding feed rate on surface roughness constraint and corresponding t_m are calculated.

At the first three steps of the proposed algorithm above, we find v_{mintm}. This

is the machining speed that minimizes processing time on the surface roughness constraint. There are two cases that we should check. One is that we have v_{mintm} at the point where both surface roughness and machine power constraints are tight. The other case is that we have v_{mintm} at the point where surface roughness and tool life constraints are tight. Figure 3.7 presents the case where minimum processing time is at intersection of surface roughness and machine power constraints. Figure 3.8 presents the case where surface roughness and tool life constraints are tight. At Step 3, we decide v_{mintm}.

Here, it is necessary to mention that corresponding t_m to v_{mintm} is the minimum processing time that we can achieve on the entire feasible region. It is a lower bound on the processing time of a job.

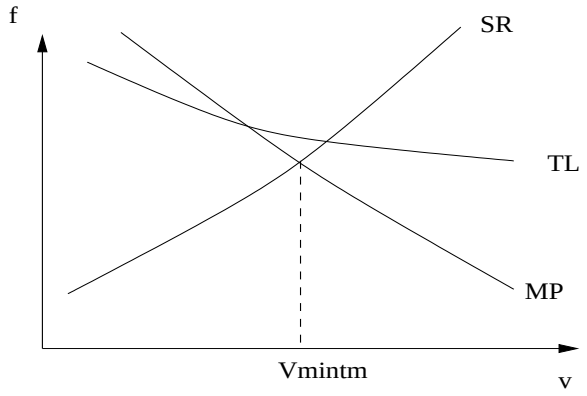

Figure 3.7: v_{mintm} at intersection of SR and MP

After deciding v_{mintm}, we find v_p and v_{m+t} by using closed form expressions at Step 4 and Step 5. At Step 6, we check if $v_{mintm} \leq min(v_p, v_{m+t})$. If so, then we do not need to apply any search because v_{mintm} is the optimal machining speed. This step covers Case 1 and Case 2 in Figure 3.9. If we can not decide v_{opt} at Step 6, we find an interval to run GSA at Step 7. At Step 7, Case 3 to Case 6 in Figure 3.9 are considered. Mainly, we expect to search for optimal v between v_p and v_{m+t}, because we are considering a convex function which is sum of machining, tooling and PM costs. However, we have v_{mintm} which comes from either the tool life constraint or the machine power constraint. v_{mintm} may limit the region that we consider for search. After deciding the interval to run GSA at

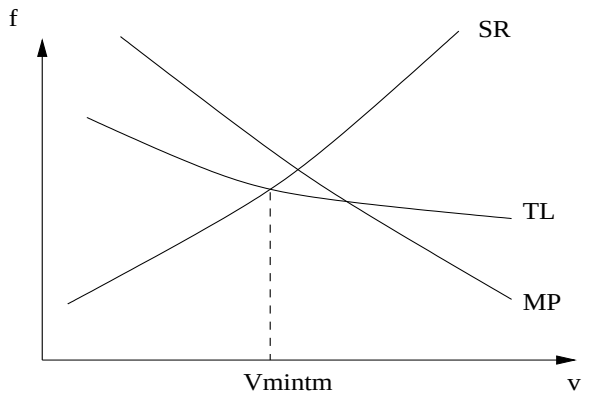

Figure 3.8: v_{mintm} at intersection of SR and TL

Step 7, through Step 7.1 to Step 7.2.4, we apply GSA to find v_{opt}.

At Step 8, we achieve the corresponding optimal f value for the calculated v_{opt} in previous steps, consequently the optimal t_m is calculated.

By the algorithm above, we have provided a solution method for SMOP. There are studies in the literature on SMOP. Gopalakrishnan and Al-Khayyal [17], Akturk and Avci [3] and Akturk et al. [5] are three examples which study SMOP with only machining and tooling cost terms. The last two studies provide optimality conditions and algorithms to solve SMOP with same constraints but including only machining and tooling cost in the objective function. In this study, we added the PM cost term into the objective function of the SMOP. So, number of terms in geometric program increased by 1. This increased the degree of difficulty of the geometric programming formulation by 1 to three. Therefore, the problem is harder than the one without PM cost term in the objective function. For this new problem we provided a solution method. So, we are now able to solve SMOP having PM cost in its objective function as well as machining and tooling costs.

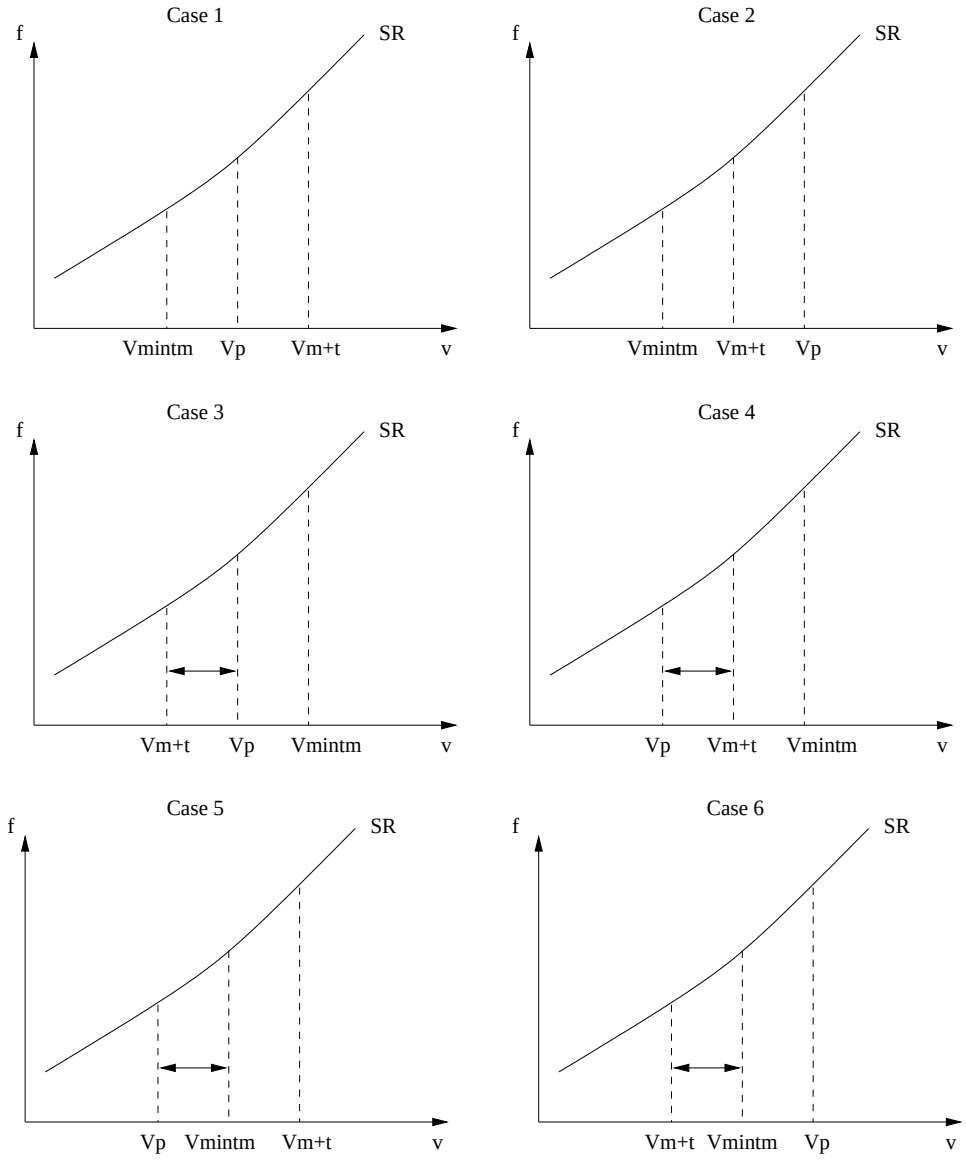

Figure 3.9: Possible intervals to run GSA

3.4 Conclusion

In this chapter we first discussed PM cost and machining conditions relationship. We provided mathematical expressions of PM cost for a job as a function of its processing time. We also defined PM index for a job. PM index of a job provides the PM need of a machine caused by operating a job.

Then we incorporate PM cost to the SMOP and design an algorithm for optimal solution. New SMOP is a more difficult problem than it was without PM cost term in it. The next step was to design an algorithm for solving SMOP with PM cost included in the objective function. The algorithm is based on the characteristics of the optimal solution and the behavior of the objective function.

Having discussed PM cost of a job and derived results for SMOP, the rest of the thesis is devoted to the scheduling problem that is introduced in the next chapter. The PM index concept that we derived in this chapter is the key for the following discussions. In the next chapter we will define the problem of minimizing total completion time of jobs on a single machine with availability constraint due to PM visits and with processing time variability of jobs. Besides the definition of the problem, some optimality properties of the problem will also be provided.

Chapter 4

Problem Definition and Modeling

4.1 Problem Definition

We are given N jobs of certain depth of cut, diameter and length. Each job has a maximum allowable surface roughness level. These jobs are to be processed on a single CNC turning machine using different cutting tools. We have machine power level and tool life constraints which restrict us while deciding machining speed and feed rate for each job as discussed in Chapter 3. Therefore, for each job we have a minimum processing time level, calculation of which is given in Chapter 3. We have a PM index function for the machine, which is a function of processing time. PM indices of the jobs are used to determine when to schedule PM visits. Between two PM visits, sum of the PM indices of the jobs can't exceed 1. This is the requirement which makes the minimizing total completion time problem difficult.

The problem is sequencing the jobs and deciding their processing times (equivalently finding the machining conditions) to minimize the total completion time. So, the problem has operation level constraints of machine power, tool life and surface roughness, and machine level constraint of PM.

4.2 Assumptions

We are trying to minimize total completion time on a CNC machine where PM visits are made when needed. We list the assumptions about the environment and operations we are dealing with:

- There is a single machine which is available except when a PM is applied on.
- There are N independent jobs of single operation all of which are available at time zero.
- Physical specifications for each job (i.e diameter, length, maximum allowable surface roughness, depth of cut) are fixed and known.
- The jobs are non resumable. Due to surface finish requirements, a cutting operation on a job is to be completed at once.
- There is a PM cost as a function of production rate, which was described in Chapter 3, for the machine. So, can calculate the PM index value for each job for the given processing time using this function.
- Sum of PM indices of jobs between two PM visits can not exceed 1.
- Each PM visit has a fixed duration and fixed cost.
- Maintenance team is always available whenever PM is needed. There is no PM resource limitation.
- Tool changes can be made offline. So, tool change time is not considered.
- A cutting operation on a job can be completed by at most one tool.
- Job loading and unloading times are negligible.

Given these assumptions we will try to decide machining conditions of jobs and schedule to minimize total completion time of the jobs. We will relate our PM strategy with machining conditions knowing that PM need is a function of operating conditions of the machine.

4.3 Modeling the Problem

In Chapter 3, we introduced the single machining operation problem. It was dealing with decision of machining conditions for a job under given constraints of surface roughness, machine power and tool life. There, we introduced PM cost function into the objective function of classical SMOP and provided a solution algorithm for the new form of the problem. So, we engaged the PM cost of a job into machining conditions decision process. Based on the results found in the previous chapter, we are able to find minimum processing time, minimum cost and minimum PM index settings for each job.

Here, in our original problem of scheduling, we have N jobs and we have to make machining conditions decisions for each job as well as sequencing them. As described in Chapter 3, PM index is a function of t_m, so, in this scheduling problem, rather than machining speed and feed rate, our decision variable is t_m, machining time. When t_m is decided, we can easily calculate v and f which give minimum total cost for a given t_m.

In our formulation, X_{ij} is a binary variable which denotes if job i is assigned to position j in a given sequence. t_{m_i} is the processing time of the job i. $P(t_{m_i})$ is the PM index of job i when its machining time is t_{m_i}. It is a nonlinear convex function of t_{m_i} and is described Chapter 3. d_j is the sum of PM indices of jobs after last PM visit up to position j. Y_j decides if a PM visit takes place before position j in sequence of jobs. $t_{m_i}^{min}$ is the minimum processing time for job i. It comes from the SMOP and it is decided by technical constraints and job specifications. T_{PM} is the PM visit duration for the system. It is assumed to be a constant predefined value.

So, a mathematical model for the problem is given below:

$$MIN \sum_{i=1}^{N}\sum_{j=1}^{N} (N-j+1)\, t_{m_i}\, X_{ij} \;+\; T_{PM} \sum_{j=1}^{N} (N-j+1) Y_j$$

ST

$$t_{m_i} \geq t_{m_i}^{min} \quad i = 1, .., N \tag{4.1}$$

$$\sum_{j=1}^{N} X_{ij} = 1 \quad i = 1, .., N \tag{4.2}$$

$$\sum_{i=1}^{N} X_{ij} = 1 \quad j = 1, .., N \tag{4.3}$$

$$\sum_{i=1}^{N} X_{ij} P(t_{m_i}) \leq d_j \quad j = 1, .., N \tag{4.4}$$

$$d_{j-1} + \sum_{i=1}^{N} X_{ij} P(t_{m_i}) \geq d_j \quad j = 1, .., N \tag{4.5}$$

$$d_j - \sum_{i=1}^{N} X_{ij} P(t_{m_i}) + Y_j > 0 \quad j = 2, .., N \tag{4.6}$$

$$d_j - \sum_{i=1}^{N} X_{ij} P(t_{m_i}) + Y_j \leq 1 \quad j = 2, .., N \tag{4.7}$$

$$d_j - d_{j-1} - \sum_{i=1}^{N} X_{ij} P(t_{m_i}) + Y_j \geq 0 \quad j = 2, .., N \tag{4.8}$$

$$d_j - d_{j-1} - \sum_{i=1}^{N} X_{ij} P(t_{m_i}) + Y_j < 1 \quad j = 2, .., N \tag{4.9}$$

$$d_j \leq 1 \quad j = 1, .., N \tag{4.10}$$

$$d_0 = 0 \tag{4.11}$$

$$X_{ij} \in \{0, 1\} \quad i = 1, .., N, \; j = 1, .., N$$

$$Y_j \in \{0, 1\} \quad j = 1, .., N$$

The objective function is mainly composed of two parts. The first part is given below:

$$\sum_{i=1}^{N} \sum_{j=1}^{N} (N - j + 1) \, t_{m_i} \, X_{ij}$$

which calculates the total effect of processing time of all jobs to the completion time of themselves and to the completion time of the following jobs. So, if we increase processing time of a job then this part is increased since completion time of this job and all following jobs are increased.

The second part of the objective function is given below:

$$T_{PM} \sum_{j=1}^{N} (N - j + 1)Y_j$$

which calculates the total effect of PM visits to the completion times of all jobs following them. So, if we insert a PM visit in somewhere in schedule then completion times of all jobs following the inserted PM visit are shifted backward and their completion times are increased.

Constraint set (4.1) keeps t_{m_i} higher or equal to the minimum processing time achievable for job i. This constraint comes from the SMOP where we decide minimum processing time machine settings. Constraint sets (4.2) and (4.3) guarantee that each job is assigned to a position and no two jobs can be assigned to the same position, respectively. Constraint sets (4.4) to (4.11) are the PM constraint sets, which guarantees that between two consecutive PM visits sum of PM indices of jobs does not exceed 1 and PM is to be scheduled where needed.

The model determines both the positions of jobs and their processing times. When the processing time of a job is found after solving this problem, we can easily calculate machining speed and feed rate for the job. We showed in Chapter 3 that, in SMOP minimum cost is achieved on the surface roughness constraint for a given processing time level. Then, by solving surface roughness equality and processing time equality together we achieve machining speed and feed rate values that give minimum cost for each job for given processing time that minimizes total completion time.

To sum up, what we want to do is to decide which job is to be processed when and how long the machine will be assigned to each job and when we will visit the machine for PM so that we minimize total completion times of jobs. If we define set of jobs between two PM visits as a 'block', the problem is forming blocks of jobs and scheduling PM's between blocks so that the total completion time of jobs will be minimized.

The total completion time problem with PM with fixed processing times is proved to be NP-hard. Considering variable processing times and nonlinear PM

index function, we are dealing with a more difficult problem in this study. However, we provide some information about the properties of the optimal solution for the problem. In the next section we will discuss those properties.

4.4 Structural Properties of Optimal Solution to the Scheduling Problem

We have some necessary conditions that an optimal solution must satisfy.

Property 1 (SPT within a PM interval): In the Shortest Processing Time (SPT) rule, we position the jobs in sequence in ascending order of their processing times so that, say positions of two jobs are i and j, if $i < j$ then $t_{m_i} \leq t_{m_j}$. The jobs to be operated between two PM visits must be sequenced in the SPT order. This can easily be proved by contradiction such that pairwise interchanging positions of two jobs which are not in the SPT order improves total completion time of a schedule. Then, any schedule which is not in the SPT order inside blocks can't be optimal. This property is common for the problems in Qi et al. [43] and Akturk et al. [4].

In our problem, we have controllable processing times. Furthermore, PM index which is the PM need of a machine caused by doing that job is a function of its processing time. So, we are able to change PM index of a job by changing its processing time. Having the processing time variability, we can prevent wastes at the end of PM intervals by adjusting the processing times of jobs. This is possible if we have jobs that are not currently set to their minimum processing times. Then, we can define a property for an optimal solution based on utilization of PM interval as follows;

Property 2 (PM Interval Utilization or Block Utilization): In an optimal solution, consider a block j, either $t_{m_i} = t_{m_i}^{min}$ for $i \in j$ or $\Sigma_{i \in j} P_i = 1$.

The condition given above means that in an optimal schedule for given problem, for a block; either all jobs are set to their minimum processing time levels,

which means there is no way to decrease their processing times further, or total PM index of the block is equal to 1, which means the block is fully utilized in terms of PM. There are similar properties in Qi et al. [43] and Akturk et al. [4] to our Property 2.

Property 3 (Average Job Time): $(T_i + T_{pm})/n_i \leq (T_j + T_{pm})/n_j$ for any two PM intervals i, j such that $i < j$. T_i and T_j are total processing times for blocks i and j whereas n_i and n_j are number of jobs in blocks i and j. This can also be proved by pairwise interchanging two blocks which are not in ascending order of average job times. Pairwise interchanging such blocks improve total completion time objective. This property is also common for the problems in Qi et al. [43] and Akturk et al. [4].

Property 4 (Limits for Processing Time of a Job); For optimality, each job must satisfy; $t_{m_i}^{min} \leq t_{m_i} \leq max(t_{m_i}^{min}, t_m^{minPM})$ where $t_{m_i^{min}}$ is the minimum processing time for job i and t_m^{minPM} is the processing time level that gives the minimum PM index which is given in Chapter 3. The left inequality above is a feasibility condition, it assures that processing time of a job is higher than the minimum processing time for that job. The right inequality is an optimality condition. As t_{m_i} increases beyond t_m^{minPM} P_i also increases. In such a case, total completion time can not improve. So, we do not allow $t_{m_i} > t_m^{minPM}$ unless $t_{m_i}^{min} > t_m^{minPM}$.

In the literature, for total completion problem with maintenance or tool change constraint on a single machine, some properties of optimal solutions were shown by Qi et al. [43] and Akturk et al. [4]. Those problems were assuming that maintenance or tool change will take place on a machine without exceeding a predefined operation time T. So, they were different than our problem in the sense that they assume that a job contributes to the PM need of the machine in proportion with its processing time whereas we assume that a job contributes to PM need by a convex function of processing time where smaller processing time values may lead to higher maintenance need.

Qi et al. [43] and Akturk et al. [4] introduced some optimality properties for their problems. As mentioned before, some of our properties are same or similar

with some properties they derived for their problems. However, they have an optimality property which does not apply to our problem. This property is as follows: $n_i \geq n_j$ for any PM blocks i, j such that $i < j$. It is not applicable to our problem, since PM index of a job in our problem may be higher for a job with smaller processing time. In our case, the contribution of a job to PM is not directly proportional with its processing time. A counterexample for our problem can be generated which violates this property. Suppose we have 5 jobs assigned to 2 blocks. Block 1 is formed by 2 jobs of $t_m = 1$ and $P_i = 0.5$ and Block 2 is formed by 3 jobs of $t_m = 2$ and $P_i = 1/3$. So, $n_1 < n_2$ and total completion for these 2 blocks is 30. If we interchange two blocks, we obtain total completion of 33 with first block in sequence having more jobs than the other one. So, property of $n_i \geq n_j$ for any PM blocks i, j such that $i < j$, may not be true for an optimal solution of our problem. This concludes that the minimizing total completion problem we are considering is quite different than the problems in the literature.

4.5 Summary

In this chapter, we described and modeled the scheduling problem. We presented the assumptions for the problem. We provided a mathematical model for the problem. Finally, a set of optimality properties is given. In the next chapter we will propose a heuristic algorithm for the problem. The optimality properties discussed here will be important to understand the algorithms which will be presented in the next chapter.

Chapter 5

Proposed Heuristic Algorithms

In Chapter 4, we defined our problem and introduced characteristics of the problem. In this chapter, solution methods for the problem will be presented. Our solution approach, simply stated, will be based on generating blocks of jobs and solving a subproblem that decides sequence and t_m's of jobs for a block.

In this chapter, we will discuss our search algorithm. This search algorithm includes two main stages: branching process and improvement process. The branching process has a logic to enumerate alternative solutions and includes a subproblem solving stage which solves a subproblem that will be defined below. The improvement process is designed to improve the solution that we achieve by the branching process. For the subproblem we designed three different heuristics. Using different heuristics to solve the subproblem we will present five different versions for the search algorithm.

Minimizing total completion time on a single machine with PM constraint is a problem that we have to decide the sequence of jobs as well as their processing times. We need to consider PM visits to the machine while making decisions. We do not have previously planned PM visits. We will decide when to make PM visits to the machine. Our approach is handling the problem with a different point of view. In our problem all processing times are controllable. Each job has a minimum processing time limit due to technical capabilities of the system and job

requirements. This flexibility of processing time decision leads us to think that, in this problem, rather than deciding where each job should stand in sequence we would focus on the questions 'how many PM blocks we will have' and 'how many jobs each block will have'. Without excluding the minimum processing time limitations we develop a search procedure which is mainly looking for the answer of question that we derived from our original problem.

To understand the search algorithm better, firstly, the subproblem, that we will need to solve many times in the search algorithm will be introduced and three alternative methods to solve this subproblem will be provided.

5.1 Subproblem: Minimizing Processing Time Effect of a Given Block to Total Completion Time

Minimizing total completion time problem on a single machine with availability constraint and with constant processing times is proven to be NP-hard. In our problem, we have processing time variability and non-linear PM index function. So, we are still working on a difficult problem. In this section a subproblem of our original problem will be discussed. Of course this subproblem is one that we can cope with.

Suppose that a block of jobs is given to us. We know which jobs are in this block. We also know where this block is positioned in the entire schedule. Since each job in this block has a processing time effect on total completion time value of schedule, we have the following question to be answered: "How should we order jobs in this block and what should be their processing times so that total processing time effect of this block to total completion time of entire schedule is minimized?". In our search algorithm, we will need to find an answer to this question many times, and the methods that we will propose for this problem will be basis to form different versions of search algorithm for the problem.

Our subproblem is, given a block of certain jobs in a schedule to find the schedule inside the block so that total processing time effect of jobs in this block to total completion time of the entire schedule is minimized. We want to decide sequence inside the block as well as processing time of each job in the block. The first heuristic that we will describe for this subproblem is the LP-based method (LPB).

5.1.1 LP-based Method (LPB)

LPB method is based on the idea of relaxing the combinatorial part of the subproblem and assuming that sequence of jobs in the block is known. Relaxing combinatorial part of the problem led us to a problem which is a linear program that we can easily solve. Below, we will discuss the idea in detail.

5.1.1.1 A Separable Programming Model

In this part, we will derive a mathematical model which minimizes processing time effect of a block for a given sequence of jobs inside the block. Then we will discuss how it is approximated by an LP model.

Suppose we have K jobs in a block and these jobs are sequenced in an ascending order of their minimum processing time levels. Say, the first job in the block is placed on position k of the entire schedule. So, the second job is on position $k+1$ and so on. Then the mathematical model for the problem is as below:

$$\min \sum_{j=k}^{K+k-1} (N-j+1)\, t_{m_j} \tag{5.1}$$

Subject to

$$t_{m_j} \geq t_{m_j}^{min} \quad j = k, .., (K+k-1) \tag{5.2}$$

$$\sum_{j=k}^{K+k-1} P(t_{m_j}) \leq 1 \tag{5.3}$$

Here t_{m_j} is the processing time of the job in position j. $t_{m_j}^{min}$ is the minimum processing time that we can assign to job in position j. This value is known from SMOP calculations and decided by technical constraints and job part's size. $P(t_{m_j})$ is the nonlinear PM index function of t_{m_j}.

The objective function here sums the processing time effect of each job in the block. The constraint set (5.2) applies the lower bound for each each job's processing time. The constraint (5.3) assures that total PM index of jobs in current block is less than or equal to 1, so that resulting schedule is feasible in terms of PM.

The nonlinear mathematical model given above is a Separable Programming model because each nonlinear term in constraint (5.3) is formed of a single variable. If all non-linear terms are convex, Separable Programming problems can be solved approximately by applying Simplex Method to piecewise linear approximation of the model as discussed in Bazaraa et al. [11].

In our model, the only nonlinear term is the PM index function which is convex and it is only included in the second constraint of PM index limit. PM index function is same for all jobs. So, we will generate breakpoints for PM index function and use them to form piecewise linear approximation model.

5.1.1.2 Generating Breakpoints for Piecewise Linear Approximation of PM Function

While designing a piecewise linear approximation function, the aim is to guarantee that, at any point in the domain of the original function we have at worst a predefined acceptable difference between the value of piecewise linear approximation function and the value of original function. To do this, a measure for the difference between two functions will be used which is called a relative error. Calling piecewise linear approximation of $P(t_{m_i})$ as $P'(t_{m_i})$, relative error is defined as below:

$$\text{Relative Error} = \frac{(P'(t_{m_i}) \; - \; P(t_{m_i}))}{P(t_{m_i})}$$

The maximum allowable relative error, Δ', is to be a value like 1%, 5%, etc. The algorithm to find breakpoints is as below:

Step 1. Define the region where the approximation will be applied. Lower bound is the minimum of minimum processing times of jobs. Upper bound is the processing time where the PM index function is minimum. This property is due to Property 4 of Chapter 4.

Step 1.1. Add defined interval to list of intervals to be processed. Add lower and upper bound t_m's to the list of breakpoints.

Step 2. Choose the first available interval $[t_{m_{lb}}, t_{m_{ub}}]$ from the list of intervals. Delete the chosen interval from interval list.

Step 2.1. Find t'_m which gives the maximum relative error, Δ, in $[t_{m_{lb}}, t_{m_{ub}}]$.

Step 2.2. If $\Delta > \Delta'$ then add t'_m to the breakpoint list and add $[t_{m_{lb}}, t'_m]$ and $[t'_m, t_{m_{ub}}]$ intervals to the interval list. Else, go to Step 3.

Step 3. If interval list is not empty go to Step.2; else sort breakpoints in the breakpoint list in ascending order and stop.

Using the algorithm above we generate breakpoints for the PM index function so that relative error for the piecewise linear approximation using these breakpoints is not higher than Δ' at any point where approximation is defined.

The algorithm above first defines upper and lower limit for the piecewise linear approximation function. This lower and upper limit forms an interval and they are recorded as breakpoints. Then, it finds the point where relative error is maximum in this interval. If relative error is higher than we can accept then it records the point found as a breakpoint. The algorithm continues looking for breakpoints in all intervals formed by pairs of breakpoints which are neighbors. Having found the breakpoints the next step is to form the model by using our piecewise linear function.

5.1.1.3 Piecewise Linear Approximation Model

After explaining the way to generate breakpoints for the nonlinear function of PM index, we can now give the mathematical model for this approximation. The processing time of a job t_{m_j} in the previous model will be transformed to an expression which is linear combination of its minimum processing time and the breakpoints higher than its minimum processing time, such that

$t_{m_j} = \lambda_{j1}\mu_{j1} + ... + \lambda_{jB_j}\mu_{jB_j}$

where μ_{jl}'s are breakpoints and λ_{jl}'s are coefficients such that $0 \leq \lambda_{jl} \leq 1$ for all l's. B_j is the number of breakpoints for job j. Here, μ_{j1} is the minimum processing time for job j and μ_{jB_j} is the processing time that gives minimum PM index. So the model is as below:

$$\min \sum_{j=k}^{K+k-1} (N-j+1)\,(\lambda_{j1}\mu_{j1} + ... + \lambda_{jB_j}\mu_{jB_j}) \tag{5.4}$$

subject to

$$\sum_{j=k}^{K+k-1} \lambda_{j1}P(\mu_{j1}) + ... + \lambda_{jB_j}P(\mu_{jB_j}) \leq 1 \tag{5.5}$$

$$\lambda_{j1} + ... + \lambda_{jB_j} = 1 \quad j = k, .., K+k-1 \tag{5.6}$$

$$\lambda_{jl} \geq 0 \quad j = k, .., K+k-1, \quad l = 1, .., B_j \tag{5.7}$$

Constraint (5.5) is the PM index limit constraint for the block that sum of PM indices of jobs in a block can't exceed 1. Constraint sets (5.6) and (5.7) provide that the model chooses points which are convex combinations of breakpoints or PM index values at breakpoints. So, using piecewise linear approximation and simplex method we can solve our subproblem with nonlinear constraints and with the condition that job sequence is known. Based on the findings above, an LP-based algorithm is given below.

5.1.1.4 Algorithm for LP-based Method (LPB)

Algorithm given below works on the given block. The jobs in the given block first ordered by the SPT rule applied for minimum processing times of jobs. Then, it solves piecewise linear approximation for the formed sequence inside block. Finally, if resulting schedule after solving LP is not in the SPT order, jobs in the block are reordered by the SPT rule according to Property 1 discussed in Chapter 4.

Step 1. Sequence jobs in the given block in ascending order of their minimum processing times.

Step 2. Solve piecewise linear approximation model for the block for the sequence formed in Step 1.

Step 2.1 If the LP in Step 2 is infeasible, Stop, the block is infeasible. Else, go to Step 3.

Step 3. If the schedule after solving LP is not in the SPT order, then reorder jobs by the SPT rule.

In Step 1, we order jobs in ascending order of their minimum processing times. This is a logical approach based on the fact that for two jobs in sequence we will try to set former of the two jobs in a block to smaller t_m than the latter one. So, we need the one with smaller minimum processing time to be positioned before the other.

The last step of this method is rearranging jobs by the SPT rule. This is because, we are solving piecewise linear approximation and solution of the LP may not be in the SPT order. Due to optimality Property 1 in Chapter 4, we need to satisfy the SPT rule inside a block. This is why we have a reordering process at the end. We also include a step that handles the case if the block is infeasible, which means the jobs do not fit to the block in any case.

We completed the discussion for LPB method. It is based on solving LP for the piecewise linear approximation model of the problem which assumes that sequence of jobs is known. The next method that we will cover for our subproblem is the First Update Shortest Method (FUS).

5.1.2 First Update Shortest Method (FUS)

Another heuristic approach which can be used for our subproblem is the FUS method. This method is based on the aim to keep the makespan of block at its minimum. If we set all jobs in a block at their minimum processing times and the total PM index of block is above 1, then we have to change t_m's of some jobs in block. In this method, we take the job with minimum processing time and increase its processing time to next t_m breakpoint available for the job. We apply this until total PM index of block is less than or equal to 1. Note that, breakpoints that we use in this method are the ones that we generated for the LPB method. Since we always select the shortest job, this method is trying to keep makespan of block as small as possible. Algorithm for this method is below:
Step 1. Set all jobs in the block to their minimum processing times, sequence them by the SPT rule.
Step 2. If total PM index of jobs in block is less than or equal to 1, go to Step 2.1. Else, Stop.
Step 2.1. Take the first job in sequence. If its processing time is less than the processing time which gives minimum PM index, increase its processing time to next breakpoint. Else, Stop, the block is infeasible.
Step 2.2. Update total PM index value for the block.
Step 2.3. If the SPT rule is not obeyed, sequence jobs by the SPT rule.
Step 2.4. If total PM index value for block is greater than 1; Go to Step 2.1. Else, Stop.

The FUS method described above, simply increases the processing time of shortest job in the block if the total PM index of block is above 1. Then, it reorders jobs in the block by the SPT rule. It evaluates total PM index of the block and repeats the process if it is higher than 1. Step 2.1 above considers the case if the jobs in the block do not fit to the block.

You can observe that FUS method is dealing with the makespan of the block rather than processing time effect of each individual job. It considers the first job and the last job as having same effect on total completion time of the entire schedule and it tries to set t_m's of the jobs in the block in balance. It is likely to

get a block at the end where there is small difference between first and last jobs of the block. The nice point for FUS is that it is easier and faster to implement it. The next heuristic we will cover is Total Completion Index (TCI) method.

5.1.3 Total Completion Index Method (TCI)

In the previous heuristic we tried to minimize the makespan of the block. In this method we will consider the processing time effect of jobs to total completion time by using an index which measures change in processing time effect of a job per change in its PM index if we set it to the next breakpoint of t_m. In this method also, we use the breakpoints that we generate for LPB method. So, total completion index of a job i at position k is;

$TCI_i = \frac{(N-k+1)(t'_{m_i} - t_{m_i})}{(P_i - P'_i)}$

where t_{m_i} is the current processing time and t'_{m_i} is the next t_m breakpoint for the job and P_i and P'_i are corresponding PM indices for t_{m_i} and $t_{m_i}{}'$, respectively.

Step 1. Set all jobs in the block to their minimum processing time values, sequence them by the SPT rule.

Step 2. If total PM index of jobs exceeds 1;

Step 2.1. Find the job, with minimum TCI, whose processing time is less than the processing time level which gives minimum PM index in the block. If no such a job exists, stop, the block is infeasible. Else, increase the selected job's processing time to the next breakpoint and go to Step 2.2.

Step 2.2. Update total PM index value for the block.

Step 2.3. If the SPT rule is not obeyed, sequence jobs by the SPT rule.

Step 2.4. Update TCI's for all jobs in the block.

Step 2.5. If total PM index of jobs in the block is greater than 1; Go to Step 2.1. Else, stop.

The TCI method looks for the job which will have minimum processing time effect change per PM index change we will get when increased its processing time to the next breakpoint. So, it considers the positions of jobs when deciding which job is to be updated. At each iteration it reorders jobs by the SPT rule

and it recalculates the TCI indices of the jobs after reordering them. We also have a feasibility check to decide if jobs fit to the given block. The three methods provided for the subproblem will be applied in the search algorithms which we define below.

5.2 Search Algorithm

Having defined the subproblem that we will need to solve in our search algorithm and proposed heuristics for them we will define the search algorithm in this section. Our search algorithm will mainly generate different schedules, i.e. different blocks, and solve the subproblem for the blocks. After searching for different schedules and finding the schedule with minimum total completion time, we will apply some additional improvement procedures to improve the total completion time of the schedule.

As stated in Chapter 4, objective function of total completion time problem is composed of two groups of terms: one of which is the processing time effect and the other is the PM effect:

$$\sum_{i=1}^{N}\sum_{j=1}^{N}(N-j+1)\,t_{m_i}\,X_{ij} \;+\; T_{PM}\;\sum_{j=1}^{N}(N-j+1)Y_j$$

It is obvious that we can minimize the first part, processing time effect, by setting all jobs to their minimum processing times and ordering the jobs by the SPT rule. If no PM visit is to be made to machine, we could easily solve the problem by this way. However setting all jobs to their minimum processing times require too many PM visits which drastically increases total completion time especially when T_{PM} is high.

There is a trade off between two parts of objective function. When we set all jobs to their minimum processing times, PM indices of jobs are at maximum levels. We can decrease number of PM visits by increasing processing times of jobs. So, by paying for processing time effect part of objective function we may

gain from the PM effect part. The search algorithm that will be presented is based on the trade off mentioned above.

In the following subsection we will discuss in detail how we generate new schedules by branching process. Then, in the next subsection we will cover the improvement procedures that we will apply after branching process.

5.2.1 Branching Process

Our search procedure will be based on branching. Our initial schedule is the one with all jobs set to their minimum processing times, ordered by the SPT rule and PM visits inserted where needed. We will consider this initial schedule as the parent node (parent schedule). From this initial schedule by shifting different sets of jobs forward to first block we will generate new schedules, equivalently saying new nodes out of parent node. After solving the block subproblems for first blocks of these child schedules we will calculate total completion times for each of them. Among these nodes, which are children of the initial schedule, we will select new parent nodes. These parent nodes will be used to generate new schedules by shifting jobs forward to second blocks. So, at each stage we will form child schedules by shifting jobs to one block and then select parent schedules out of these child schedules. What is important in this structure is that at the first stage we crowd the first block of initial schedule, at the second stage we crowd the second block of parent schedules and so on. At a stage, for all parents, same indexed block is to be crowded and at each stage different indexed block is considered.

Based on the method defined above, we generate child schedules from parent schedules and choose new parents out of children. We repeat this process as long as we can find new parents.

The above logic resembles a beam search method roughly, but in our method each node represents a complete schedule whereas in beam search applications in scheduling, each node represents a partial schedule. Examples of beam search on

scheduling problems can be found in Ow and Morton [38] and Akturk and Kilic [6].

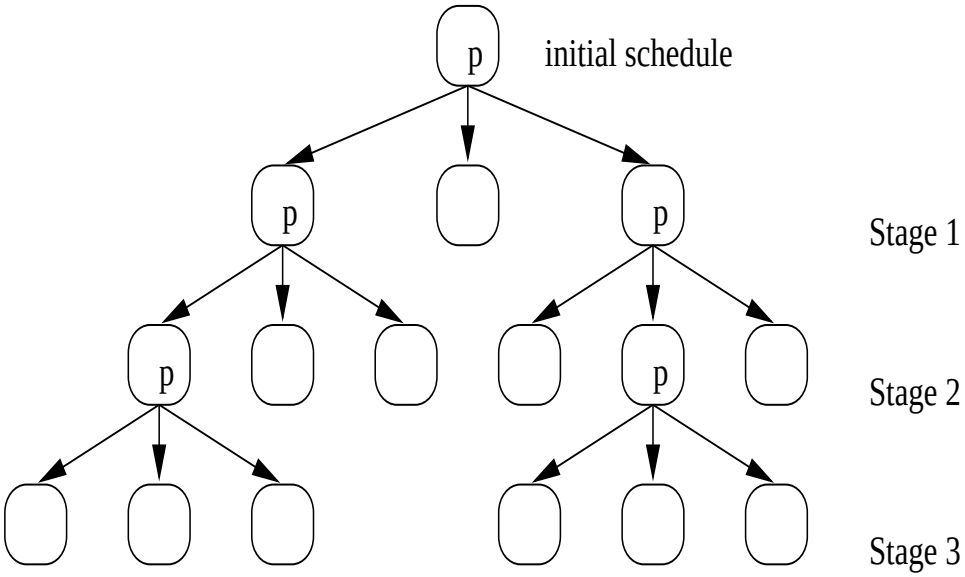

Figure 5.1: Structure of Branching Process Applied

Figure 5.1 shows how branching and filtering process works in our method. Each node in this figure represents a schedule which is generated from the parent. Arrows represent the parent-child relationship between schedules. The nodes with 'p' are the parent nodes selected for that level. For instance in Figure 5.1, 3 child schedules are generated from each parent and at each level 2 parents are selected. In our search algorithm we have two parameters, a number of child schedules to be generated from a parent, C, and a number of parent schedules to be selected among all child schedules at a stage, P.

When generating new schedules from a parent schedule, we shift jobs forward to a block. For each child of a parent we shift different number of jobs. We decide the number of jobs to be shifted by using a predefined PM index value α.

Suppose that from a parent schedule we will generate child nodes by shifting jobs to block i. For the first child we shift jobs having total PM index of α to block i. For the second child we shift jobs having total PM index of 2α to block

i, and so on using $3\alpha, ..., C\alpha$ where C is number of children to be generated from a parent. So, we are generating different schedules at each child.

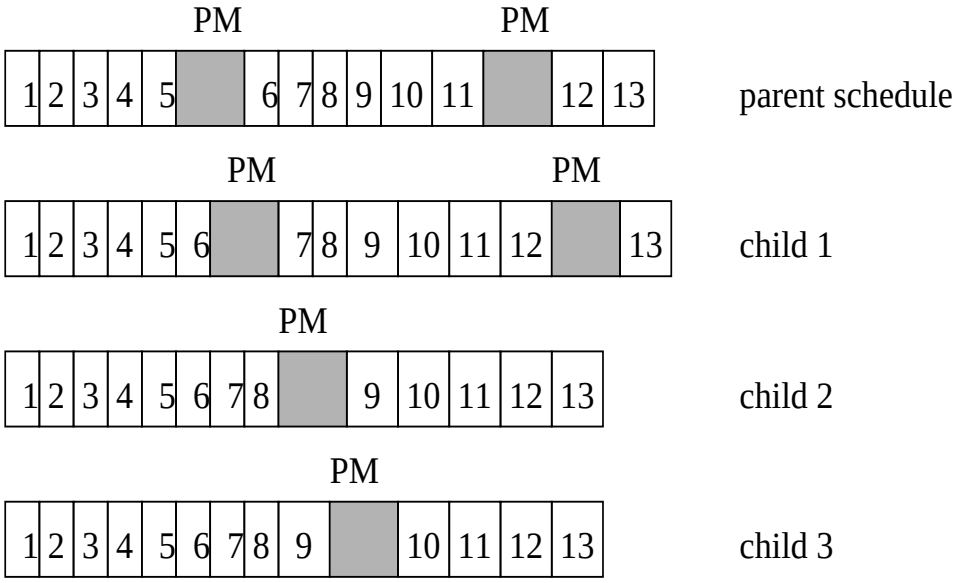

Figure 5.2: Job Shifting Process

Figure 5.2 simply shows how child schedules are generated from a parent schedule. Each box with an index on it represents a job and shaded boxes are PM visits during which the CNC machine is not available. Each index on a job represents the job number. This figure is just for representing block compositions rather than processing times or PM indices.

As can be seen in Figure 5.2, child schedules are formed by shifting jobs to first block of parent schedule. As we shift jobs from second block to the first block some jobs are also shifted from third block to second block. So, in fact, when we are crowding a block, all jobs beyond that block is shifted forward.

When we shift jobs to a block, sum of PM indices of jobs in that block exceeds 1. So, block is infeasible. To make it feasible, we need to change processing times of jobs in the block. While doing this we try to minimize the processing time effect of jobs in the block. To do this, we apply 3 different heuristics (LPB, FUS, TCI) which were explained before. Those heuristics will either find feasible schedules

for the block or decide that such a block is infeasible. A block is infeasible, if it is not possible to decrease total PM index of block to 1 after shifting jobs to the block. So, it is the case when jobs do not fit to the block. In such a case we omit such infeasible schedules.

Having generated child nodes from all parent nodes at a stage, we have a population of schedules. We calculate total completion time for each schedule. Then, we choose a predefined number P of them with minimum total completion times as parent nodes for the next stage. Selected schedules as parents will be used to generate population in the next stage. Therefore, parent schedules must have jobs that we can shift in the next stage, otherwise that schedule can not be parent even if it has minimum total completion time.

If we reach a stage where no nodes can be parents for the next stage, i.e. in all child schedules, no jobs to be shifted in the next stage can be found in the schedule, then search ends. This means we have no more blocks to crowd. The schedule with the best total completion time found up to then is the output of the branching process applied.

To sum up, in branching process we generate different schedules starting from the schedule which minimizes processing time effect in total completion time. By moving jobs forward we find new schedules and solve our subproblem for new formed blocks with total PM index higher than 1. At each stage we crowd a block and produce child schedules. New parents are selected out of these child schedules and passed to next stage. We apply this until no block to crowd in the next stage exists in schedule population. Having discussed the branching structure we will introduce the fine tuning approach in our branching process.

5.2.1.1 Fine Tuning in Branching Process

Besides the branching mechanism defined above we employed an improvement method denoted as fine tuning in the branching process. It is applied at each stage for each parent. As explained before we generate each child schedule by shifting a set of jobs to a block. Number of jobs to be shifted is decided by the

PM index value of $k\alpha$, if we are generating the child k of that parent. In fine tuning, for each parent, we select the child of minimum total completion time and generate schedules by using neighbor PM index values like $(k + 1/2)\alpha$ and $(k - 1/2)\alpha$. This is done, because when we analyze the total completion time values of child schedules of a parent, we realize that as k increases in $k\alpha$, resulting total completion time values behave as they are a convex function of k. Although we know that such a function is not convex, the characteristic of the problem promises for better schedules around best schedule. So, we apply this procedure in our algorithm. Each child schedule generated by this way is treated like a child schedule we defined before. Block subproblems are solved for these child schedules and they are considered when selecting parent schedules.

Above we discussed branching process and fine tuning approach applied in branching process. So, we completed the discussion of branching part of our search algorithm. At the end of the branching process, we output the schedule we found with minimum total completion time as the best schedule achieved so far. Our search algorithm includes two improvement procedures to be applied to this best schedule achieved so far. Below improvement steps will be covered.

5.2.2 Improvement Process

We have two improvement steps that we apply after branching process. The first one is the Last Block Improvement procedure and the other one is Block Rearrangement procedure. We first discuss Last Block Improvement procedure.

5.2.2.1 Last Block Improvement

One of the improvement tools to be applied to the schedule found by branching process is the last block improvement. At the end of our branching process we are sometimes left with a schedule whose last block is not fully utilized, equivalently having total PM index less than 1. As a result of branching process, the last block is composed of the jobs having the highest minimum processing time values.

So, jobs in preceding blocks can have greater processing times than the jobs in last block whereas they have smaller minimum processing times. So, we can interchange those jobs to include jobs with smaller minimum processing times in the last block. By this way, we can improve total completion time of the schedule. We keep processing times in preceding blocks same while decreasing the processing times of jobs in the last block. To apply this tool a test is made such that if total PM index of last block is less than $1 - \gamma$, we apply it. γ is a parameter that user defines. So, if a schedule's last block's total PM index is less than $1 - \gamma$, we look for the opportunities to bring the jobs with smaller minimum processing times to last block. This improvement tool is motivated by the Property 2 (Block Utilization) we introduced in Chapter 4. The next improvement step is the Block Rearrangement procedure which will be discussed below.

5.2.2.2 Block Rearrangement

The last possible improvement tool we implemented comes from one of the optimality properties that we listed in Chapter 4. After the last block improvement procedure applied to the schedule resulted from the branching process, we check if the blocks of this schedule is ordered in ascending order of average processing times which is defined in Property 3, in Chapter 4. If the schedule does not obey this property, we rearrange the blocks in ascending order of average processing times. This is what we mainly do in block rearrangement procedure.

In some versions of our search algorithm, as will be discussed later, we are applying additional steps to the block rearrangement procedure above. In additional steps, after block rearrangement, we apply LPB to all blocks of the schedule. So, we update processing times of jobs as their positions have changed due to block rearrangement. Then, we evaluate average processing times of blocks. If blocks are not still sequenced in ascending order of their average processing times, then we again rearrange blocks and apply LPB. We repeat this process, evaluate average processing times of blocks- rearrange blocks-apply LPB, until no rearrangement is necessary.

Up to now, we have described the branching process and improvement steps which are main parts of our search algorithm. We discussed the basic ideas that led us to design the logic at each part. We discussed how each procedure is applied. However, our discussions were covering the ideas rather than putting things in a complete logical format. Below, we will give algorithms of those methods discussed above in a logical sequence. We will start with the search algorithm using LPB method to solve subproblems. It will cover each step we include in our search algorithm. Then, we will give other versions of search algorithm, but this time we will just refer to the algorithm using LPB rather than repeating same steps for all versions.

5.2.3 Search Algorithm using LPB Method

In this version of our search algorithm we use LPB method to solve block subproblems. Also, in block rearrangement procedure we apply block rearrangement and LPB method to blocks in a repeating manner. Below, we include steps for the search algorithm using LPB method.

Step 1. Set all jobs to their minimum processing time levels and sequence them in the SPT order. Schedule PM's by using PM index values.

Step 2. Set PM index value α which is used to decide the number of jobs to be moved when generating child schedules. If $T_{PM} > 45$ then $\alpha = 0.4$, else $\alpha = 0.6$ (as we used in our experiments).

Step 3.1. (Branching Process) Set initial schedule as starting parent schedule for stage $i = 1$. Set it as best schedule achieved so far.

Step 3.2. For each parent do

Step 3.2.1. Generate m child schedules by moving jobs of total PM index $\alpha, 2\alpha, ..., m\alpha$ to block i.

Step 3.2.2. For, each formed child, apply the LPB method to block i.

Step 3.2.3. Calculate the total completion time for each child schedule, if total completion time of a child is less than total completion time of best schedule achieved so far, then set it as the best schedule achieved so far.

Step 3.2.4. Select the child schedule with minimum total completion time. Say,

child k.

Step 3.2.5. Generate two new child schedules from the parent schedule by shifting jobs of total PM index $(k + 1/2)\alpha$ and $(k - 1/2)\alpha$ to block i.

Step 3.2.6. Apply LPB method to block i of each child schedule generated at Step 3.2.5.

Step 3.2.7. Calculate total completion time for each schedule achieved at Step 3.2.6. Set new best schedule achieved so far if it exists.

Step 3.3. Select the first P child schedules, having minimum total completion time and having jobs that can be shifted forward to block $i+1$, as parent schedules. Go to Step 3.2. If no schedules can be selected as parent schedule then Stop and output the best schedule achieved so far. Else, go to Step 3.2.

Step 4. (Last Block Improvement) If in the best schedule achieved so far last block's total PM index is less than $1 - \gamma$, then go to Step 4.1. Else, go to Step 5. ($\gamma = 0.05$ for our runs.)

Step 4.1. Starting from the first job of last block, for each job of the last block, name the job as k.

Step 4.1.1. Starting from the first block of the schedule, search for a job j with minimum $t_{m_j}^{min}$, such that $t_{m_j}^{min} < t_{m_k}$.

Step 4.1.2. If found; interchange the positions of jobs j and k. Set $t_{m_k} = t_{m_j}{}^{old}$ and $t_{m_j} = t_{m_j}^{min}$. If not found, break the loop and Go to Step 4.2.

Step 4.2. Evaluate total PM index of the last block. If it exceeds 1, solve block subproblem for the last block by LPB method.

Step 5. (Block Rearrangement) Evaluate average processing time values for all blocks.

Step 5.1. If blocks are in ascending order of average processing times, then go to Step 6. Else, rearrange blocks in ascending order of average processing times and go to Step 5.2.

Step 5.2. Apply LPB for all blocks and go to Step 5.

Step 6. Report the final schedule achieved.

In Step 1, we form the initial schedule which has minimum processing time effect on total completion time. In Step 2, we decide value for α to be used when deciding number of jobs to be shifted to a block. In Step 3, we apply branching

and fine tuning processes. In this step we generate new schedules and evaluate them. In Step 4, last block improvement procedure is employed. Finally, in Step 5, we apply block rearrangement process which also applies LPB method to blocks of schedules achieved at each iteration. After block rearrangement our search algorithm ends.

The search algorithm defined above has four other versions which will be defined below. These versions are based on three heuristics that were designed to solve block subproblem. The next algorithm we will cover is the one using FUS method.

5.2.4 Search Algorithm using FUS method

In this algorithm, block generation is as same as the algorithm using the LPB method. The difference between two methods arise in steps where we solve block subproblem. Instead of LPB, FUS is applied in **Step 3.2.2** and **Step 3.2.6**. In search algorithm using FUS, **Step 5.2** is not needed, since FUS method is not dependent to the positions of jobs. Only a single rearrangement of blocks is applied, if necessary. The next algorithm to be covered is again based on FUS method but applies LPB to the schedule resulted from branching process.

5.2.5 Search Algorithm using FUS method with LPB

Different than the algorithm using the FUS, this method applies the LPB method to the blocks of the best schedule found in **Step 3.3** of the algorithm. Also, again different than the algorithm based on FUS, we have **Step 5.2** as same as algorithm based on LPB. After each block rearrangement we apply the LPB method to all blocks of the schedule in **Step 5.1**. The next algorithm below is employing a test to decide applying LPB to the final schedule of branching process.

5.2.6 Search Algorithm using FUS method with test for LPB

Since solving LP has a computational cost we designed a test procedure for a block and applied it in this algorithm to decide whether or not to solve LP for a block. The procedure is below;
Step 1. Search first job i, which can be set to its previous breakpoint, out of the block. If found, then go to Step 1.1. Else, stop, LPB will not be applied.
Step 1.1. Calculate possible PM index loss and total completion time gain by setting job to its previous breakpoint without changing its position.
Step 2. Initialize j as the position of last job in block. Take the job in position j of the block.
Step 2.1. Calculate PM index gain and total completion time loss by setting the job j to next breakpoint.
Step 2.2. If PM loss in Step 1.1 and PM gain in Step 2.1 result in a feasible block assignment and if total completion time gain in Step 1.1 is greater than total completion time loss in Step 2.1; stop, LPB will be applied.
Else if $j = i + 1$ then stop, no more jobs to be processed.
Else, $j = j - 1$ and go to **Step 2.1**.

So, the procedure defined above first finds a job to decrease its t_m and then looks for another job to increase its t_m so that we still have a feasible schedule with better total completion time objective. This procedure is applied to decide if LPB will be applied to the schedule in **Step 3.2** of the search algorithm. This test is applied to all blocks starting from the first block up to a block it decides to apply LPB or ends with the last block. If the test decides to apply LPB to block i then LPB is applied to block i and to all blocks following block i.

The test defined above is applied to the schedule achieved at **Step 3.3** of search algorithm using FUS. So, at **Step 3.3** test is applied to blocks of the best schedule achieved so far and LPB is applied to blocks of the schedule if the test decides so. The last search algorithm below is based on TCI method.

5.2.7 Search Algorithm using TCI method

This algorithm is same as the algorithm based on FUS except that it applies TCI method in **Step 3.2** instead of FUS method.

5.3 Summary

In this chapter, we proposed search algorithms for our problem that we defined in Chapter 4. First, we introduced the subproblem that we solve many times in search algorithm. We provided three methods (LPB, FUS, TCI) to solve this subproblem. Then, we discussed the search algorithm that we designed to solve our original problem. Our search algorithm has two main parts, branching process and improvement steps. These parts were discussed in detail. Finally, we provided different versions of our search algorithm which differ by the heuristics they use to solve subproblems. In the next chapter, we will provide a numerical example to discuss the algorithms we provided here.

Chapter 6

Numerical Example

If we summarize what we did up to now, we first characterized the PM and machining conditions relationship. Then, using this relationship we defined the total completion minimization problem with controllable processing times and PM intervals where the CNC machine is unavailable. Moreover, we introduced a subproblem of this problem and designed a search algorithm based on this subproblem. In Chapter 5, the search algorithm and heuristics for the subproblem that we use in the search algorithm were described in detail. In this chapter, we will discuss the detailed steps of the proposed algorithms on a numerical example. We will present how different algorithms behave on our numerical example.

6.1 Problem Instance

We start with solving SMOP for all jobs that we have in the numerical example. For each job, we have part specifications like length and diameter of the job to be cut. We have technical constraints related to the machine, cutting tool and job part. These constraints are machine power, tool life and surface roughness. As stated in Chapter 5, the starting schedule for the search procedure is formed by setting all jobs' processing times to the minimum processing time for each job and sorting jobs by the SPT rule. Minimum processing time for each job is calculated

by solving SMOP for the minimum processing time objective. Since this is a trivial task, we will just present the set of jobs and their minimum processing times and will not get into the solution of SMOP for these jobs. Instead, search algorithm and base heuristics will be presented in detail.

For the numerical example we consider a problem instance with 20 jobs. These jobs are to be processed on a single CNC machine with machine horsepower H=10. Each job has certain specifications and certain allowable surface roughness level. Minimum processing time for each job is found as described in Chapter 3. In Table 6.1, minimum processing time levels for each job are given.

Job	min t_m	Job	min t_m
0	1.6	10	1.95
1	2.6	11	1.85
2	1.1	12	1.9
3	1.7	13	2.9
4	2.1	14	1.8
5	2.0	15	1.7
6	1.8	16	1.6
7	2.7	17	1.91
8	1.4	18	1.2
9	1.0	19	1.6

Table 6.1: Minimum Processing Times

We have a PM index function defined for the numerical example. Parameters for the PM index function are given in Table 6.2.

A	=	5
B	=	1800
k	=	2.5
C_{PM}	=	5
T	=	750

Table 6.2: PM index function parameters

So, the PM index function is:

$P_i = \frac{5t_m^{2.5}+1800}{3750t_m^{1.5}}$

Given the PM index function above, breakpoints that we achieved by relative error of 1% are given in Table 6.3.

t_m	PM index	t_m	PM index
1	0.481333	3.23	0.086993
1.12	0.406455	3.55	0.076496
1.23	0.353511	3.87	0.068208
1.33	0.314715	4.5	0.056283
1.44	0.279697	5.12	0.048258
1.63	0.232826	5.74	0.042557
1.83	0.196334	6.36	0.038406
2.02	0.169884	7	0.035250
2.2	0.150031	7.65	0.032885
2.38	0.133903	9.04	0.029713
2.56	0.120600	10.59	0.028048
2.9	0.101061	12.39	0.027526

Table 6.3: Breakpoints and PM indices

6.2 Search Procedure Parameters

In the numerical example when applying the branching process, we will generate 3 child schedules for each parent schedule. Additionally, 2 fine tuning child schedules will be generated from each parent. At each stage, we will choose 2 parent schedules. In the scheduling environment we are considering PM visit lasts 10 minutes. Mainly, we will apply search procedure using the LPB heuristic for the subproblem. We will also discuss differences of other versions of search algorithms.

Our search algorithms have two main parts as discussed in Chapter 5. The first part is the first 3 steps of the algorithm which is the branching process. This

part applies initial schedule formation followed by alternative schedule generation by branching. For each new generated schedule, a subproblem is solved. The second part of the algorithm consists of Step 4 and Step 5. This part applies improvement procedures to the schedule achieved at the end of Step 3.

6.3 Initial Schedule

Given the jobs above and the PM index function with breakpoints the initial schedule found at the end of Step 1 of branching process is given in Figure 6.1.

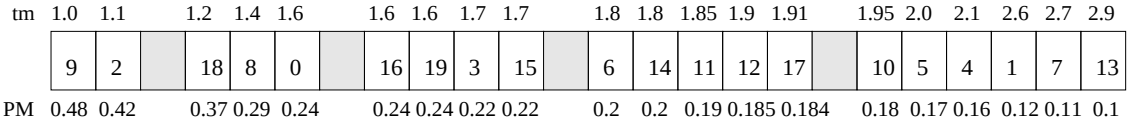

Figure 6.1: Initial schedule found at Step 1, with $\sum C_i = 830.02$

In the figure above, shaded boxes represent PM visits and other boxes are the jobs in sequence. Numbers in the boxes are the job indices. Value over each box is the current processing time of the job. Value under each box is the PM index of the job corresponding to its processing time. Sizes of the jobs in the figure are not proportional with processing times.

So, as in Figure 6.1, we have two jobs in the first block and have a PM visit following these two jobs. Then we have 3, 4, 5 and 6 jobs in the following blocks, respectively. As stated before, all jobs are set to their minimum processing time levels and they are ordered by the SPT rule. PM visits are scheduled when necessary. Total completion time for this initial schedule is 830.02. All search procedures with different base heuristics start with the same initial schedule.

As explained in Chapter 5, this schedule minimizes the processing time effect of all jobs by setting them to minimum processing times and by ordering them by the SPT rule. However, this initial schedule has many PM visits which means PM effect in total completion time is high. Having the processing time effect at

minimum at starting schedule, we look for alternative schedules with fewer PM visits in order to decrease PM effect. The aim is to decrease PM effect as much as possible while slightly increasing processing time effect so that we try to improve the total completion objective.

At Step 2 of the algorithm we set α, which is the PM index value to decide the set of jobs to be shifted. Originally, in our algorithm we set α with respect to PM visit duration, T_{PM}. If T_{PM} is high then α is also set to a high value and vice versa. This is because, when T_{PM} is high, decreasing number of PM blocks is more promising to improve total completion time. For the current example, we use $\alpha = 0.4$. When shifting jobs to a block; total PM index of jobs to be shifted to the block will be decided by α, 2α, and so on. After forming the initial schedule and setting α, we will start generating new schedules in the next section.

6.4 Schedules Generated From Initial Schedule - Stage 1

We will use the initial schedule in Figure 6.1 as the starting parent schedule to generate child schedules. At Step 3.2.1 of the search algorithm, we generate 3 child schedules from the initial schedule by shifting jobs of total PM index of 0.4, 0.8 and 1.2 to the first block. We generate 3 child schedules, because number of child schedules generated from each parent is chosen as 3 for this numerical example.

Shifting process starts with the job following the first block. For the first child schedule jobs of total PM index 0.4 is to be shifted to block 1. So, only job 18 in block 2 is shifted to block 1. After shifting job 18 to block 1, other blocks are rearranged. The sequence of jobs following block 1 is same as before, but PM visits are rescheduled according to the new PM indices of jobs. After shifting the job, block 1 is infeasible, since sum of PM indices of jobs in block 1 exceeds 1. So, we need to solve the subproblem of fitting jobs in a block, to minimize total processing time effect of jobs in block. This subproblem is solved by the LPB

heuristic and the resulting schedule is given in Figure 6.2.

LPB heuristic is quite powerful for minimizing processing time effect of a block while utilizing the PM capacity of the block. Given a block of jobs set to their minimum processing times, in LPB heuristic, these jobs are ordered by SPT rule at the beginning. Then for this job sequence piecewise linear approximation model is derived and solved for minimizing processing time effect of the block. If the resulting processing times are not in ascending order then jobs are rearranged by the SPT rule.

Continuing with the example, child schedules formed by PM index values 0.8 and 1.2 are provided in figures 6.3 and 6.4, respectively. As can be seen from figures 6.2, 6.3 and 6.4, number of jobs shifted to the first block increases for each new child schedule.

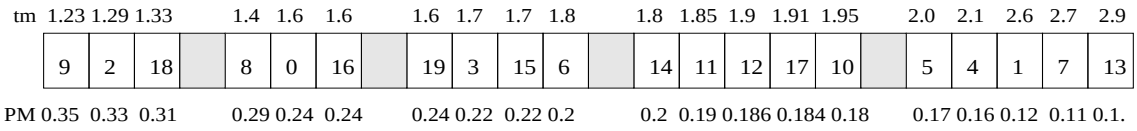

Figure 6.2: Child 1 at stage 1 generated by PM index of 0.4, $\sum C_i = 800.49$

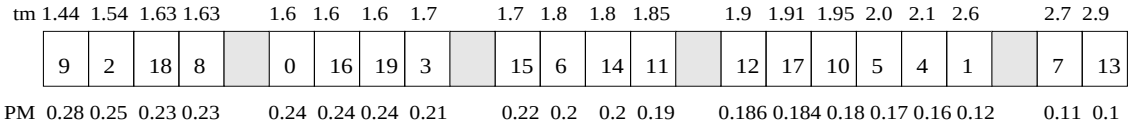

Figure 6.3: Child 2 at stage 1 generated by PM index of 0.8, $\sum C_i = 738.85$

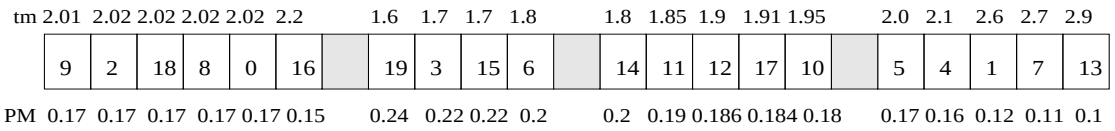

Figure 6.4: Child 3 at stage 1 generated by PM index of 1.2, $\sum C_i = 698.84$

It can easily be seen from figures 6.2, 6.3 and 6.4 that when we crowd the first block, we are forced to increase processing times of jobs in first block in order

to fit them into the block. For these first 3 child schedules generated from the initial schedule, we observe that total completion time is improved at each new child schedule. Our search algorithm is based on the idea that as we crowd the blocks we will get better schedules. However, in cases when T_PM is too low, we can expect that we do not achieve improved schedules.

Since we solved LP model provided in Chapter 5 for the first blocks of 3 child schedules at stage 1, processing times of jobs in the first blocks have changed. Some jobs are set to breakpoints and some are set to points between two breakpoints. So, we have minimized the processing time effect of the blocks for given piecewise linear approximation of PM index function.

After generating three child schedules at Step 3.2.1, we will now produce two more child schedules around the best schedule we have generated. This is what we call fine tuning in the branching process. The schedule with minimum total completion time among three child schedules is the last one(in Figure 6.4). Best total completion time we achieved so far is 698.84. Best schedule is generated by the PM index value of 1.2. So, we will form two new schedules by PM index values of 1.2+0.4/2=1.4 and 1.2-0.4/2=1.0. Here, 1.2 is the PM index value generated best schedule from the current parent and 0.4 is the step size to decide PM index values when generating child schedules. So, here we are looking for the schedules around the best one we achieved from current parent. By intuition, we expect to find child schedules with smaller total completion times compared to three child schedules we formed so far. So, two new schedules are shown in figures 6.5 and 6.6.

Figure 6.5: Child schedule 4 at stage 1 generated by 1.0, $\sum C_i = 700.81$

In figure 6.4 we see that, in best schedule firstly generated by PM index value of 1.2, first block has 6 jobs. If we check fine tuning schedules we generated in figures 6.5 and 6.6, we see that first blocks have 5 and 7 jobs respectively. So,

Figure 6.6: Child schedule 5 at stage 1 generated by 1.4, $C_i = 701.96$

fine tuning approach is in some sense looking for the neighbor schedules of the best schedule achieved initially.

For the first 3 child schedules, best total completion time we have achieved was 698.84. Two new schedules we achieved have total completion times of 700.81 and 701.96. Although we could not improve the best total completion time so far at this step, we achieved schedules with total completion times very close to the best total completion time achieved so far. So, these two child schedules have a chance to be selected as parents for the next stage.

At this point, observe that including 7 jobs in the first block resulted worse total completion time than including 6 jobs. So, we can not say that, putting as many jobs as we can to the earlier blocks and minimizing the number of blocks improves the overall completion time objective. In other words, minimizing only PM effect does not solve the problem. Minimizing processing time effect does not solve it either. So, for this problem we need a better algorithm to enumerate alternative schedules. It should use the opportunities which arise from the tradeoff between two parts of objective function.

Having generated five child schedules through initial schedule as explained above, we completed the child schedule generation for stage 1. Up to this point our aim was crowding first block of the initial schedule to obtain new schedules with improved total completion times.

The next step to be implemented in algorithm is Step 3.3. We need to select two parent schedules among five schedules we have achieved at stage 1. As stated in algorithm, we will choose two schedules having best total completion times out of five schedules. So, we select child schedule 3 (shown in Fig.6.4) and child

schedule 4 (shown in Fig.6.5), which have total completion times of 698.84 and 700.81 respectively, as parent schedules for the next stage.

Note that, while choosing parent schedules out of a schedule population, we check the ability of the schedules to be parents for the next stage. As we will try to shift jobs forward to second block of parent schedules in the next stage, selected parent schedules must have at least one job at their third blocks. This means they are capable of producing new schedules for the next stage. Our search algorithm ends when it can not find any schedules that could be parents for the next stage. Also, if it can not find parents as many as we decided at the beginning then it just goes on with the parents found.

Having selected the parents out of child schedules of stage 1, in the next section we will generate new schedules out of these two parents.

6.5 Schedules Generated at Stage 2

As we selected the parent schedules throughout the child schedules generated from the initial schedule, next thing to do is to generate new child schedules out of these two parent schedules by crowding the second blocks of them. Child schedules for this stage are generated from each parent just as we did in the last section when we were generating schedules from initial schedule. Only difference is that, this time we shift jobs forward to the second blocks. So, new child schedules, generated from child 3 of stage 1 (shown in Fig.6.4) and LPB method applied to the second blocks, are given in figures 6.7, 6.8 and 6.9.

tm	2.01	2.02	2.02	2.02	2.02	2.2		1.83	2.02	2.02	2.07	2.2	2.2		1.9	1.91	1.95	2.0	2.1	2.6		2.7	2.9
	9	2	18	8	0	16		19	3	15	6	14	11		12	17	10	5	4	1		7	13
PM	0.17	0.17	0.17	0.17	0.17	0.15		0.2	0.17	0.17	0.16	0.15	0.15		0.186	0.184	0.18	0.17	0.16	0.12		0.11	0.1

Figure 6.7: Child schedule 1 at stage 2 generated by 0.4, $\sum C_i = 670.23$

Since child 2 is the best of three schedules generated, we formed two more

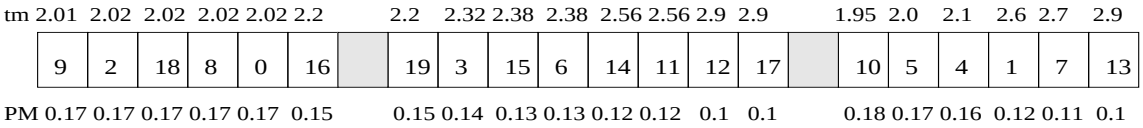

Figure 6.8: Child schedule 2 at stage 2 generated by 0.8, $\sum C_i = 668.83$

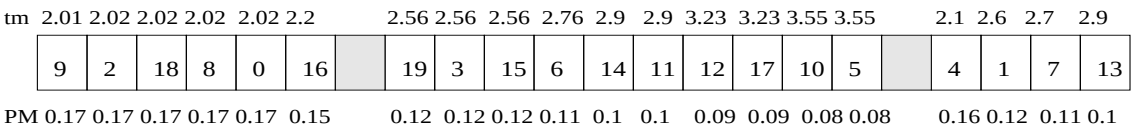

Figure 6.9: Child schedule 3 at stage 2 generated by 1.2, $\sum C_i = 692.02$

schedules, applying fine tuning in Step 3.2.5 of the algorithm, shown in figures 6.10 and 6.11. Unlike the previous stage, here, we improved the best total completion time achieved so far down to 668.49. Although it is a slight improvement, it is a good example for our intuition that we are more likely to improve total completion times around the best child we achieved up to now.

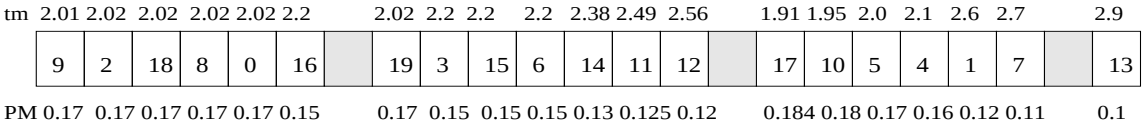

Figure 6.10: Child schedule 4 at stage 2 generated by 0.6, $\sum C_i = 668.49$

As we can see from schedules above we improved the total completion time value we have achieved so far. Child 4, having total completion time of 668.49, is the best schedule we achieved so far.

Note that at this stage we do not touch the first block of parent schedule. We just try to produce new schedules by including different number of jobs in the second block of parent schedule. The child schedules for the second parent at stage 2 (shown in Figure 6.5) are in figures 6.12, 6.13 and 6.14.

For the three child schedules generated from parent 2 of this stage, best total completion time is achieved at child 7 in Figure 6.13. So, we generate two more

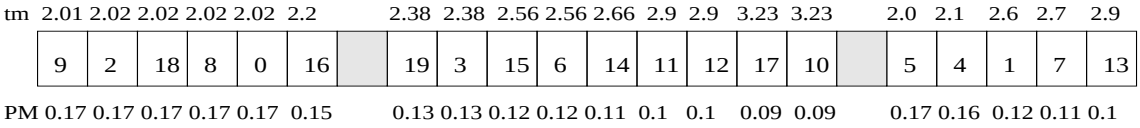

Figure 6.11: Child schedule 5 at stage 2 generated by 1.0, $\sum C_i = 680.24$

Figure 6.12: Child schedule 6 at stage 2 generated by 0.4, $\sum C_i = 689.50$

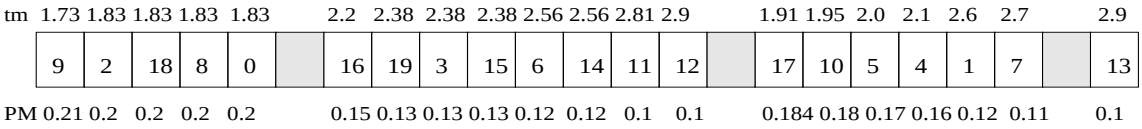

Figure 6.13: Child schedule 7 at stage 2 generated by 0.8, $\sum C_i = 680.37$

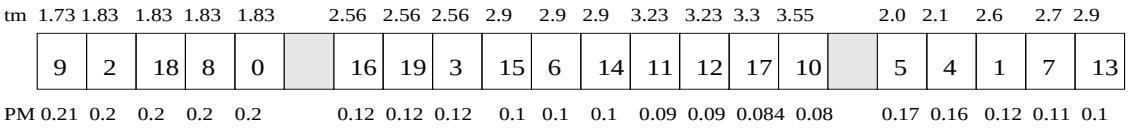

Figure 6.14: Child schedule 8 at stage 2 generated by 1.2, $\sum C_i = 699.76$

child schedules in the neighborhood of this schedule and show them in Figure 6.15 and 6.16. As can be seen in Figure 6.15, we found a better child with total completion time of 677.70 for this parent.

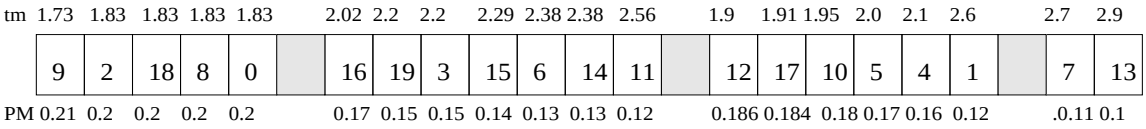

Figure 6.15: Child schedule 9 at stage 2 generated by 0.6, $\sum C_i = 677.70$

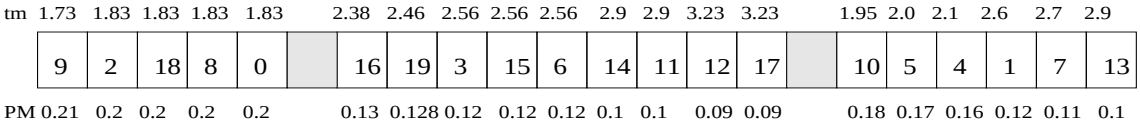

Figure 6.16: Child schedule 10 at stage 2 generated by 1.0, $\sum C_i = 684.70$

Now, at stage 2, we have ten child schedules generated out of two parent schedules. We found better schedules at this stage. This is what we expect from this search algorithm. As stated before, we start by setting all jobs to their minimum processing times and ordering them by the SPT rule. Then, we look for opportunities to improve total completion time by decreasing number of blocks and letting jobs to have higher processing times.

So, at this stage, best schedule achieved so far is the child schedule 4 with total completion time of 668.49. For stage 2, the best two schedules being able to be parents for the next stage are child 1 in figure 6.7 and child 4 in figure 6.10. So, we will try to move jobs to the third blocks of these two parent schedules in the next stage.

6.6 Schedules Generated at Stage 3

In the first parent schedule which is child 4 of previous stage (shown in Figure 6.10), there is only one job in block 4. This job is the last job of the entire

schedule. So, only one child schedule can be formed from this parent by just shifting the single last job to block 3. This child schedule can be found in figure 6.17.

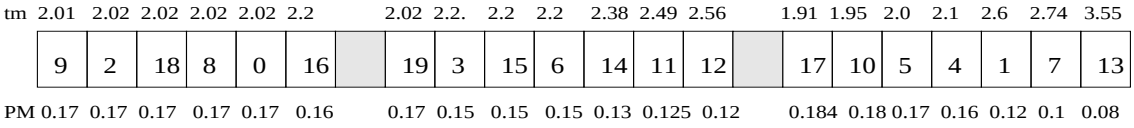

Figure 6.17: Child schedule 1 at stage 3, $\sum C_i = 659.23$

In the second parent schedule which is child 1 of stage 2 (shown in Figure 6.7), there are two jobs in block 4 which is the last block of the schedule. Our child generating process came up with two different child schedules, in figure 6.18 and 6.19, by moving those two jobs in last block one by one to block 3.

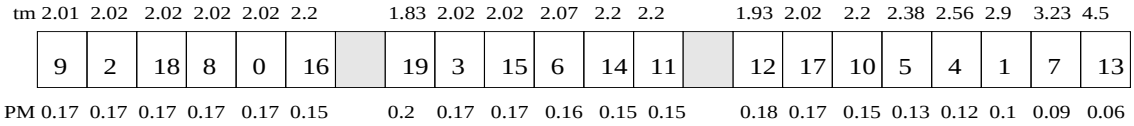

Figure 6.18: Child schedule 2 at stage 3, $\sum C_i = 660.08$

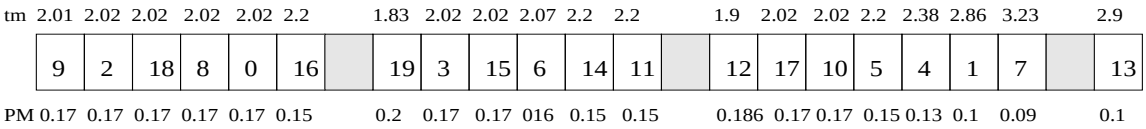

Figure 6.19: Child schedule 3 at stage 3, $\sum C_i = 665.37$

At stage 3 we achieved only three child schedules, because there were few jobs in fourth blocks of parent schedules to generate more schedules. We improved the best total completion time so far to 659.23 in child 1 in Figure 6.17. All three child schedules are formed of three blocks. This means we have no parent for the next stage. So, the search procedure based on child generation by crowding blocks ends here.

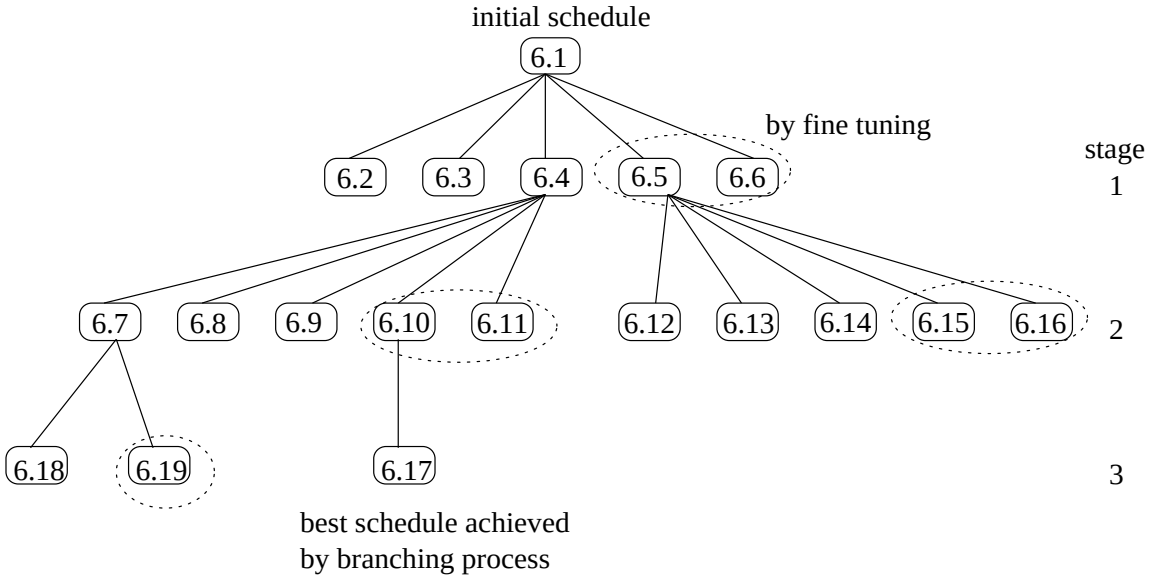

Figure 6.20: Generated tree structure by branching process - Figure no. representation

In Figure 6.20, a summary of the branching process we applied is shown. Each node represents a schedule we generated in our search algorithm. Corresponding figure index for each schedule is put on each node for the schedule. Best schedule achieved by branching process is the schedule in Figure 6.17. The initial schedule that we achieved has $\sum C_i = 830.02$ and the best schedule we found has $\sum C_i = 659.23$. We decreased total completion time by 20% by our branching process.

Before moving to the next steps of the algorithm, other heuristics than LPB to solve the block subproblem will be discussed. In the schedules we presented above, all block subproblems are solved by the LPB method. In other versions of the search algorithm, subproblems are solved by FUS or TCI methods. So, in the following two subsections those methods will be discussed. We will consider the last block of child schedule 1 at stage 3 (shown in Figure 6.17) for the discussions in the following two subsections.

6.6.1 FUS Method

As discussed in Chapter 5, the FUS method is also proposed for solving the block subproblem and it is basically focused on keeping makespan of a block as short as possible. In this section, we will first show application of the FUS method on a subproblem. Then we will mention the search algorithms based on the FUS method.

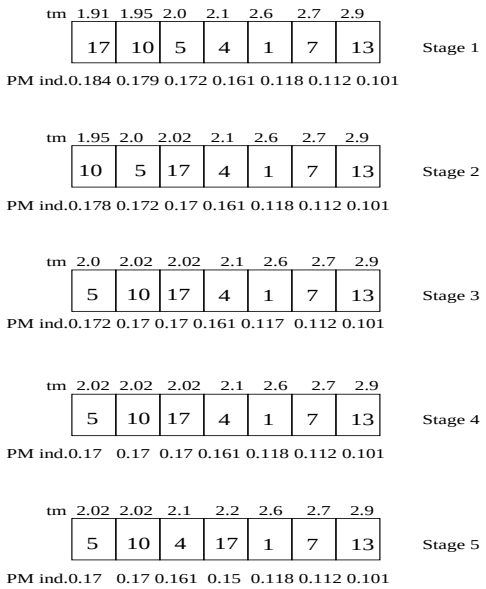

Figure 6.21: Solving block subproblem for block 3 of child 1 at stage 3 by FUS

In Figure 6.21, application of the FUS method to block 3 of child 1 at stage 3 (shown in Figure 6.17) was presented. As described in previous chapter, when a block's total PM index is above 1, FUS method always updates the job with the minimum processing time. It increases the processing time of that job to next breakpoint closest to job's processing time. Then, if the job sequence in block is not in the SPT order, algorithm rearranges the jobs by the SPT rule. This process is applied to the block until its total PM index is less than or equal to 1.

In stage 1 in Figure 6.21, total PM index of the block is 1.027, so job 17, which is the shortest job, is updated from 1.91 to 2.02. Then, jobs are rearranged. New schedule inside block is presented in stage 2 in Figure 6.21. Total PM index of

the block at stage 2 is 1.013. So, same procedure is applied until the PM index is reduced to below 1 in stage 5.

As described in previous chapter, we have designed three different search algorithms which use the FUS method to solve subproblems. One algorithm, FUS, only uses the FUS method. Another algorithm, FUS-LPB, uses the FUS method for intermediate schedules in branching process but applies LPB to all blocks of final schedule achieved by the branching process. LPB is applied to the schedule at the end of Step 3 of the algorithm. The last algorithm, FUS-test, instead of directly applying LPB to all blocks resulted from search procedure at the end of Step 3, first applies a test to decide whether or not to apply LPB.

In Figure 6.22, the schedule which is the result of branching process based on FUS method and the schedule obtained after applying LPB to this schedule are provided. Applying the LPB method to the first schedule has improved the total completion time of the entire schedule since it considers positions of jobs and it almost fully utilizes PM capacity of the block. Total completion time is decreased to 659.03 from 662.7.

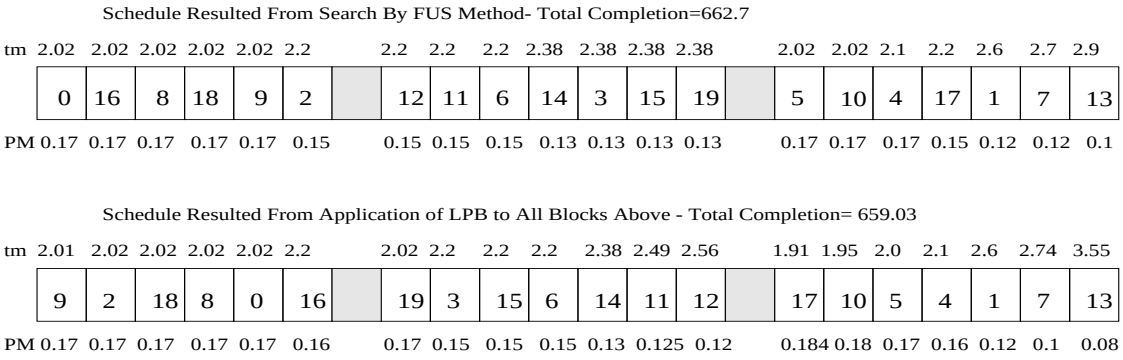

Figure 6.22: Result of search using FUS method with LPB

As explained in Chapter 5, in FUS method with tested LPB, before deciding to apply LPB method to blocks we first employ a test procedure. Test procedure starts with the first block and looks for possible improvement for each block until it finds a one or finishes all blocks without finding an improvement. If the test finds an improvement for a block then the algorithm applies the LPB method to

that block and to all blocks following that block. This could be seen in Figure 6.23. LPB is applied to block 2 and following ones but not to block 1, because test procedure could not find improvement in block 1.

What test procedure does for a block is, initially to set the first possible job in the block to its previous breakpoint or to its minimum processing time. So, the total PM index of the block is greater than 1 for the moment. Then, it looks for a job starting from the last job of the block such that setting this job's processing time to its next breakpoint will decrease total PM index of block to a value less than or equal to 1 whereas the total completion time for the schedule will improve.

For the first block of first schedule in Figure 6.23, setting job 0 to 1.83, we can not find a job to increase its processing time and improve total completion. But, in second block; if we decrease processing time of job 12 to 2.02 and increase processing time of job 19 to 2.56, total completion time is improved by 0.56 whereas total PM index of block increases to 0.9915. So, test decides improvement is possible for block 2. Then algorithm applies LPB method to block 2 and to block 3. This resulted an improve in total completion time value from 662.7 to 659.23.

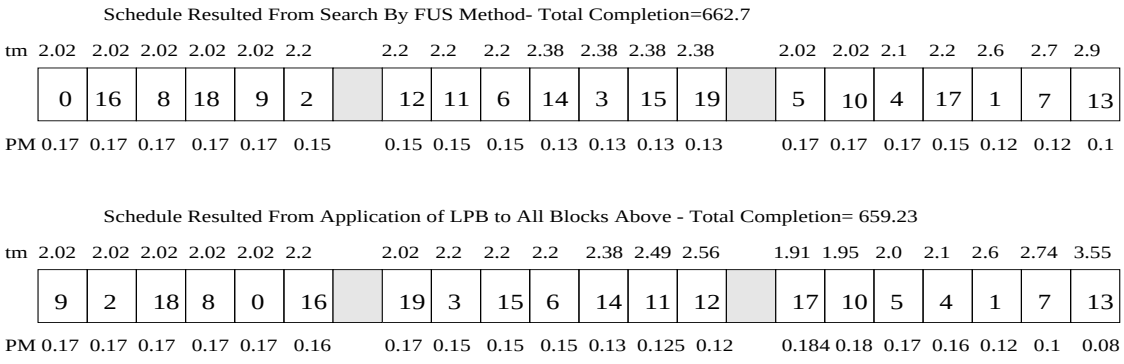

Figure 6.23: Result of search using FUS method with test for LPB

Having covered the FUS heuristic and its application in FUS based search algorithms, next we will cover the TCI heuristic.

6.6.2 TCI Method

In Chapter 5 we discussed the TCI method for solving block subproblems. We also derived a version of search algorithm which uses TCI method. In this section we will present this method on the same numerical example.

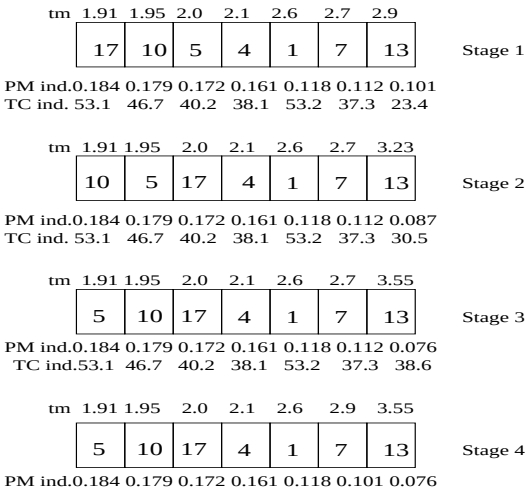

Figure 6.24: Solving block subproblem for block 3 of child 1 at stage 3 by TCI

In Figure 6.24, application of the TCI method is presented for block 3 of child 1 at stage 3 (shown in Figure 6.17). In the TCI method, we have an indexing system on jobs. We call it TCI index. It is measuring the total completion difference of the entire schedule per PM index difference of a job if the job's processing time is increased to the next breakpoint. The TCI method always updates the job with the minimum TCI index. The idea is to update the job which will have minimum effect on total completion time per its gain of PM index. After updating a job if the jobs are not in the SPT order, jobs are rearranged. Also, TCI indices are recalculated. This process is applied until total PM index of jobs in block is less than or equal to 1.

In Figure 6.24, at stage 1, we again have the same block as in the previous example for FUS. Total PM index of block is 1.027. Last job, 13, has the minimum TCI index. So, we first update job 13. Then, since the sequence is still in the

SPT order, without rearranging jobs we recalculate TCI index for job 13. We also recalculate total PM index of block. It is 1.013. So, same procedure is applied. Finally, at stage 4 total PM index of block is reduced to 0.991 and algorithm stopped.

Note that in FUS method we started with the first job in block, but in TCI we started with the last job since its TCI index is minimum. Final schedules we achieved for two methods are quite different. If we compare the total completion times of two schedules inside blocks, it is 61.66 for FUS and 60.62 for TCI method. This is due to the fact that TCI method considers contribution of each job to the objective function and each job's usage from PM capacity of block. Whereas, FUS method is just considering the makespan of block. If we compare makespan for each block, it is 16.54 for FUS but 17.01 for TCI methods.

At this point, we can also add comments about the LPB method. As long as we know the sequence of jobs in a block, we are sure that we can find optimal solution for piecewise linear approximation model by solving LP. LPB method considers the effects of positions of jobs to the total completion objective in the objective function of LP models that it solves for blocks. Also, LPB tries to fully utilize PM capacity of the block. So, in terms of minimizing processing time effect of a block LPB is more effective. Using LPB method, total completion time for the block, that we also considered above for FUS and TCI, is 60.3 whereas makespan for the block is 16.85. LPB found the best total completion time for the considered block. The schedule for the block solved by LPB is in figure 6.17, block 3.

LPB is an effective method for solving the subproblem. However, as we applied LPB in a search algorithm to hundreds of blocks, we have to pay for computation time. We can handle FUS method more quickly than LPB. In experimental design chapter we will see if these methods will make difference for results of search algorithm.

Before discussing the base heuristics above, we completed the search procedure part of the algorithm on our numerical example. We have applied the algorithm using LPB method up to Step 4. What was done up to Step 4 is searching

the solution space by generating different blocks, i.e. different schedules, and solving subproblems for generated blocks. Remember that we came up with a final schedule (in Figure 6.17) at the end of this search procedure.

Then we discussed FUS and TCI methods for solving block subproblem. We showed applications of different search procedures based on FUS which are FUS with LPB and FUS with tested LPB. So for the search procedure part of the algorithm we covered differences for different versions of LPB, FUS, TCI, FUS with LPB and FUS with tested LPB. In the following sections, improvement steps of our search algorithms will be discussed by examples.

6.7 Last Block Utilization

Last block utilization procedure is motivated by Property 2 (Block Utilization) discussed in Chapter 4. The idea is to improve the total completion time of the schedule we achieved by branching process, if the last block of the schedule is weakly utilized. Mainly, we try to move the jobs with smaller minimum processing times to the last block.

At the end of the branching process which is applied in the first three steps of the algorithm, we are likely to achieve a schedule with a last block not fully utilized, i.e. its total PM index value is less than $1 - \gamma$. We use a γ value since this procedure has a computational cost and we want to apply it if there is considerable PM capacity in the last block. When this is the case, we want to look for the opportunities to interchange jobs of the other blocks with jobs of the last block. This can improve total completion time of the schedule by higher utilization of the last block.

We will first consider the best schedule we achieved by the branching process in Figure 6.17. In Step 4 of the algorithm, we have a condition that we will apply this improvement process to the last block if it has total PM index value less than 0.95. When we apply this condition to our schedule in Figure 6.17, we see that total PM index for block 3 of schedule is 0.994. So, we do not apply last block

improvement procedure to this schedule.

Instead, to show this improvement procedure on an example we will use another schedule again derived for the problem we have been considering above. But, the schedule we will use was found by the branching process using $\alpha = 0.2$ and using the FUS method to solve subproblems. So, in Figure 6.25, application of last block improvement to this schedule can be observed.

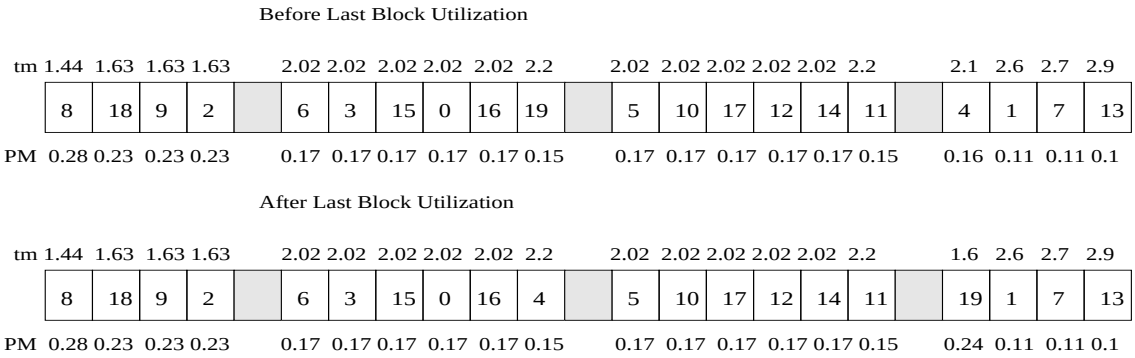

Figure 6.25: Last block utilization example

Applying Step 4 to the first schedule in Figure 6.25, we see that total PM index of last block is 0.48, so we consider it as a weakly utilized block and apply Step 4.1. We start by the first job of last block, job 4. Starting from the first block, we look for a job such that pairwise interchanging it with job 4 will improve total completion time of the schedule.

In block 1, all jobs' processing times are smaller than the minimum processing time of job 4. So, we can not assign current processing times of jobs in block 1 as processing time of job 4 after interchange due to technical constraints of SMOP. So, we skip block 1 and look at block 2. First 5 jobs in block 2 also have smaller processing times than minimum processing time of job 4. So we can not interchange any of them with job 4. But, last job of block 2, job 19, has processing time higher than minimum processing time of job 4. So, if we put job 4 into current position of job 19, we can set job 4's processing time to 2.2 which is current processing time of job 19. So, we interchange job 4 and job 19 and set processing time of job 4 to 2.2 and processing time of job 19 to 1.6 which is its

minimum processing time.

Then, we try the procedure above for following jobs in the last block. The procedure above is applied until no job could be found to interchange for a job considered in the last block. In such a case trying for further jobs is useless.

For the second job in the last block we can not find a job in other blocks to interchange with, since its minimum processing time is higher than all jobs in other blocks. So, we do not look for further jobs to interchange. We skip to the Step 4.2 of the algorithm.

We evaluate total PM index of last block in Step 4.2. It is 0.56. So, we do not need to solve block subproblem for the last block. This ends the last block improvement process. We can observe here that our total completion time has improved from 698.7 to 696.7.

Having covered last block utilization on an example, next step is the block rearrangement procedure, which will be covered in next section.

6.8 Block Rearrangement and LPB application

In Chapter 4, we modeled the problem and then introduced some optimality properties for the problem. One of the optimality properties is considering average processing time values of blocks which is the value found by dividing sum of makespan of the block and a PM visit duration to the number of jobs in the block. We know that blocks should be in ascending order of average processing times. Otherwise rearranging blocks by ascending order of average processing time improves total completion time objective.

After the last block utilization process which is applied at Step 4 of the algorithm, we evaluate average processing times of all blocks in Step 5. In Step 5.1, we check if the blocks are in ascending order of average processing times. If they are not, in Step 5.2, we apply LPB for rearranged blocks and go to Step 5. If the blocks are not again in ascending order of average processing time then

we rearrange the blocks (in Step 5.1) and then apply LPB (in Step 5.2). This continues until when the resulting blocks in the schedule after an LPB application is in ascending order of average processing times. This procedure is implemented only in search algorithms using LPB method and using FUS method with LPB.

In algorithms using FUS method, TCI method and FUS method with test for LPB, we do not include Step 5.2, i.e. we do not apply LPB after block rearrangement. This means we apply a single block rearrangement procedure and stop.

The schedule that we achieved for the numerical example in Figure 6.17 satisfies the optimality property explained above since average processing times for blocks are 3.715, 3.72 and 3.83, respectively. So, we will apply block rearrangement and LPB to a schedule which we derived by applying branching process using FUS method with LPB by $\alpha = 0.2$. The starting schedule and the schedules achieved by block rearrangement and LPB method are in Figure 6.26.

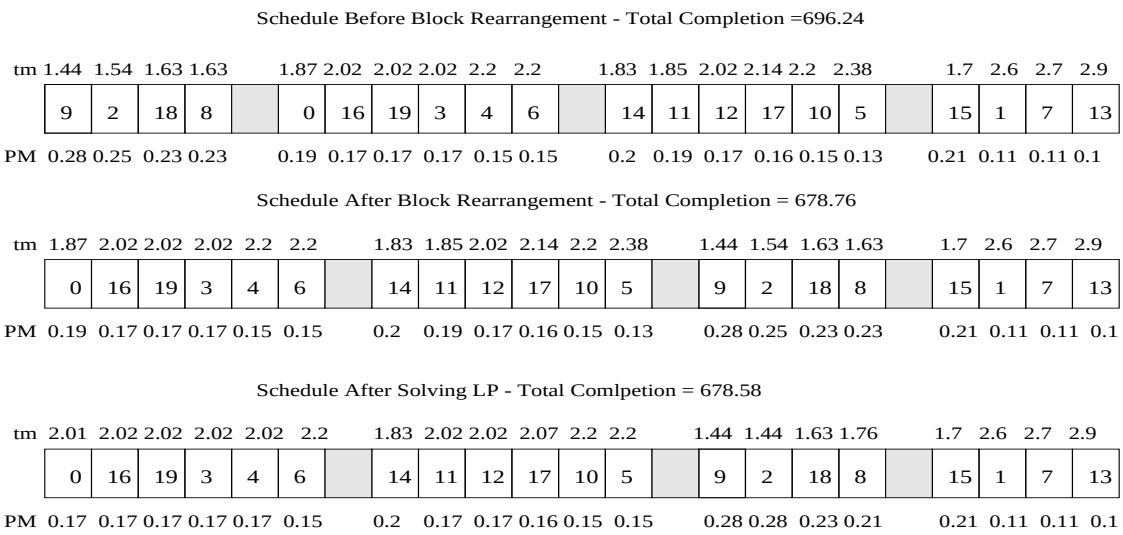

Figure 6.26: Block Rearrangement and Applying LPB

In the first schedule before block rearrangement, average processing times for the blocks are 4.06, 3.72, 3.73 and 4.98 respectively. Rearranging them in ascending order we get the second schedule in Figure 6.26. Then, since the positions of the jobs have changed as we rearranged the blocks, we again apply

LPB method to all blocks. Finally, we get the last schedule in Figure 6.26. For the final schedule average processing times for blocks are 3.71, 3.72, 4.07 and 4.98 respectively. So, the schedule satisfies the Property 3 defined in Chapter 4. Then, we stop. If the schedule did not satisfy this property then the algorithm would go back to Step 5 and again rearrange the blocks. This would be followed by application of LPB method again. These two steps, rearrange and apply LPB, would repeat until all of the blocks satisfy this property. Note, that as we rearrange blocks and apply LPB method, total completion time is improved at each new schedule we found.

So, we have discussed our algorithms and heuristics on small examples. In the next chapter, an experimental design and analysis of the results will be provided for the algorithms discussed up to now.

Chapter 7

Experimental Design

In this chapter, we first introduce the experimental factors and the parameters of the problem. We will define the machining environment for the problem (number of jobs, T_{PM}, etc.). The parameters we set for our search algorithm will be given. After defining such experimental conditions we will discuss the results of our runs.

We first give the results of single-pass results. These results are based on the machining conditions of jobs that we selected to minimize different cost objectives. Having decided processing times of jobs by different cost objectives by solving SMOP, we order the jobs by the SPT rule and calculate total completion time. The results of these runs will be discussed.

After single pass results, we will discuss results for our search algorithm. The results of the search algorithms with different base heuristics will be discussed and compared with each other.

All heuristics were coded in C language and compiled with Gnu C compiler. For the heuristics which solve linear programming models, library routines of CPLEX LP solver were used. All codes were run on SunOS 5.7 on a station Sun Enterprise 4000 with 1024 MB memory and 6 cpu of 248 Mhz.

7.1 Experimental Settings

Three experimental factors are used to test the efficiency of the algorithms. Each factor has 2 possible values. So, it is a 2^3 full factorial design. These factors are listed in Table 7.1. 10 replications of each factorial setting are taken.

Factor	Definition	Level 1	Level 2
T_{PM}	PM duration	30	90
A	PM function parameter	10	20
B	PM function parameter	30	45

Table 7.1: Experimental Design Factors

The first experimental factor in Table 7.1 is T_{PM} which is the PM duration of the system. It is the length of the time slot that the machine is not available. It is multiplier in objective function of the total completion time problem. So, it strongly influences the total completion time of a given schedule. A good heuristic should perform well for different levels of T_{PM}. T_{PM} is an experimental factor to be able to see the performances of different heuristics for different T_{PM} levels.

Other two experimental factors used in this study are A and B which are coefficients for the PM cost function that we defined in Chapter 3. In this study, PM cost function and equivalently PM index function are assumed to have a certain behavior which we derived by assumption that higher production rate increases PM cost. Having the same behavior, different PM index functions are generated for experiments to test the heuristics against.

Besides different PM duration levels and different PM index functions, a random set of jobs is needed to run the algorithms. Parameters used to generate jobs are given in Table 7.2 summarizing all machine related, tool related, job related and PM cost function parameters.

Heuristics are run for 2000 jobs which corresponds to a 5-day production rate if the machine produces at a rate of 1 job/min. Since PM is considered, a long term instance is considered to examine the heuristics. Another important issue

α	=	4.2	C_o	=	0.5
β	=	1.65	C_t	=	1.25
γ	=	1.2	S	=	UN[300,500]
C	=	56158018	D	=	UN[2.5,4]
b	=	0.9	L	=	UN[5,9]
c	=	0.78	d	=	UN[0.025,0.3]
e	=	0.65	H	=	10
C_m	=	1.706	T	=	2000
g	=	-1.54	k	=	2.5
h	=	1.04	C_{PM}	=	10
l	=	0.32	C_s	=	211825000

Table 7.2: Technical coefficients and parameters

for this problem is the number of breakpoints that are generated to solve the problem. Two levels of relative error parameter, 0.01 and 0.05, are applied for the same instances.

The search algorithm is tuned such that from each parent, 6 child schedules and 2 fine tuning child schedules are generated at most. At each level at most 8 parent schedules are selected for the next level. The algorithm uses 0.4, as the PM index value which decides how many jobs will be shifted to a block, when PM duration is low. It uses 0.6 when PM duration level is high.

7.2 Single-pass Results

As discussed in Chapter 3, by solving SMOP, we can find machine settings for a job to minimize machining+tooling+PM cost (M+T+PM), machining+tooling cost (M+T) and machining+PM cost (M+PM) objectives. We can also decide minimum processing time achievable for a job (Pl). As discussed in Chapter 4, setting a job's t_m beyond the processing time level which gives the minimum PM index level is not allowed unless the job's minimum processing time level is greater than this level. So, for each job we define an upper bound (Pu), setting job's t_m beyond which we are increasing both t_m and $P(t_m)$ of a job.

$Pu_i = \max(t_{m_i}^{min}, t_m^{minPM})$.

Here, we set machining conditions of all jobs to minimize different objective functions given above. Moreover we sequenced the jobs by the SPT rule and calculated total completion time for each schedule. So, we want to see how total completion time behaves for different cost objectives.

Setting all jobs to their minimum processing times, Pl_i , and sequencing them in the SPT order produces an upper bound for the number of PM visits, as it minimizes processing time effect of all jobs. Meanwhile, setting all jobs to their upper bounds of processing times results each job has minimum PM index. Sequencing by the SPT rule we get a schedule which is the one that has minimum number of PM visits among any SPT ordered schedules. We set all jobs to machining conditions which minimize different objectives and we generated schedules by the SPT rule. Detailed results for these schedules are in appendix, in tables A.1 to A.4. Summary results for total completion time for these schedules are given in Table 7.3.

	Objective		
Setting	Min	Max	Average
Pl	1707415	6390347	3680921
Pu	2877807	4533194	3685264
M+T	1258676	2813665	1912287
M+PM	1143572	2665664	1828462
M+T+PM	1252141	2242151	1686361

Table 7.3: SPT results for different settings

As can be seen on Table 7.3, minimum average total completion time is achieved by M+T+PM. However, when the results are examined for different levels of T_{PM} separately from Tables 7.4 and 7.6 , we see that M+PM is performing better than M+T+PM when $T_{PM} = 30$. It can also be observed that if PM duration is high then Pu is better than Pl and vice versa. We can conclude that total completion times for all settings are strongly affected by the T_{PM} level which can be seen in Table 7.5.

In tables 7.6 and 7.7, we have the analysis of variance of the paired differences of single pass results. In these tables we denote single pass results as SP(). From Table 7.7 we see that T_{PM} has significant effect on paired differences of single pass results.

It can also be concluded that rather than setting processing times to their lower or upper bounds, considering machining and PM costs results better total completion times for the problem. This is due to the fact that by considering costs we also balance the two parts of the objective function of total completion time; processing time effect and PM effect.

7.3 Results of Search Algorithms

In Chapter 5, a search algorithm is proposed for the problem. This search algorithm have 5 different versions based on the 3 heuristics proposed for the defined subproblem. The search algorithm is composed of a branching process which is up to Step 3.3 and a final improvement method of Step 4 and Step 5. In final improvement parts, last block utilization and block rearrangement are applied. For LPB and FUS-LPB, block rearrangement is applied by LPB one at a time until no improvement is possible. For other heuristics only a single block rearrangement operation is carried out. Detailed results of experimental runs for our heuristics can be found in appendix in tables A.5 to A.14.

In Table 7.8 results obtained at different steps of algorithms are summarized. At Step 3.3, we get results of basic search algorithm without any improvement. At Step 5.1, results obtained after last block utilization and first application of block rearrangement are reported.

At Step 3.3, after the branching process, on the average LPB gives the best results followed by FUS-LPB, FUS-test, TCI and FUS, respectively. At Step 5.1 after last block utilization and first block rearrangement, as expected, average results are improved. This improvement is mostly due to block rearrangement rather than last block utilization. In table, we can see that there is a considerable

Setting		N	Mean	St.Dev.	% 95 CI for Mean		Min	Max
					Lower	Upper		
Pl	0	40	2082115	336919	1974363	2189867	1707415	2448947
	1	40	5279725	1011137	4956348	5603103	4146235	6390347
	Total	80	3680920	1774627	3285996	4075844	1707415	6390347
Pu	0	40	3588858	549612	3413084	3764633	2877807	4374434
	1	40	3781668	518999	3615684	3947652	3098007	4533194
	Total	80	3685263	539918	3565110	3805416	2877807	4533194
M+T	0	40	1379701	111133	1344158	1415243	12586767	1502444
	1	40	2444873	333418	2338241	2551506	2083384	2813665
	Total	80	1912287	590098	1780967	2043607	1258676	2813665
M+PM	0	40	1250532	102696	1217688	1283376	1143572	1358869
	1	40	2406391	233307	2331776	2481007	2151078	2665664
	Total	80	1828462	608530	1693040	1963883	1143572	2665664
M+T+PM	0	40	1342653	80567	1316886	1368419	1252141	1432991
	1	40	2030068	188412	1969811	2090325	1821602	2242151
	Total	80	1686360	374646	1602987	1769734	1252141	2242151

Table 7.4: Descriptives for Single-pass Results by T_{PM}

	F	Sig.
Pl	360	.000
Pu	2.6	.111
M+T	367.4	.000
M+PM	822.4	.000
M+T+PM	450.1	.000

Table 7.5: ANOVA for Single-pass Results for T_{PM}

		N	Mean	St.Dev.	% 95 CI for mean		Min	Max
					Lower	Upper		
SP(M+T+PM)	0	40	92120	22419	84950	99290	65707	118840
-	1	40	-376323	45829	-390980	-361666	-430405	-316435
SP(M+PM)	Sum	80	-142101	238410	-195156	-89045	-430405	118840
SP(M+PM)	0	40	-129168	9075	-132071	-126266	-145004	-115104
-	1	40	-38481	101000	-70783	-6180	-161024	77329
SP(M+T)	Sum	80	-83825	84609	-102654	-64996	-161024	77329
SP(M+T+PM)	0	40	-37048	30732	-46876	-27219	-72402	-2973
-	1	40	-414805	145981	-461492	-368118	-578118	-260793
SP(M+T)	Sum	80	-225926	217056	-274230	-177623	-578118	-2973

Table 7.6: Descriptives for Differences of Single-pass Results by T_{PM}

	F	Sig.
SP(M+T+PM) - SP(M+PM)	3372.1	.000
SP(M+PM) - SP(M+T)	32	.000
SP(M+T+PM) - SP(M+T)	256.5	.000

Table 7.7: ANOVA for Differences of Single-pass Results for T_{PM}

improvement at Step 5.1. At this step on the average, FUS based heuristics are better than LPB and TCI.

At Step 6 final results are achieved. After Step 5.1, block rearrangement and LPB are applied to schedules for LPB and FUS-LPB. This is due to the fact that block rearrangement changes the positions of jobs in schedule and solving LPs for the new sequence improve results. At this last step, it is observed that FUS-LPB is doing better than the others on the average.

From Table 7.8 we can also observe how each step of an algorithm contributed to the overall improvement achieved by the algorithm. Consider LPB heuristic, average total completion time for the initial schedules is 3,680,921. At the end of branching process total completion times are reduced to average level of 1,713,384. The last block improvement step slightly improves total completion time. The last block rearrangement step has a considerable effect on total completion time so, finally, average total completion time reduces to 1,581,238. If we analyze the improvement by the LPB algorithm, we can see that, when T_{PM} is high, LPB achieved an average improvement of 63% on initial schedule and when T_{PM} is low, LPB achieved an average improvement of 40%.

Here, it is important to state that, in fact, all heuristics' results are very close to each other. But, when checked paired t-test for the results at Step 3.3, in Table 7.9, we see that we can order heuristics by their performances since they have statistically significant differences on results. The differences are small but statistically significant. So, through Table 7.9 we can derive a relationship for results of heuristics as below:
LPB $\leq$ FUS-LPB $\leq$ FUS-test $\leq$ TCI $\leq$ FUS.

If paired sample statistics for the results of heuristics are examined in Table 7.10, we again see small but significant differences between results. However, this time the order is different than it was for Step 3.3:
FUS-LPB $\leq$ FUS $\leq$ FUS-test, LPB $\leq$ TCI.

The next issue to analyze at this step is the effect of T_{PM} level to the performances of heuristics. As can be seen in Table 7.11, the difference between

Algorithm Steps	Heuristics	Objective		
		Min	Max	Average
Step 3.3 Branching Process Results	LPB	1163024	2401222	1713384
	TCI	1163206	2401620	1713643
	FUS-LPB	1163024	2401277	1713502
	FUS	1163206	2402263	1714020
	FUS-test	1163030	2401286	1713543
Step 5.1 After first rearrangement	LPB	1133300	2120267	1582935
	TCI	1133429	2120821	1583157
	FUS-LPB	1133300	2117597	1581506
	FUS	1133429	2114671	1580907
	FUS-test	1133334	2117597	1581509
Step 6 Final results	LPB	1133289	2116181	1581238
	TCI	1133429	2120821	1583157
	FUS-LPB	1133289	2113979	1580592
	FUS	1133429	2114671	1580907
	FUS-test	1133334	2117597	1581509

Table 7.8: Results at Different Steps of Search Algorithms

results of FUS-LPB and LPB increases as T_{PM} is increased. Also, Table 7.12 shows that T_{PM} level has significant effect on paired differences of results of our heuristics. It can be seen that when $T_{PM} = 30$ LPB is better than FUS and when $T_{PM} = 90$ FUS is better than LPB. When checking the results of experiments it is recognized that for the case when LPB is better than FUS the difference is quite low but there are many such cases. But, for the rare cases where FUS is better than LPB the difference between results are extremely high. So, when we look at the means and paired tests we see that FUS $\leq$ LPB although it is so rarely met in our experiments. So, another statistic, frequencies for each heuristic that they achieved best result, are also reported in Table 7.13. We see that TCI, FUS and FUS-test never achieved a best result in 80 runs. FUS-LPB achieved best result 78 times and LPB 31 times. LPB achieved almost all of the best results when T_{PM} is low. From these results we can conclude that FUS-LPB is likely to perform well for all conditions.

		Paired Differences					
				% 95 CI			
		Mean	St.Dev.	lower	upper	t	Sig.
Pair 1	LPB - TCI	-258.9	189.7	-301.1	-216.6	-12.2	.000
Pair 2	LPB - FUS-LPB	-117.4	391.4	-204.6	-30.4	-2.7	.009
Pair 3	LPB - FUS	-635.7	615.4	-772.7	-498.8	-9.2	.000
Pair 4	LPB - FUS-test	-158.4	415	-250.8	-66.1	-3.4	.001
Pair 5	TCI - FUS-LPB	141.4	360.7	61.1	221.7	3.5	.001
Pair 6	TCI - FUS	-376.8	546.5	-498.5	-255.2	-6.2	.000
Pair 7	TCI - FUS-test	100.4	386.3	14.5	186.4	2.3	.023
Pair 8	FUS-LPB - FUS	-518.3	473.2	-623.6	-413	-9.8	.000
Pair 9	FUS-LPB - FUS-test	-41	75.3	-57.7	-24.2	-4.9	.000
Pair 10	FUS - FUS-test	477.3	483.9	369.6	585	8.8	.000

Table 7.9: Paired Samples Statistics for Results at Step 3.3

		Paired Differences					
				% 95 CI			
		Mean	St.Dev.	lower	upper	t	Sig.
Pair 1	LPB - TCI	-1918.8	2236.4	-2416.4	-1421.1	-7.7	.000
Pair 2	LPB - FUS-LPB	645.5	1706.3	265.7	1025.2	3.4	.001
Pair 3	TCLP - FUS	330.8	1655.4	-37.6	699.2	1.8	.078
Pair 4	TCLP - FUS-test	-270.8	2286.6	-779.7	238	-1.1	.293
Pair 5	TCI - FUS-LPB	2564.2	3412.2	1804.8	3323.6	6.7	.000
Pair 6	TCI - FUS	2249.6	3324.1	1509.8	2989.3	6.1	.000
Pair 7	TCI - FUS-test	1647.9	2971.5	986.6	2309.2	5	.000
Pair 8	FUS-LPB - FUS	-314.6	158.6	-349.9	-279.3	-17.7	.000
Pair 9	FUS-LPB - FUS-test	-916.3	1484.6	-1246.7	-585.9	-5.5	.000
Pair 10	FUS - FUS-test	-601.7	1426.4	-919.1	-284.2	-3.8	.000

Table 7.10: Paired Samples Statistics for Final Results

		N	Mean	St.Dev.	% 95 CI for mean Lower	% 95 CI for mean Upper	Min	Max
LPB - FUS-LPB	0	40	50.2	205.2	-15.5	115.8	-7	1113
	1	40	1240.8	2264.6	516.5	1965	-1136	9339
	Total	80	645.4	1706.3	265.7	1025.2	-1136	9339
LPB - FUS	0	40	-153.8	217.1	-223.2	-84.4	-308	933
	1	40	815.4	2241	98.8	1532.2	-1454	8953
	Total	80	330.8	1655.4	-37.6	699.2	-1454	8953
LPB - FUS-test	0	40	30.8	210.1	-36.4	97.9	-184	1100
	1	40	-572.4	3218.8	-1601.8	457.0	-4913	9084
	Total	80	-270.8	2286.6	-779.7	238	-4913	9084
FUS-LPB - FUS	0	40	-204	58.4	-222.6	-185.3	-325	-98
	1	40	-425.3	149.6	-473.2	-377.4	-937	-230
	Total	80	-314.6	158.6	-349.9	-279.3	-937	-98

Table 7.11: Descriptives for Differences of Search Algorithm Results

	F	Sig.
LPB - FUS-LPB	11	.001
LPB - FUS	7.4	.008
LPB - FUS-test	1.4	.241
FUS-LPB - FUS	76	.000

Table 7.12: ANOVA for Differences of Search Algorithm Results

The next thing we examined is the effect of relative error value to total completion times and computational costs. The results of heuristics for relative error=0.01 are in appendix in tables A.5 to A.14 and results for relative error=0.05 are in appendix in tables A.15 to A.24. We include objective function and runtime summary for both relative error=0.01 and 0.05 in Table 7.14. If we compare the runtime levels between heuristics we find out the relation as below:
FUS $\leq$ FUS-test $\leq$ FUS-LPB $\leq$ LPB $\leq$ TCI.
As long as LP is used in a heuristic, runtime of the heuristic gets higher. In our experiments TCI seems worst, but a more effective coding might result lower runtime for this heuristic. But, it seems that TCI is worst of all heuristics both in terms of objective and runtime.

Heuristics	number of best results		
	total	$T_{PM} = 30$	$T_{PM} = 90$
LPB	31	29	2
TCI	0	0	0
FUS-LPB	78	39	39
FUS	0	0	0
FUS-test	0	0	0

Table 7.13: Number of Best Results for Search Algorithms

If we check the difference between two levels of relative error, we see that runtime values decrease to less than half whereas the objective function value just slightly gets worse.

Relative Error	Heuristics	Objective			Runtimes		
		Min	Max	Average	Min	Max	Average
0.01	LPB	1133289	2116181	1581238	166.7	362.6	258.9
	TCI	1133429	2120821	1583157	201.3	418.8	280.3
	FUS-LPB	1133289	2113979	1580593	43.4	84.4	57.3
	FUS	1133429	2114671	1580907	36.4	72	49.6
	FUS-test	1133334	2117597	1581509	37.7	76.4	52.3
0.05	LPB	1138251	2131447	1588461	82.2	169.2	121.3
	TCI	1139419	2134534	1590089	89.1	190.9	128.3
	FUS-LPB	1139023	2126347	1587219	21.3	37.9	26.9
	FUS	1139419	2126929	1587840	17.2	33.9	23.8
	FUS-test	1139419	2132811	1588418	17.4	33.9	23.9

Table 7.14: Results of Algorithms with different relative error levels

7.4 Summary

In this chapter, we first discussed the single-pass results of problem instances generated by setting all jobs' machining conditions to minimize different objectives. We saw that, neither setting all jobs to their minimum processing times

nor setting them to their maximum processing times gives the best results. Instead, looking for different processing times between Pl and Pu by different cost objectives, we achieved better results. This shows that we are not dealing with a straightforward problem. We need to employ effective search algorithm.

Then, we discussed results of our search algorithms using different heuristics for solving subproblems. At different stages of our search algorithm, different versions of algorithm were promising. Before block rearrangement step, LPB did better than the others whereas analyzing final results we saw that FUS-LPB did best. Although differences between results were very small, they were statistically significant. Another criteria we checked was the number of best results achieved. This showed that most of the time we achieved best result by FUS-LPB method. FUS-LPB was also one of the best search algorithms in terms of computational cost. We next considered the results of search algorithms for different relative error levels, equivalently saying different numbers of breakpoints. We saw that increasing relative error slightly increased total completion values but effectively reduced computational cost.

In the next chapter we will provide conclusions we derived in our study. We will also state the contributions of this study. Finally, future research directions will be proposed.

Chapter 8

Conclusion

In the preceding chapters we discussed the problem and proposed solution methods. Here, a brief summary of this study will be presented as well as the contributions. Also, future research topics related to this area will be introduced.

Up to now, a PM approach is discussed and single machining operation problem with PM cost is studied. A mathematical model for the problem is provided and a solution method is developed. Moreover, motivated by the PM approach proposed, minimizing total completion time with controllable processing times and machine availability are considered. We provide necessary optimality conditions as well. A subproblem for the problem is introduced and heuristics are given for the subproblem. Based on these heuristics a local search algorithm is given and experimental results are discussed.

8.1 Contributions

PM is becoming a more important tool for production. Rather than using classical PM approaches, using a special PM policy focusing the needs of a system is needed. In this study, we considered a CNC turning machine to design a new PM approach. A PM approach for CNC turning machines is proposed which is

based on the machining conditions (or production rate) of the machine. A mathematical relationship between machining conditions and PM need of a machine is established. This mathematical relationship is applied on machining conditions selection problem for turning operation. The relationship is defined by a PM index function of processing time for a job. It decides what portion of a PM visit is due to considered job. Using this function, we developed a geometric programming model which decides machining speed and feed rate that minimizes total of machining,tooling and PM costs for a job. Optimality condition for this new problem is found. Based on the optimality condition given and the convexity of objective function an algorithm is designed to find the optimal machining conditions for a cutting operation of a job.

Having derived a PM policy for the machine, we considered the total completion time problem on a machining environment where this new PM policy is applied. So, we have machine availability constraint due to PM visits and we have controllable processing times.

For the problem of total completion time minimization with PM constraint and controllable processing times, a mathematical model of MIP with nonlinear PM index terms is introduced. For the problem, necessary optimality conditions are derived.

Then, a local search algorithm is designed to solve the problem. This algorithm's one leg is solving the subproblem of processing time effect minimization, and the other is generating new schedules. To solve the subproblem of processing time effect minimization for a block, 3 different heuristics are designed. Heuristics are based on generating discrete processing time levels and choosing among them. One of the heuristics is LP based and it mainly solves piecewise linear approximation model for a block. Using these 3 heuristics, 5 different versions of search algorithm are formed. By experimental runs the performances of these algorithms are compared.

So, having proposed a new PM approach based on machining conditions, we developed a new search algorithm for the total completion time problem having

variable processing times and machine availability constraint. This search algorithm is important for being able to generate solutions for thousands of jobs in a considerably short computation time.

8.2 Future Research Directions

By this study, PM decisions are integrated with the machining conditions selection for a CNC machine. A search algorithm is introduced to solve total completion time problem. Here, only total completion objective is considered, manufacturing cost is not considered. For the next step, studying a bicriteria problem of manufacturing cost and total completion time can be considered to analyze time versus cost tradeoffs.

Another extension can be considering a multi-machine production environment with PM resource constraint such that no two machines can be maintained at the same time. Scheduling jobs on machines as well as PM visits to PM team can be studied for the same problem.

Bibliography

[1] I. Adiri, J. Bruno, E. Frostig, and A.H.G.R. Kan. Single-machine flow-time scheduling with a single breakdown. *Acta Informatica*, 26:679–696, 1989.

[2] M.S. Akturk. An exact tool allocation approach for CNC machines. *International Journal of Computer Integrated Manufacturing*, 12(2):129–140, 1999.

[3] M.S. Akturk and S. Avci. Tool allocation and machining conditions optimization for CNC machines. *European Journal of Operational Research*, 94:335–348, 1996.

[4] M.S. Akturk, J.B. Ghosh, and E.D. Gunes. Scheduling with tool changes to minimize total comletion time: A study of heuristics and their performance. To appear in Naval Research Logistics, 2003.

[5] M.S. Akturk, J.B. Ghosh, and R.K. Kayan. Scheduling with tool changes to minimize total completion time under controllable machining conditions. Technical report, Deptartment of Industrial Engineering, Bilkent University, 2002.

[6] M.S. Akturk and K. Kilic. Generating short-term observation schedules for space mission projects. *Journal of Intelligent Manufacturing*, 10:387–404, 1999.

[7] M.S. Akturk and S. Onen. Joint lot sizing and tool management in a CNC environment. *Computers in Industry*, 40:61–75, 1999.

[8] D. Banjevic, A.K.S. Jardine, V. Makis, and M. Ennis. A control-limit policy and software for condition-based maintenance optimization. *INFOR*, 39(1):32–50, FEB 2001.

[9] R.E. Barlow and L.C. Hunter. Optimum preventive maintenance policies. *Operations Research*, 8:90–100, 1960.

[10] R.E. Barlow and F. Proshan. *Mathematical Theory of Reliability*. Wiley, New York, 1965.

[11] M.S. Bazaraa, H.D. Sherali, and C.M. Shetty. *Nonlinear Programming: Theory and Algorithms*. Wiley, New York, 1993.

[12] A. Bhattacharya, R. Faria-Gonzalez, and I. Ham. Regression analysis for predicting surface finish and its application in the determination of optimum machining conditions. *Journal of Engineering Industry*, 92:711–714, 1970.

[13] J.T. Black. *The Design of The Factory With a Future*. McGraw-Hill Inc., New York, 1991.

[14] T.C.E Cheng, Z.L. Chen, and C.-L. Lee. Parallel-machine scheduling with controllable processing times. *IIE Transactions*, 28:177–180, 1996.

[15] R.L. Daniels, B.J. Hoopes, and J.B. Mazzola. Scheduling parallel manufacturing cells with resource flexibility. *Management Science*, 42(9):1260–1276, September 1996.

[16] D.S. Ermer and S. Kromordihardjo. Optimization of multi-pass turning with constraints. *ASME, Journal of Engineering for Industry*, 103:462–468, 1981.

[17] B. Gopalakrishnan and F. Al-Khayyal. Machine parameter selection for turning with constraints:an analytical approach based on geometric programming. *International Journal of Production Research*, 29(9):1897–1908, 1991.

[18] M. Gopalakrishnan, S. Mohan, and Z. He. A tabu search heuristic for preventive maintenance scheduling. *Computers and Industrial Engineering*, (40):149–160, 2001.

[19] F.E. Gorczyca. *Application of Metal Cutting Theory*. Industrial Press, 1987.

[20] G.H. Graves and C.-Y. Lee. Scheduling maintenance and semiresumable jobs on a single machine. *Naval Research Logistics*, 46:845–863, 1999.

[21] K. Hitomi. *Manufacturing systems engineering : A unified approach to manufacturing technology and production management.* Taylor and Francis, London, 1979.

[22] E. Iakovou, C.M. Ip, and C. Koulamas. Throughput-dependent periodic maintenance policies for general production units. *Annals of Operations Research*, 91:41–47, 1999.

[23] A.K.S. Jardine, editor. *Optimizing condition based maintenance decisions*, New York, 2002. Annual Reliability and Maintainability Symposium, IEEE.

[24] A.K.S. Jardine and J.A. Buzacott. Equipment reliability and maintenance. *European Journal of Operational Research*, 19:285–296, 1985.

[25] S. Karabati and P. Kouvelis. Flow-line scheduling problem with controllable processing times. *IIE Transactions*, 29:1–14, 1997.

[26] C.-Y. Lee. Current trends in dterministic scheduling. *Annals of Operations Research*, 70:1–41, 1997.

[27] C.-Y. Lee and Z.-L. Chen. Scheduling jobs and maintenance activities on parallel machines. *Naval Research Logistics*, 47:145–165, 2000.

[28] C.-Y. Lee and V.J. Leon. Machine scheduling with a rate modifying activity. *European Journal of Operational Research*, 128:119–128, 2001.

[29] C.-Y. Lee and S.D. Liman. Single machine flow-time scheduling with scheduled maintenance. *Acta Informatica*, 29:375–382, 1992.

[30] C.-Y. Lee and C.-S. Lin. Single-machine scheduling with maintenance and repair rate-modifying activities. *European Journal of operational Research*, 135:493–513, 2001.

[31] C.Y. Lee. Machine scheduling with an availability constraint. *Journal of Global Optimization*, 9(3-4):395–416.

[32] B. Malakooti and J. Deviprasad. An interactive multiple criteria approach for parameter selection in metal cutting. *Operations Research*, 37(5):805, September-October 1989.

[33] J.J. McCall. Maintenance policies for stochastically failing equipment:a survey. *Management Science*, 11(5):493–524, 1965.

[34] J.S. Mitchell. The history of condition monitoring and condition based maintenance. *Sound and Vibration*, 33(11), 1999.

[35] C. Nolden. Predictive maintenance. route to zero unplanned downtime. *Plant engineering*, 41(4):38–43, 1987.

[36] E. Nowicki and S. Zdrzalka. A survey of results for sequencing problems with controllable processing times. *Discrete Applied Mathematics*, 26:271–287, 1990.

[37] S. Osaki and T. Nakagawa. Bibliography for reliability and availability of stochastic systems. *IEEE Transactions on Reliability*, R 25:284–287, 1976.

[38] P.S. Ow and T.E. Morton. Filtered beam search in scheduling. *International Journal of Production Research*, 26:35–62, 1988.

[39] S.S. Panwalkar and R. Rajagopalan. Single-machine sequencing with controllable processing times. *European Journal of Operational Research*, 59:298–302, 1992.

[40] H. Pham and H. Wang. Imperfect maintenance. *European Journal of Operational Research*, 94:425–438, 1996.

[41] W.P. Pierskalla and J.A. Voelker. A survey of maintenance models: the control and surveillence of deteriorating systems. *Naval Research Logistics Quarterly*, 23:353–388, 1976.

[42] M. Pinedo. *Scheduling: theory, algorithms and systems.* Prentice Hall, Englewood Cliffs, 1995.

[43] X. Qi, T. Chen, and F. Tu. Scheduling the maintenance on a single machine. *Journal of the Operational Research Society*, 50:1071–1078, 1999.

[44] Y.S. Sherif and M.L. Smith. Optimal maintenance models for systems subject to failure - a review. *Naval Research Logistics Quarterly*, 28(1):47–74, 1981.

[45] Z. Simeu-Abazi and C. Sassine. Maintenance integration in manufacturing systems: From the modeling tool to evaluation. *The International Journal of Flexible Manufacturing Systems*, 13:267–285, 2001.

[46] F.W. Taylor. On the art of cutting metals. *Transaction ASME*, 28:31–350, 1906.

[47] M.A. Trick. Scheduling multiple variable-speed machines. *Operations Research*, 42(2):234–248, March-April 1994.

[48] C. Valdez-Flores and R.M. Feldman. A survey of preventive maintenance models for stochastically deteriorating single unit systems. *Naval Research Logistics*, 36:419–446, 1996.

[49] R.G. Vickson. Choosing the job sequence and processing times to minimize processing plus flow cost on a single machine. *Operations Research*, 28:1155–1167, 1980.

[50] R.G. Vickson. Two single-machine sequencing problems involving controllable job processing times. *AIEE Transactions*, 12:258–262, 1980.

[51] H. Wang. A survey of maintenance policies of deteriorating systems. *European Journal of Operational Research*, 139:469–489, 2002.

Appendix A

Computational Results

					Pl		Pu	
T_{PM}	A	B	#	seed	TC	cpu	TC	cpu
0	0	0	0	4528957	1740839	0.17	3713751	0.16
0	0	0	1	3537898	1740384	0.16	3713751	0.17
0	0	0	2	4953896	1765940	0.14	3713751	0.17
0	0	0	3	5569231	1745866	0.17	3713751	0.16
0	0	0	4	4356879	1707415	0.15	3713751	0.17
0	0	0	5	7546897	1755792	0.16	3713751	0.17
0	0	0	6	1555626	1771397	0.14	3713751	0.16
0	0	0	7	3254879	1748607	0.14	3713751	0.2
0	0	0	8	9856214	1732405	0.15	3713751	0.17
0	0	0	9	1239876	1759771	0.15	3713751	0.17
0	0	1	0	4528957	2396639	0.14	4374434	0.16
0	0	1	1	3537898	2388354	0.16	4374434	0.16
0	0	1	2	4953896	2423600	0.16	4374434	0.17
0	0	1	3	5569231	2426776	0.17	4374434	0.17
0	0	1	4	4356879	2350075	0.16	4374434	0.16
0	0	1	5	7546897	2428002	0.17	4374434	0.16
0	0	1	6	1555626	2444147	0.15	4374434	0.16
0	0	1	7	3254879	2406387	0.15	4374434	0.14
0	0	1	8	9856214	2405935	0.16	4374434	0.17
0	0	1	9	1239876	2432731	0.15	4374434	0.18
0	1	1	0	4528957	2405339	0.17	3389443	0.17
0	1	1	1	3537898	2398074	0.16	3389443	0.16
0	1	1	2	4953896	2430680	0.15	3389443	0.17
0	1	1	3	5569231	2432206	0.16	3389443	0.15
0	1	1	4	4356879	2364325	0.16	3389443	0.17
0	1	1	5	7546897	2432832	0.14	3389443	0.16
0	1	1	6	1555626	2448947	0.16	3389443	0.17
0	1	1	7	3254879	2412777	0.15	3389443	0.17
0	1	1	8	9856214	2412715	0.16	3389443	0.16
0	1	1	9	1239876	2441221	0.16	3389443	0.17
0	1	0	0	4528957	1746299	0.17	2877807	0.16
0	1	0	1	3537898	1751544	0.15	2877807	0.16
0	1	0	2	4953896	1771160	0.18	2877807	0.18
0	1	0	3	5569231	1751866	0.17	2877807	0.18
0	1	0	4	4356879	1714315	0.17	2877807	0.15
0	1	0	5	7546897	1763472	0.15	2877807	0.16
0	1	0	6	1555626	1777157	0.15	2877807	0.17
0	1	0	7	3254879	1755327	0.15	2877807	0.18
0	1	0	8	9856214	1737505	0.15	2877807	0.18
0	1	0	9	1239876	1765801	0.15	2877807	0.16

Table A.1: Single Pass Results

					Pl		Pu	
T_{PM}	A	B	#	seed	TC	cpu	TC	cpu
1	0	0	0	4528957	4238699	0.15	3837231	0.18
1	0	0	1	3537898	4254024	0.17	3837231	0.16
1	0	0	2	4953896	4337720	0.19	3837231	0.17
1	0	0	3	5569231	4287886	0.14	3837231	0.15
1	0	0	4	4356879	4146235	0.15	3837231	0.16
1	0	0	5	7546897	4304052	0.16	3837231	0.2
1	0	0	6	1555626	4357697	0.14	3837231	0.17
1	0	0	7	3254879	4281507	0.16	3837231	0.16
1	0	0	8	9856214	4225945	0.16	3837231	0.16
1	0	0	9	1239876	4305271	0.16	3837231	0.17
1	0	1	0	4528957	6206099	0.15	4533194	0.16
1	0	1	1	3537898	6197934	0.15	4533194	0.17
1	0	1	2	4953896	6310700	0.18	4533194	0.15
1	0	1	3	5569231	6330616	0.16	4533194	0.17
1	0	1	4	4356879	6074215	0.16	4533194	0.16
1	0	1	5	7546897	6320682	0.16	4533194	0.16
1	0	1	6	1555626	6375947	0.16	4533194	0.17
1	0	1	7	3254879	6254847	0.16	4533194	0.16
1	0	1	8	9856214	6246535	0.15	4533194	0.18
1	0	1	9	1239876	6324151	0.14	4533194	0.15
1	1	1	0	4528957	6232199	0.16	3658243	0.21
1	1	1	1	3537898	6227094	0.16	3658243	0.15
1	1	1	2	4953896	6331940	0.15	3658243	0.16
1	1	1	3	5569231	6346906	0.15	3658243	0.17
1	1	1	4	4356879	6116965	0.17	3658243	0.16
1	1	1	5	7546897	6335172	0.17	3658243	0.17
1	1	1	6	1555626	6390347	0.15	3658243	0.16
1	1	1	7	3254879	6274017	0.15	3658243	0.17
1	1	1	8	9856214	6266875	0.16	3658243	0.17
1	1	1	9	1239876	6349621	0.16	3658243	0.16
1	1	0	0	4528957	4255079	0.14	3098007	0.18
1	1	0	1	3537898	4287504	0.17	3098007	0.17
1	1	0	2	4953896	4353380	0.17	3098007	0.17
1	1	0	3	5569231	4305886	0.16	3098007	0.19
1	1	0	4	4356879	4166935	0.15	3098007	0.16
1	1	0	5	7546897	4327092	0.15	3098007	0.17
1	1	0	6	1555626	4374977	0.15	3098007	0.18
1	1	0	7	3254879	4301667	0.15	3098007	0.16
1	1	0	8	9856214	4241245	0.18	3098007	0.19
1	1	0	9	1239876	4323361	0.15	3098007	0.15

Table A.2: Single Pass Results

					M+T		M+PM		M+T+PM	
T_{PM}	A	B	#	seed	TC	cpu	TC	cpu	TC	cpu
0	0	0	0	4528957	1267537	0.18	1144500	0.16	1262403	3.63
0	0	0	1	3537898	1261455	0.18	1143784	0.17	1257242	3.41
0	0	0	2	4953896	1263985	0.16	1144160	0.17	1254641	3.96
0	0	0	3	5569231	1258676	0.17	1143572	0.17	1252141	3.78
0	0	0	4	4356879	1263124	0.17	1144186	0.16	1260151	3.89
0	0	0	5	7546897	1260599	0.18	1144711	0.17	1256915	4.22
0	0	0	6	1555626	1263250	0.18	1144140	0.17	1254705	4.1
0	0	0	7	3254879	1261387	0.17	1144437	0.17	1256374	4.2
0	0	0	8	9856214	1263452	0.19	1143979	0.18	1258080	4.2
0	0	0	9	1239876	1269314	0.16	1144758	0.19	1262027	4.2
0	0	1	0	4528957	1487527	0.18	1345536	0.17	1418353	3.73
0	0	1	1	3537898	1479165	0.18	1344953	0.18	1414467	3.81
0	0	1	2	4953896	1484665	0.16	1345344	0.18	1413448	4.16
0	0	1	3	5569231	1479416	0.2	1344352	0.18	1410594	4.42
0	0	1	4	4356879	1480564	0.18	1345017	0.18	1417917	4.17
0	0	1	5	7546897	1480409	0.17	1345304	0.16	1414705	3.96
0	0	1	6	1555626	1484920	0.18	1344508	0.18	1413262	4.27
0	0	1	7	3254879	1480987	0.19	1345034	0.15	1414851	5.06
0	0	1	8	9856214	1483232	0.18	1344886	0.19	1415552	4.57
0	0	1	9	1239876	1490414	0.16	1345410	0.19	1420318	3.97
0	1	1	0	4528957	1498807	0.18	1357938	0.17	1431378	4.51
0	1	1	1	3537898	1492035	0.18	1358630	0.17	1426825	4.09
0	1	1	2	4953896	1498465	0.17	1358869	0.16	1426063	4.46
0	1	1	3	5569231	1493426	0.17	1358744	0.17	1424451	4.16
0	1	1	4	4356879	1492744	0.16	1358827	0.17	1431161	3.97
0	1	1	5	7546897	1491329	0.17	1358458	0.19	1428777	3.92
0	1	1	6	1555626	1497220	0.17	1358347	0.18	1426896	4.07
0	1	1	7	3254879	1491607	0.17	1358505	0.17	1427606	4.25
0	1	1	8	9856214	1494452	0.18	1358163	0.17	1428569	3.86
0	1	1	9	1239876	1502444	0.18	1358383	0.17	1432991	4.37
0	1	0	0	4528957	1282747	0.18	1155071	0.17	1273911	4.14
0	1	0	1	3537898	1275615	0.16	1153797	0.18	1268593	4.42
0	1	0	2	4953896	1275715	0.17	1154361	0.16	1267032	4.26
0	1	0	3	5569231	1271696	0.2	1153019	0.18	1264034	4.22
0	1	0	4	4356879	1277704	0.17	1154496	0.17	1272238	4.24
0	1	0	5	7546897	1275839	0.16	1154728	0.19	1268960	4.06
0	1	0	6	1555626	1276300	0.16	1153723	0.16	1266463	3.97
0	1	0	7	3254879	1274497	0.17	1154496	0.17	1268002	4.04
0	1	0	8	9856214	1278122	0.18	1154232	0.17	1270001	4.35
0	1	0	9	1239876	1283204	0.18	1155939	0.18	1274024	3.81

Table A.3: Single Pass Results

					M+T		M+PM		M+T+PM	
T_{PM}	A	B	#	seed	TC	cpu	TC	cpu	TC	cpu
1	0	0	0	4528957	2090077	0.17	2154120	0.18	1823463	4.44
1	0	0	1	3537898	2086455	0.19	2163784	0.18	1821602	4.44
1	0	0	2	4953896	2110225	0.15	2160860	0.19	1825421	4.62
1	0	0	3	5569231	2101016	0.17	2166212	0.18	1824361	4.94
1	0	0	4	4356879	2083384	0.18	2158966	0.17	1822591	4.13
1	0	0	5	7546897	2088239	0.18	2162371	0.17	1823195	4.39
1	0	0	6	1555626	2105770	0.17	2163240	0.17	1823985	4.74
1	0	0	7	3254879	2092447	0.18	2163237	0.18	1823434	4.16
1	0	0	8	9856214	2094032	0.17	2157499	0.18	1824060	4.34
1	0	0	9	1239876	2104394	0.17	2151078	0.18	1827947	4.28
1	0	1	0	4528957	2750047	0.19	2610516	0.19	2185933	4.78
1	0	1	1	3537898	2739585	0.19	2617253	0.18	2187387	4.8
1	0	1	2	4953896	2772265	0.17	2615124	0.16	2194588	4.61
1	0	1	3	5569231	2763236	0.17	2618212	0.18	2192274	4.67
1	0	1	4	4356879	2735704	0.18	2613177	0.18	2190117	4.45
1	0	1	5	7546897	2747669	0.18	2614244	0.17	2190085	4.53
1	0	1	6	1555626	2770780	0.17	2614168	0.18	2192662	4.4
1	0	1	7	3254879	2751247	0.19	2614454	0.17	2192391	4.04
1	0	1	8	9856214	2753372	0.17	2612206	0.2	2190152	4.23
1	0	1	9	1239876	2767694	0.18	2606670	0.18	2196418	3.9
1	1	1	0	4528957	2783887	0.17	2651898	0.17	2232738	4.94
1	1	1	1	3537898	2778195	0.15	2662550	0.17	2232145	4.9
1	1	1	2	4953896	2813665	0.18	2659909	0.18	2240083	4.46
1	1	1	3	5569231	2805266	0.17	2665664	0.16	2241471	4.17
1	1	1	4	4356879	2772244	0.16	2658787	0.19	2237561	3.92
1	1	1	5	7546897	2780429	0.18	2657938	0.16	2239977	3.95
1	1	1	6	1555626	2807680	0.19	2659927	0.16	2241216	4.44
1	1	1	7	3254879	2783107	0.18	2659125	0.19	2238326	4.33
1	1	1	8	9856214	2787032	0.16	2656203	0.14	2236889	3.95
1	1	1	9	1239876	2803784	0.17	2649703	0.16	2242151	4.19
1	1	0	0	4528957	2135707	0.17	2188811	0.16	1865271	3.91
1	1	0	1	3537898	2128935	0.18	2196837	0.19	1862893	3.82
1	1	0	2	4953896	2145415	0.17	2194521	0.18	1869792	4.13
1	1	0	3	5569231	2140076	0.18	2197619	0.19	1867214	4.54
1	1	0	4	4356879	2127124	0.19	2192916	0.16	1866118	4.27
1	1	0	5	7546897	2133959	0.16	2195488	0.19	1866560	4.34
1	1	0	6	1555626	2144920	0.17	2195023	0.16	1866463	4.6
1	1	0	7	3254879	2131777	0.18	2196456	0.18	1865542	3.92
1	1	0	8	9856214	2138042	0.18	2191272	0.16	1867061	3.84
1	1	0	9	1239876	2146064	0.17	2187639	0.19	1871204	4.23

Table A.4: Single Pass Results

					LPB				
T_{PM}	A	B	#	seed	Step 3.3	Step 4.2	Step 5.1	Final	cpu
0	0	0	0	4528957	1172163	1172163	1136209	1136209	344.9
0	0	0	1	3537898	1168490	1168490	1133531	1133520	352.92
0	0	0	2	4953896	1169636	1169636	1134530	1134522	356.11
0	0	0	3	5569231	1165779	1165779	1133300	1133289	362.61
0	0	0	4	4356879	1163024	1163024	1133854	1133851	361.57
0	0	0	5	7546897	1169366	1169366	1135638	1135627	350.39
0	0	0	6	1555626	1172030	1172030	1134016	1134010	358.01
0	0	0	7	3254879	1168598	1168598	1134353	1134346	350.66
0	0	0	8	9856214	1166885	1166885	1134818	1134817	353.8
0	0	0	9	1239876	1170842	1170842	1137140	1137129	354.01
0	0	1	0	4528957	1419404	1419404	1333447	1333441	325.64
0	0	1	1	3537898	1417584	1417584	1331893	1331888	331.65
0	0	1	2	4953896	1420738	1420738	1332906	1332897	328.33
0	0	1	3	5569231	1415313	1415313	1332244	1332240	333.05
0	0	1	4	4356879	1409198	1409198	1330953	1330947	325.52
0	0	1	5	7546897	1416720	1416720	1332044	1332037	318.53
0	0	1	6	1555626	1423587	1423587	1332650	1332647	323.31
0	0	1	7	3254879	1418717	1418717	1331918	1331917	331.02
0	0	1	8	9856214	1416691	1416691	1332280	1332274	330.69
0	0	1	9	1239876	1419684	1419684	1334474	1334472	329.49
0	1	1	0	4528957	1429721	1429721	1345422	1345417	243.58
0	1	1	1	3537898	1427731	1427731	1344392	1344387	242.04
0	1	1	2	4953896	1431386	1431386	1345468	1345462	240.54
0	1	1	3	5569231	1427668	1427668	1345607	1345599	240.14
0	1	1	4	4356879	1419825	1419825	1344853	1344846	236.56
0	1	1	5	7546897	1428808	1428808	1346305	1346299	219.89
0	1	1	6	1555626	1432812	1432812	1343828	1343824	229.17
0	1	1	7	3254879	1429722	1429722	1344577	1344573	222.88
0	1	1	8	9856214	1427672	1427672	1345355	1345347	225.62
0	1	1	9	1239876	1429173	1429173	1346015	1346009	232.08
0	1	0	0	4528957	1180827	1180827	1146020	1146011	236.52
0	1	0	1	3537898	1177675	1177675	1143700	1143690	234.23
0	1	0	2	4953896	1180081	1180081	1145959	1145951	228.99
0	1	0	3	5569231	1175247	1175247	1143736	1143724	231.55
0	1	0	4	4356879	1172411	1172411	1144491	1144479	233.97
0	1	0	5	7546897	1178682	1178682	1145902	1145894	222.7
0	1	0	6	1555626	1181908	1181908	1145029	1145026	221.85
0	1	0	7	3254879	1178443	1178443	1145044	1145036	224.05
0	1	0	8	9856214	1175770	1175770	1144736	1144727	223.02
0	1	0	9	1239876	1180558	1180558	1147594	1147584	225.79

Table A.5: LPB Results, rel.err=0.01

					LPB				
T_{PM}	A	B	#	seed	Step 3.3	Step 4.2	Step 5.1	Final	cpu
1	0	0	0	4528957	1874948	1874948	1741083	1737754	304.21
1	0	0	1	3537898	1870899	1868838	1731989	1730157	303.72
1	0	0	2	4953896	1873177	1870588	1731605	1730602	294.54
1	0	0	3	5569231	1865459	1862743	1731572	1730655	295.05
1	0	0	4	4356879	1856741	1856741	1739795	1735920	303.52
1	0	0	5	7546897	1874126	1872348	1735551	1734525	289.2
1	0	0	6	1555626	1874940	1873551	1730457	1728040	290.64
1	0	0	7	3254879	1871556	1871556	1742459	1738720	287.57
1	0	0	8	9856214	1865157	1862120	1730577	1728850	296.39
1	0	0	9	1239876	1873432	1871306	1733629	1732438	279.23
1	0	1	0	4528957	2349035	2349035	2069381	2063664	294.26
1	0	1	1	3537898	2346024	2346024	2066857	2060923	289.94
1	0	1	2	4953896	2353487	2353487	2074949	2068943	295.25
1	0	1	3	5569231	2346991	2346991	2072022	2066225	284.98
1	0	1	4	4356879	2327189	2327189	2068916	2062889	275.65
1	0	1	5	7546897	2348380	2348380	2069555	2063337	287.11
1	0	1	6	1555626	2360548	2360548	2074304	2068397	283.83
1	0	1	7	3254879	2341704	2341704	2066939	2061066	287.54
1	0	1	8	9856214	2343778	2343778	2068067	2061705	276.38
1	0	1	9	1239876	2357659	2357659	2072183	2065405	279.81
1	1	1	0	4528957	2392491	2392491	2116536	2113945	184.53
1	1	1	1	3537898	2385592	2385592	2117479	2113557	182.55
1	1	1	2	4953896	2393723	2393723	2120267	2116181	183.29
1	1	1	3	5569231	2384209	2384209	2116892	2112813	180.95
1	1	1	4	4356879	2371817	2371027	2119728	2114508	180.03
1	1	1	5	7546897	2389556	2389556	2119994	2115960	173.86
1	1	1	6	1555626	2397793	2397793	2117359	2113675	177
1	1	1	7	3254879	2384689	2384689	2113260	2111671	172.92
1	1	1	8	9856214	2385204	2385204	2117177	2113553	173.35
1	1	1	9	1239876	2401222	2401222	2118130	2114120	176.37
1	1	0	0	4528957	1907637	1907637	1782633	1780545	166.71
1	1	0	1	3537898	1910569	1910569	1784586	1782595	170.16
1	1	0	2	4953896	1910470	1910470	1782103	1780550	170.81
1	1	0	3	5569231	1902001	1902001	1781704	1779825	171.49
1	1	0	4	4356879	1894458	1893073	1780346	1778541	170.97
1	1	0	5	7546897	1912616	1912616	1787729	1785622	168.06
1	1	0	6	1555626	1919754	1919754	1783806	1783616	167.35
1	1	0	7	3254879	1906668	1906668	1780099	1778672	168.35
1	1	0	8	9856214	1902289	1902289	1782167	1780251	167.86
1	1	0	9	1239876	1912211	1912211	1784720	1782683	168.4

Table A.6: LPB Results, rel.err=0.01

					TCI			
T_{PM}	A	B	#	seed	Step 3.3	Step 4.2	Final	cpu
0	0	0	0	4528957	1172412	1172412	1136393	341.39
0	0	0	1	3537898	1168674	1168674	1133698	349.27
0	0	0	2	4953896	1169773	1169773	1134620	345.88
0	0	0	3	5569231	1165908	1165908	1133429	340.14
0	0	0	4	4356879	1163206	1163206	1134007	351.64
0	0	0	5	7546897	1169534	1169534	1135749	362.05
0	0	0	6	1555626	1172221	1172221	1134152	351.49
0	0	0	7	3254879	1168227	1168227	1133826	363.98
0	0	0	8	9856214	1167092	1167092	1134973	364.37
0	0	0	9	1239876	1171006	1171006	1137281	354.26
0	0	1	0	4528957	1419745	1419745	1333688	236.84
0	0	1	1	3537898	1417687	1417687	1332026	226.20
0	0	1	2	4953896	1421041	1421041	1333112	231.16
0	0	1	3	5569231	1415712	1415712	1332509	230.07
0	0	1	4	4356879	1409547	1409547	1331189	234.92
0	0	1	5	7546897	1416948	1416948	1332177	247.96
0	0	1	6	1555626	1423955	1423955	1332860	249.29
0	0	1	7	3254879	1418986	1418986	1332108	241.20
0	0	1	8	9856214	1417103	1417103	1332531	244.36
0	0	1	9	1239876	1419954	1419954	1334631	240.67
0	1	1	0	4528957	1430003	1430003	1345626	216.04
0	1	1	1	3537898	1428082	1428082	1344678	217.04
0	1	1	2	4953896	1431762	1431762	1345809	206.86
0	1	1	3	5569231	1428072	1428072	1345896	216.17
0	1	1	4	4356879	1420147	1420147	1345065	213.86
0	1	1	5	7546897	1429176	1429176	1346612	214.62
0	1	1	6	1555626	1433176	1433176	1344115	234.29
0	1	1	7	3254879	1430042	1430042	1344871	212.33
0	1	1	8	9856214	1428107	1428107	1345651	217.74
0	1	1	9	1239876	1429447	1429447	1346240	211.17
0	1	0	0	4528957	1181020	1181020	1146165	276.01
0	1	0	1	3537898	1177894	1177894	1143875	265.36
0	1	0	2	4953896	1180266	1180266	1146124	267.32
0	1	0	3	5569231	1175440	1175440	1143898	260.46
0	1	0	4	4356879	1172624	1172624	1144664	277.21
0	1	0	5	7546897	1177611	1177611	1144961	272.95
0	1	0	6	1555626	1182099	1182099	1145171	267.94
0	1	0	7	3254879	1178672	1178672	1145247	275.23
0	1	0	8	9856214	1175871	1175871	1144833	284.80
0	1	0	9	1239876	1180824	1180824	1147807	275.28

Table A.7: TCI Results, rel.err=0.01

					TCI			
T_{PM}	A	B	#	seed	Step 3.3	Step 4.2	Final	cpu
1	0	0	0	4528957	1875168	1875168	1741338	409.54
1	0	0	1	3537898	1871141	1869082	1732154	400.8
1	0	0	2	4953896	1873378	1870789	1731767	375.88
1	0	0	3	5569231	1865700	1862989	1731780	364.45
1	0	0	4	4356879	1856932	1856932	1740037	418.8
1	0	0	5	7546897	1874370	1872593	1735645	390.42
1	0	0	6	1555626	1875129	1873740	1730620	380.08
1	0	0	7	3254879	1871779	1871779	1742710	398.13
1	0	0	8	9856214	1865345	1862314	1730789	412.85
1	0	0	9	1239876	1873647	1871524	1733798	399.72
1	0	1	0	4528957	2349357	2349357	2069719	303.25
1	0	1	1	3537898	2346418	2346418	2067056	288.47
1	0	1	2	4953896	2353832	2353832	2075265	313.67
1	0	1	3	5569231	2347358	2347358	2072387	293.89
1	0	1	4	4356879	2327501	2327501	2069198	289.77
1	0	1	5	7546897	2348766	2348766	2069868	307.21
1	0	1	6	1555626	2360811	2360811	2074553	312.93
1	0	1	7	3254879	2341986	2341986	2067209	298.25
1	0	1	8	9856214	2344079	2344079	2068242	296.58
1	0	1	9	1239876	2358002	2358002	2072436	294.98
1	1	1	0	4528957	2392981	2392981	2116950	216.39
1	1	1	1	3537898	2385856	2385856	2117742	205.61
1	1	1	2	4953896	2394164	2394164	2120821	206.87
1	1	1	3	5569231	2384562	2384562	2117173	201.31
1	1	1	4	4356879	2372198	2371408	2120131	217.38
1	1	1	5	7546897	2389851	2389851	2120301	210.74
1	1	1	6	1555626	2398241	2398241	2117802	209.17
1	1	1	7	3254879	2385043	2385043	2113736	210.26
1	1	1	8	9856214	2385684	2385684	2117735	211.36
1	1	1	9	1239876	2401620	2401620	2118551	210.94
1	1	0	0	4528957	1907876	1907876	1782883	254.69
1	1	0	1	3537898	1910856	1910856	1784907	264.92
1	1	0	2	4953896	1910735	1910735	1782354	253.04
1	1	0	3	5569231	1902264	1902264	1781919	250.66
1	1	0	4	4356879	1894706	1893320	1780605	271
1	1	0	5	7546897	1912879	1912879	1788067	263.44
1	1	0	6	1555626	1920117	1920117	1784115	255.43
1	1	0	7	3254879	1906930	1906930	1780404	256.37
1	1	0	8	9856214	1902715	1902715	1782606	258.56
1	1	0	9	1239876	1912434	1912434	1784923	256.02

Table A.8: TCI Results, rel.err=0.01

					FUS-LPB				
T_{PM}	A	B	#	seed	Step 3.3	Step 4.2	Step 5.1	Final	cpu
0	0	0	0	4528957	1172163	1172163	1136209	1136209	65.29
0	0	0	1	3537898	1168490	1168490	1133531	1133520	66.46
0	0	0	2	4953896	1169636	1169636	1134530	1134522	67.26
0	0	0	3	5569231	1165779	1165779	1133300	1133289	65.63
0	0	0	4	4356879	1163024	1163024	1133854	1133851	66.58
0	0	0	5	7546897	1169365	1169365	1135636	1135625	67.59
0	0	0	6	1555626	1172030	1172030	1134016	1134010	66.48
0	0	0	7	3254879	1168023	1168023	1133654	1133646	68.09
0	0	0	8	9856214	1166885	1166885	1134818	1134817	67.38
0	0	0	9	1239876	1170842	1170842	1137140	1137129	70.78
0	0	1	0	4528957	1419404	1419404	1333447	1333441	49.92
0	0	1	1	3537898	1417397	1417397	1331785	1331780	48.68
0	0	1	2	4953896	1420738	1420738	1332906	1332897	49.52
0	0	1	3	5569231	1415313	1415313	1332244	1332240	49.60
0	0	1	4	4356879	1409198	1409198	1330953	1330947	49.57
0	0	1	5	7546897	1416720	1416720	1332044	1332037	50.70
0	0	1	6	1555626	1423587	1423587	1332650	1332647	50.93
0	0	1	7	3254879	1418717	1418717	1331918	1331917	49.55
0	0	1	8	9856214	1416737	1416737	1332286	1332281	50.10
0	0	1	9	1239876	1419684	1419684	1334474	1334472	50.01
0	1	1	0	4528957	1429721	1429721	1345422	1345417	44.87
0	1	1	1	3537898	1427719	1427719	1344375	1344370	46.12
0	1	1	2	4953896	1431374	1431374	1345447	1345441	43.62
0	1	1	3	5569231	1427657	1427657	1345590	1345582	45.29
0	1	1	4	4356879	1419825	1419825	1344853	1344846	44.30
0	1	1	5	7546897	1428804	1428804	1346297	1346290	43.85
0	1	1	6	1555626	1432812	1432812	1343828	1343824	47.11
0	1	1	7	3254879	1429707	1429707	1344560	1344556	43.44
0	1	1	8	9856214	1427672	1427672	1345355	1345347	44.28
0	1	1	9	1239876	1429173	1429173	1346015	1346009	43.93
0	1	0	0	4528957	1180827	1180827	1146020	1146011	53.19
0	1	0	1	3537898	1177675	1177675	1143700	1143690	51.69
0	1	0	2	4953896	1180081	1180081	1145959	1145951	52.04
0	1	0	3	5569231	1175247	1175247	1143736	1143724	50.84
0	1	0	4	4356879	1172404	1172404	1144484	1144473	53.20
0	1	0	5	7546897	1177434	1177434	1144789	1144781	51.66
0	1	0	6	1555626	1181908	1181908	1145029	1145026	50.65
0	1	0	7	3254879	1178443	1178443	1145044	1145036	52.05
0	1	0	8	9856214	1175766	1175766	1144733	1144724	53.78
0	1	0	9	1239876	1180558	1180558	1147594	1147584	52.86

Table A.9: FUS-LPB Results, rel.err=0.01

					FUS-LPB				
T_{PM}	A	B	#	seed	Step 3.3	Step 4.2	Step 5.1	Final	cpu
1	0	0	0	4528957	1875897	1873071	1733173	1732716	83.73
1	0	0	1	3537898	1870899	1869106	1729716	1729524	77.46
1	0	0	2	4953896	1873177	1870611	1730976	1730444	78.31
1	0	0	3	5569231	1865459	1862784	1730601	1730452	71.45
1	0	0	4	4356879	1857663	1856462	1735383	1734482	84.44
1	0	0	5	7546897	1874126	1872350	1735430	1734485	79.08
1	0	0	6	1555626	1874940	1873952	1727329	1727163	72.95
1	0	0	7	3254879	1873151	1871781	1737071	1736401	81.52
1	0	0	8	9856214	1865157	1862247	1729195	1728633	81.22
1	0	0	9	1239876	1873432	1871315	1733279	1732367	80.34
1	0	1	0	4528957	2349082	2349082	2068027	2063659	63.69
1	0	1	1	3537898	2346128	2346128	2065463	2060622	66.19
1	0	1	2	4953896	2354131	2353567	2059858	2059604	63.7
1	0	1	3	5569231	2347012	2347012	2071138	2066187	67.43
1	0	1	4	4356879	2327808	2327808	2064972	2060975	66.48
1	0	1	5	7546897	2348443	2348443	2068114	2063318	67.89
1	0	1	6	1555626	2361674	2361129	2060572	2060278	63.35
1	0	1	7	3254879	2341757	2341757	2065676	2061056	60.78
1	0	1	8	9856214	2343915	2343915	2066387	2061444	66.63
1	0	1	9	1239876	2359135	2358293	2061889	2061527	65.39
1	1	1	0	4528957	2392490	2392490	2116533	2113631	48.85
1	1	1	1	3537898	2385677	2385677	2115934	2113524	48.22
1	1	1	2	4953896	2394484	2393871	2113625	2112261	47.61
1	1	1	3	5569231	2384246	2384246	2116008	2112345	47.11
1	1	1	4	4356879	2371816	2371363	2115483	2113979	49.34
1	1	1	5	7546897	2390234	2389731	2113439	2112040	47.14
1	1	1	6	1555626	2397833	2397833	2116290	2113668	47.46
1	1	1	7	3254879	2384689	2384689	2113260	2111652	47.84
1	1	1	8	9856214	2385257	2385257	2116550	2112643	47.76
1	1	1	9	1239876	2401277	2401277	2117597	2113494	48.16
1	1	0	0	4528957	1908891	1908878	1781747	1781681	50.7
1	1	0	1	3537898	1910659	1910659	1783919	1782584	53.09
1	1	0	2	4953896	1910576	1910576	1781701	1780518	51
1	1	0	3	5569231	1902086	1902086	1781170	1779768	53.6
1	1	0	4	4356879	1894458	1893584	1778515	1778237	53.93
1	1	0	5	7546897	1912706	1911684	1781224	1780981	52.5
1	1	0	6	1555626	1919752	1919752	1783835	1783616	50.86
1	1	0	7	3254879	1906757	1906757	1779840	1778643	54.71
1	1	0	8	9856214	1902392	1902392	1781601	1780224	51.16
1	1	0	9	1239876	1912359	1912359	1783737	1782642	51.05

Table A.10: FUS-LPB Results, rel.err=0.01

					FUS			
T_{PM}	A	B	#	seed	Step 3.3	Step 4.2	Final	cpu
0	0	0	0	4528957	1172412	1172412	1136393	59.89
0	0	0	1	3537898	1168674	1168674	1133698	61.29
0	0	0	2	4953896	1169773	1169773	1134620	60.52
0	0	0	3	5569231	1165908	1165908	1133429	59.33
0	0	0	4	4356879	1163206	1163206	1134007	61.62
0	0	0	5	7546897	1169534	1169534	1135750	62.44
0	0	0	6	1555626	1172221	1172221	1134152	60.84
0	0	0	7	3254879	1168227	1168227	1133826	62.69
0	0	0	8	9856214	1167092	1167092	1134973	62.23
0	0	0	9	1239876	1171006	1171006	1137281	62.25
0	0	1	0	4528957	1419745	1419745	1333688	43.52
0	0	1	1	3537898	1417687	1417687	1332026	41.83
0	0	1	2	4953896	1421041	1421041	1333112	42.77
0	0	1	3	5569231	1415712	1415712	1332509	42.72
0	0	1	4	4356879	1409547	1409547	1331189	42.95
0	0	1	5	7546897	1416948	1416948	1332177	44.61
0	0	1	6	1555626	1423955	1423955	1332860	45.23
0	0	1	7	3254879	1418986	1418986	1332108	43.45
0	0	1	8	9856214	1417103	1417103	1332531	44.22
0	0	1	9	1239876	1419954	1419954	1334631	44.07
0	1	1	0	4528957	1430003	1430003	1345626	40.14
0	1	1	1	3537898	1428094	1428094	1344695	40.67
0	1	1	2	4953896	1431732	1431732	1345760	38.69
0	1	1	3	5569231	1428040	1428040	1345846	40.24
0	1	1	4	4356879	1420147	1420147	1345065	39.82
0	1	1	5	7546897	1429144	1429144	1346549	39.39
0	1	1	6	1555626	1433176	1433176	1344115	42.44
0	1	1	7	3254879	1430032	1430032	1344860	39.00
0	1	1	8	9856214	1428107	1428107	1345651	39.93
0	1	1	9	1239876	1429447	1429447	1346240	39.33
0	1	0	0	4528957	1181020	1181020	1146165	48.93
0	1	0	1	3537898	1177894	1177894	1143875	47.11
0	1	0	2	4953896	1180266	1180266	1146124	47.40
0	1	0	3	5569231	1175440	1175440	1143898	46.33
0	1	0	4	4356879	1172625	1172625	1144664	49.14
0	1	0	5	7546897	1177611	1177611	1144961	47.97
0	1	0	6	1555626	1182099	1182099	1145171	46.77
0	1	0	7	3254879	1178672	1178672	1145247	48.45
0	1	0	8	9856214	1175877	1175877	1144838	49.79
0	1	0	9	1239876	1180824	1180824	1147807	48.82

Table A.11: FUS Results, rel.err=0.01

					FUS			
T_{PM}	A	B	#	seed	Step 3.3	Step 4.2	Final	cpu
1	0	0	0	4528957	1876189	1873363	1732946	70.49
1	0	0	1	3537898	1871141	1869350	1729916	69.15
1	0	0	2	4953896	1873418	1870852	1730748	65.15
1	0	0	3	5569231	1865707	1863032	1730756	63.03
1	0	0	4	4356879	1858027	1856826	1734881	72.04
1	0	0	5	7546897	1874528	1872752	1734815	67
1	0	0	6	1555626	1875129	1874141	1727535	65.04
1	0	0	7	3254879	1873383	1872006	1736697	68.24
1	0	0	8	9856214	1865386	1862476	1729000	70.8
1	0	0	9	1239876	1873769	1871652	1732801	68.63
1	0	1	0	4528957	2350958	2350958	2064157	53.7
1	0	1	1	3537898	2347782	2347782	2060957	51.29
1	0	1	2	4953896	2354495	2353931	2059990	55.53
1	0	1	3	5569231	2348687	2348687	2066523	51.86
1	0	1	4	4356879	2328547	2328547	2061247	51.84
1	0	1	5	7546897	2350585	2350585	2063703	53.91
1	0	1	6	1555626	2361948	2361403	2060663	55.23
1	0	1	7	3254879	2343560	2343560	2061447	52.65
1	0	1	8	9856214	2345508	2345508	2061767	52.06
1	0	1	9	1239876	2359528	2358686	2061912	52.86
1	1	1	0	4528957	2393161	2393161	2114161	38.91
1	1	1	1	3537898	2386519	2386519	2113807	37.12
1	1	1	2	4953896	2395097	2394483	2113198	37.2
1	1	1	3	5569231	2385185	2385185	2112768	36.39
1	1	1	4	4356879	2372320	2371867	2114671	39
1	1	1	5	7546897	2390660	2390157	2112837	37.85
1	1	1	6	1555626	2399435	2399435	2114145	37.25
1	1	1	7	3254879	2385131	2385131	2112185	37.69
1	1	1	8	9856214	2386358	2386358	2113203	37.97
1	1	1	9	1239876	2402263	2402263	2113953	38.04
1	1	0	0	4528957	1909135	1909135	1781999	44.34
1	1	0	1	3537898	1911999	1911999	1783068	46.3
1	1	0	2	4953896	1911431	1911431	1780827	44.2
1	1	0	3	5569231	1903024	1903024	1780061	43.91
1	1	0	4	4356879	1894770	1893649	1778694	47.25
1	1	0	5	7546897	1913077	1911776	1781680	46.2
1	1	0	6	1555626	1920170	1920170	1783976	44.37
1	1	0	7	3254879	1907612	1907612	1779042	44.4
1	1	0	8	9856214	1903644	1903644	1780814	44.62
1	1	0	9	1239876	1913378	1913378	1782930	44.67

Table A.12: FUS Results, rel.err=0.01

					FUS-test			
T_{PM}	A	B	#	seed	Step 3.3	Step 4.2	Final	cpu
0	0	0	0	4528957	1172412	1172412	1136393	60.16
0	0	0	1	3537898	1168507	1168507	1133531	63.74
0	0	0	2	4953896	1169773	1169773	1134620	60.75
0	0	0	3	5569231	1165837	1165837	1133334	60.85
0	0	0	4	4356879	1163030	1163030	1133855	64.00
0	0	0	5	7546897	1169409	1169409	1135640	64.60
0	0	0	6	1555626	1172056	1172056	1134017	63.26
0	0	0	7	3254879	1168049	1168049	1133655	64.93
0	0	0	8	9856214	1166922	1166922	1134823	64.68
0	0	0	9	1239876	1170958	1170958	1137202	62.99
0	0	1	0	4528957	1419404	1419404	1333447	46.57
0	0	1	1	3537898	1417397	1417397	1331785	45.40
0	0	1	2	4953896	1420738	1420738	1332906	46.33
0	0	1	3	5569231	1415313	1415313	1332244	45.96
0	0	1	4	4356879	1409209	1409209	1330953	46.22
0	0	1	5	7546897	1416720	1416720	1332044	47.65
0	0	1	6	1555626	1423587	1423587	1332650	48.43
0	0	1	7	3254879	1418717	1418717	1331918	46.45
0	0	1	8	9856214	1416737	1416737	1332286	47.16
0	0	1	9	1239876	1419684	1419684	1334474	47.18
0	1	1	0	4528957	1429721	1429721	1345422	42.35
0	1	1	1	3537898	1427719	1427719	1344375	42.81
0	1	1	2	4953896	1431374	1431374	1345447	41.09
0	1	1	3	5569231	1427657	1427657	1345590	42.69
0	1	1	4	4356879	1419825	1419825	1344853	42.44
0	1	1	5	7546897	1428804	1428804	1346297	41.70
0	1	1	6	1555626	1432812	1432812	1343828	44.60
0	1	1	7	3254879	1429707	1429707	1344560	41.36
0	1	1	8	9856214	1427672	1427672	1345355	42.17
0	1	1	9	1239876	1429173	1429173	1346015	41.63
0	1	0	0	4528957	1180827	1180827	1146020	50.85
0	1	0	1	3537898	1177680	1177680	1143700	49.44
0	1	0	2	4953896	1180198	1180198	1146027	48.27
0	1	0	3	5569231	1175248	1175248	1143736	48.52
0	1	0	4	4356879	1172444	1172444	1144489	50.53
0	1	0	5	7546897	1177456	1177456	1144794	49.35
0	1	0	6	1555626	1181908	1181908	1145029	48.63
0	1	0	7	3254879	1178509	1178509	1145073	49.54
0	1	0	8	9856214	1175804	1175804	1144754	50.82
0	1	0	9	1239876	1180562	1180562	1147594	50.83

Table A.13: FUS-test Results, rel.err=0.01

					FUS-test			
T_{PM}	A	B	#	seed	Step 3.3	Step 4.2	Final	cpu
1	0	0	0	4528957	1875906	1873080	1733173	75.09
1	0	0	1	3537898	1870899	1869106	1729716	73.56
1	0	0	2	4953896	1873177	1870611	1730976	70.42
1	0	0	3	5569231	1865478	1862803	1730601	67.19
1	0	0	4	4356879	1857674	1856473	1735383	76.44
1	0	0	5	7546897	1874126	1872350	1735430	71.69
1	0	0	6	1555626	1874944	1873957	1727329	68.97
1	0	0	7	3254879	1873290	1871914	1737102	69.53
1	0	0	8	9856214	1865386	1862476	1729000	70.89
1	0	0	9	1239876	1873477	1871361	1733286	71.64
1	0	1	0	4528957	2349112	2349112	2068028	58.01
1	0	1	1	3537898	2346128	2346128	2065463	55.72
1	0	1	2	4953896	2354155	2353590	2059859	59.56
1	0	1	3	5569231	2347029	2347029	2071138	56.82
1	0	1	4	4356879	2327822	2327822	2064973	56.13
1	0	1	5	7546897	2348443	2348443	2068114	58.94
1	0	1	6	1555626	2361781	2361236	2060579	58.4
1	0	1	7	3254879	2341767	2341767	2065676	56.59
1	0	1	8	9856214	2343915	2343915	2066387	56.75
1	0	1	9	1239876	2359135	2358293	2061889	57.62
1	1	1	0	4528957	2392498	2392498	2116533	42.35
1	1	1	1	3537898	2385698	2385698	2115934	40.7
1	1	1	2	4953896	2394668	2394054	2113748	39.11
1	1	1	3	5569231	2384300	2384300	2116009	39.47
1	1	1	4	4356879	2371923	2371470	2115510	41.23
1	1	1	5	7546897	2390660	2390157	2112837	37.7
1	1	1	6	1555626	2397840	2397840	2116290	40.77
1	1	1	7	3254879	2384729	2384729	2113265	39.94
1	1	1	8	9856214	2385428	2385428	2116612	39.62
1	1	1	9	1239876	2401286	2401286	2117597	41.41
1	1	0	0	4528957	1908963	1908949	1781776	46.1
1	1	0	1	3537898	1910659	1910659	1783919	50.1
1	1	0	2	4953896	1910576	1910576	1781701	47.71
1	1	0	3	5569231	1902197	1902197	1781206	45.87
1	1	0	4	4356879	1894458	1893584	1778515	50.42
1	1	0	5	7546897	1912706	1911684	1781224	49.26
1	1	0	6	1555626	1919860	1919860	1783846	46.62
1	1	0	7	3254879	1907039	1907039	1780033	44.67
1	1	0	8	9856214	1902392	1902392	1781601	48.18
1	1	0	9	1239876	1912359	1912359	1783737	48

Table A.14: FUS-test Results, rel.err=0.01

					LPB				
T_{PM}	A	B	#	seed	Step 3.3	Step 4.2	Step 5.1	Final	cpu
0	0	0	0	4528957	1179470	1179470	1141152	1141146	162.33
0	0	0	1	3537898	1176161	1176161	1139813	1139802	169.23
0	0	0	2	4953896	1176477	1176477	1139700	1139693	168.71
0	0	0	3	5569231	1172659	1172659	1138261	1138251	167.07
0	0	0	4	4356879	1173442	1173442	1142509	1142509	163.96
0	0	0	5	7546897	1178087	1178087	1142439	1142429	167.35
0	0	0	6	1555626	1180553	1180553	1140801	1140794	166.19
0	0	0	7	3254879	1178964	1178964	1141986	1141978	163.99
0	0	0	8	9856214	1177167	1177167	1142858	1142848	164.17
0	0	0	9	1239876	1179796	1179796	1144058	1144048	161.56
0	0	1	0	4528957	1430058	1430058	1342184	1342184	151.29
0	0	1	1	3537898	1430565	1430565	1342150	1342147	149.38
0	0	1	2	4953896	1431859	1431859	1342937	1342933	151.67
0	0	1	3	5569231	1425231	1425231	1339508	1339505	153.55
0	0	1	4	4356879	1419502	1419502	1339913	1339911	152.12
0	0	1	5	7546897	1430685	1430685	1342719	1342717	144.6
0	0	1	6	1555626	1433057	1433057	1340297	1340295	149.86
0	0	1	7	3254879	1431620	1431620	1342121	1342119	147.62
0	0	1	8	9856214	1429232	1429232	1342457	1342455	152.97
0	0	1	9	1239876	1432728	1432728	1344722	1344720	148.01
0	1	1	0	4528957	1434923	1434923	1349300	1349297	113.42
0	1	1	1	3537898	1434152	1434152	1349474	1349472	115.2
0	1	1	2	4953896	1437415	1437415	1349768	1349765	113.39
0	1	1	3	5569231	1432206	1432206	1348794	1348792	114.78
0	1	1	4	4356879	1426553	1426553	1349763	1349761	111.96
0	1	1	5	7546897	1434194	1434194	1349954	1349952	109.79
0	1	1	6	1555626	1438966	1438966	1348030	1348027	112.25
0	1	1	7	3254879	1436579	1436579	1349437	1349435	111.17
0	1	1	8	9856214	1433547	1433547	1349666	1349663	110.39
0	1	1	9	1239876	1437234	1437234	1352062	1352059	110.52
0	1	0	0	4528957	1184839	1184839	1148656	1148655	116.73
0	1	0	1	3537898	1183076	1183076	1147372	1147366	116.65
0	1	0	2	4953896	1183193	1183193	1147687	1147686	115.39
0	1	0	3	5569231	1179194	1179194	1146041	1146041	115.66
0	1	0	4	4356879	1176521	1176521	1146989	1146984	112.67
0	1	0	5	7546897	1183142	1183142	1148848	1148844	110.5
0	1	0	6	1555626	1185827	1185827	1147749	1147747	111.21
0	1	0	7	3254879	1183033	1183033	1148185	1148183	112.61
0	1	0	8	9856214	1180406	1180406	1147958	1147953	110.81
0	1	0	9	1239876	1184658	1184658	1150153	1150152	110.96

Table A.15: LPB Results, rel.err=0.05

					LPB				
T_{PM}	A	B	#	seed	Step 3.3	Step 4.2	Step 5.1	Final	cpu
1	0	0	0	4528957	1872627	1872476	1734772	1734771	132.18
1	0	0	1	3537898	1875378	1873381	1735510	1733971	136.7
1	0	0	2	4953896	1880754	1880754	1738060	1738045	133.51
1	0	0	3	5569231	1874329	1874329	1741272	1741262	133.97
1	0	0	4	4356879	1859192	1857072	1733280	1732101	138.98
1	0	0	5	7546897	1882705	1880942	1739076	1737357	134.58
1	0	0	6	1555626	1884414	1883124	1737592	1735672	131.52
1	0	0	7	3254879	1875069	1875069	1736685	1736650	132.13
1	0	0	8	9856214	1867534	1867500	1736143	1736133	136.02
1	0	0	9	1239876	1883629	1880242	1734281	1733448	132.13
1	0	1	0	4528957	2369233	2369233	2089528	2084754	119.49
1	0	1	1	3537898	2361906	2361906	2082677	2077065	126.35
1	0	1	2	4953896	2370407	2370407	2090074	2085161	125.44
1	0	1	3	5569231	2361005	2361005	2086694	2080942	122.06
1	0	1	4	4356879	2345650	2345650	2086948	2081956	119.96
1	0	1	5	7546897	2364213	2364213	2085979	2081238	124.83
1	0	1	6	1555626	2374359	2374359	2089524	2083005	124.77
1	0	1	7	3254879	2355173	2355173	2082274	2077360	126.28
1	0	1	8	9856214	2365579	2365579	2092224	2086416	120.79
1	0	1	9	1239876	2372235	2370989	2079231	2071902	124.6
1	1	1	0	4528957	2405597	2405597	2128602	2125988	85.8
1	1	1	1	3537898	2395947	2394813	2118479	2116074	86.81
1	1	1	2	4953896	2402602	2401693	2120932	2117158	85.77
1	1	1	3	5569231	2392306	2392306	2123669	2120449	85.3
1	1	1	4	4356879	2381932	2381175	2127613	2123853	85.38
1	1	1	5	7546897	2396017	2395199	2119639	2116499	84.54
1	1	1	6	1555626	2405878	2405878	2124217	2120416	85.18
1	1	1	7	3254879	2395619	2395619	2117942	2117573	83.18
1	1	1	8	9856214	2400867	2400867	2134282	2131447	82.56
1	1	1	9	1239876	2408565	2408565	2122463	2119634	84.38
1	1	0	0	4528957	1917048	1917048	1792096	1790121	84.64
1	1	0	1	3537898	1914068	1914068	1788849	1787007	86.21
1	1	0	2	4953896	1918284	1918284	1790416	1789424	83.59
1	1	0	3	5569231	1909127	1909127	1789719	1788314	84.7
1	1	0	4	4356879	1904554	1903364	1791878	1790849	84.18
1	1	0	5	7546897	1920786	1919478	1791616	1790732	83.02
1	1	0	6	1555626	1922636	1922636	1787567	1787538	86.38
1	1	0	7	3254879	1915301	1915301	1788386	1787308	83.11
1	1	0	8	9856214	1912214	1912214	1793106	1791594	82.25
1	1	0	9	1239876	1915305	1915305	1789075	1787398	84.98

Table A.16: LPB Results, rel.err=0.05

					TCI			
T_{PM}	A	B	#	seed	Step 3.3	Step 4.2	Final	cpu
0	0	0	0	4528957	1179946	1179946	1141505	159.21
0	0	0	1	3537898	1176486	1176486	1140066	160.15
0	0	0	2	4953896	1176984	1176984	1140030	159.28
0	0	0	3	5569231	1174181	1174181	1139419	155.3
0	0	0	4	4356879	1172396	1172396	1140841	160.91
0	0	0	5	7546897	1175740	1175740	1139861	165.42
0	0	0	6	1555626	1179919	1179919	1140175	162.28
0	0	0	7	3254879	1179432	1179432	1142279	157.43
0	0	0	8	9856214	1177652	1177652	1143244	161.58
0	0	0	9	1239876	1180144	1180144	1144255	166.83
0	0	1	0	4528957	1430225	1430225	1342668	113.78
0	0	1	1	3537898	1430333	1430333	1342105	107.43
0	0	1	2	4953896	1432366	1432366	1342797	108.94
0	0	1	3	5569231	1425849	1425849	1339950	109.2
0	0	1	4	4356879	1421312	1421312	1341991	114.26
0	0	1	5	7546897	1431521	1431492	1343309	109.12
0	0	1	6	1555626	1433754	1433754	1340738	115.32
0	0	1	7	3254879	1432348	1432348	1342634	111.54
0	0	1	8	9856214	1430102	1430102	1343113	113.81
0	0	1	9	1239876	1431687	1431687	1343912	116.14
0	1	1	0	4528957	1434523	1434523	1348305	99.44
0	1	1	1	3537898	1434947	1434947	1349936	96.32
0	1	1	2	4953896	1437041	1437041	1349190	95.17
0	1	1	3	5569231	1432569	1432569	1349032	98.58
0	1	1	4	4356879	1427025	1427025	1350207	101.88
0	1	1	5	7546897	1434848	1434848	1350341	98.9
0	1	1	6	1555626	1439749	1439749	1348358	103.88
0	1	1	7	3254879	1437288	1437288	1349779	96.71
0	1	1	8	9856214	1434264	1434264	1350103	98.87
0	1	1	9	1239876	1436755	1436755	1351024	101.06
0	1	0	0	4528957	1185246	1185246	1148972	127.08
0	1	0	1	3537898	1183800	1183800	1147840	121.78
0	1	0	2	4953896	1184932	1184932	1149429	123.29
0	1	0	3	5569231	1180449	1180449	1147104	120.63
0	1	0	4	4356879	1176957	1176957	1147293	128.39
0	1	0	5	7546897	1183594	1183594	1149113	126.08
0	1	0	6	1555626	1186486	1186486	1148177	120.71
0	1	0	7	3254879	1183321	1183321	1148363	125.16
0	1	0	8	9856214	1180619	1180619	1147940	127.06
0	1	0	9	1239876	1185048	1185048	1150374	129.09

Table A.17: TCI Results, rel.err=0.05

					TCI			
T_{PM}	A	B	#	seed	Step 3.3	Step 4.2	Final	cpu
1	0	0	0	4528957	1873237	1873064	1735313	189.56
1	0	0	1	3537898	1876004	1874017	1735988	185.53
1	0	0	2	4953896	1881303	1881303	1738329	177.2
1	0	0	3	5569231	1875012	1875012	1741654	171.63
1	0	0	4	4356879	1859722	1857606	1733625	190.94
1	0	0	5	7546897	1883316	1881567	1739669	181.94
1	0	0	6	1555626	1884833	1883555	1737877	177.57
1	0	0	7	3254879	1875514	1875514	1736866	181.72
1	0	0	8	9856214	1867872	1867835	1736490	185.82
1	0	0	9	1239876	1884176	1880789	1734717	185.62
1	0	1	0	4528957	2370193	2370193	2090181	132.03
1	0	1	1	3537898	2362797	2362797	2083488	133.48
1	0	1	2	4953896	2371312	2371312	2091026	135.49
1	0	1	3	5569231	2361487	2361487	2087314	130.51
1	0	1	4	4356879	2346188	2346188	2087773	135.27
1	0	1	5	7546897	2365154	2365154	2086878	141
1	0	1	6	1555626	2375314	2375314	2090611	140.44
1	0	1	7	3254879	2356272	2356272	2083264	140.12
1	0	1	8	9856214	2366751	2366751	2093466	134.15
1	0	1	9	1239876	2373410	2372183	2079958	139.32
1	1	1	0	4528957	2406549	2406549	2125565	95.07
1	1	1	1	3537898	2396886	2395759	2119249	91.21
1	1	1	2	4953896	2403232	2402330	2121727	91.05
1	1	1	3	5569231	2392933	2392933	2124218	89.53
1	1	1	4	4356879	2382998	2382245	2128204	97.7
1	1	1	5	7546897	2396644	2395838	2120518	93.14
1	1	1	6	1555626	2406791	2406791	2124913	92.54
1	1	1	7	3254879	2396574	2396574	2118116	91.69
1	1	1	8	9856214	2401609	2401609	2134534	92.45
1	1	1	9	1239876	2409512	2409512	2122928	95.56
1	1	0	0	4528957	1917703	1917703	1792511	117.31
1	1	0	1	3537898	1914765	1914765	1789347	119.79
1	1	0	2	4953896	1919013	1919013	1790970	113.63
1	1	0	3	5569231	1909636	1909636	1790282	113.69
1	1	0	4	4356879	1905072	1903823	1792282	123.59
1	1	0	5	7546897	1921368	1920019	1791965	118.84
1	1	0	6	1555626	1923303	1923303	1788056	115.92
1	1	0	7	3254879	1916061	1916061	1788726	115.87
1	1	0	8	9856214	1912637	1912637	1793464	116.59
1	1	0	9	1239876	1915858	1915858	1789310	117.33

Table A.18: TCI Results, rel.err=0.05

					FUS-LPB				
T_{PM}	A	B	#	seed	Step 3.3	Step 4.2	Step 5.1	Final	cpu
0	0	0	0	4528957	1179470	1179470	1141152	1141146	31.15
0	0	0	1	3537898	1176161	1176161	1139813	1139802	31.43
0	0	0	2	4953896	1176477	1176477	1139700	1139693	31.36
0	0	0	3	5569231	1173691	1173691	1139033	1139023	30.53
0	0	0	4	4356879	1171897	1171897	1140422	1140414	31.02
0	0	0	5	7546897	1175322	1175322	1139595	1139585	32.02
0	0	0	6	1555626	1179209	1179209	1139695	1139689	31.63
0	0	0	7	3254879	1178955	1178955	1141974	1141967	30.68
0	0	0	8	9856214	1177167	1177167	1142858	1142848	31.11
0	0	0	9	1239876	1179786	1179786	1144046	1144035	31.66
0	0	1	0	4528957	1429531	1429531	1342170	1342166	24.34
0	0	1	1	3537898	1429489	1429489	1341534	1341532	23.61
0	0	1	2	4953896	1431896	1431896	1342482	1342479	23.68
0	0	1	3	5569231	1425231	1425231	1339508	1339505	23.75
0	0	1	4	4356879	1420359	1420359	1340884	1340883	24.46
0	0	1	5	7546897	1430698	1430669	1342803	1342820	24.55
0	0	1	6	1555626	1433057	1433057	1340297	1340295	24.58
0	0	1	7	3254879	1431620	1431620	1342121	1342119	24.08
0	0	1	8	9856214	1429232	1429232	1342457	1342455	24.28
0	0	1	9	1239876	1431143	1431143	1343661	1343659	24.38
0	1	1	0	4528957	1433752	1433752	1347819	1347816	21.81
0	1	1	1	3537898	1434144	1434144	1349458	1349456	21.64
0	1	1	2	4953896	1436385	1436385	1348831	1348829	21.26
0	1	1	3	5569231	1431937	1431937	1348626	1348624	21.84
0	1	1	4	4356879	1426456	1426456	1349779	1349777	22.08
0	1	1	5	7546897	1434188	1434188	1349942	1349940	21.69
0	1	1	6	1555626	1439015	1439015	1347977	1347974	22.56
0	1	1	7	3254879	1436554	1436554	1349388	1349386	21.27
0	1	1	8	9856214	1433547	1433547	1349666	1349663	21.48
0	1	1	9	1239876	1436101	1436101	1350716	1350714	21.94
0	1	0	0	4528957	1184861	1184861	1148654	1148653	25.46
0	1	0	1	3537898	1183076	1183076	1147372	1147366	24.51
0	1	0	2	4953896	1184484	1184484	1149165	1149158	25.26
0	1	0	3	5569231	1179966	1179966	1146842	1146837	24.34
0	1	0	4	4356879	1176502	1176502	1146966	1146960	25.63
0	1	0	5	7546897	1183142	1183142	1148847	1148844	25.16
0	1	0	6	1555626	1185827	1185827	1147749	1147747	24.15
0	1	0	7	3254879	1183026	1183026	1148178	1148176	25.04
0	1	0	8	9856214	1180212	1180212	1147655	1147649	25.24
0	1	0	9	1239876	1184658	1184658	1150153	1150151	25.63

Table A.19: FUS-LPB Results, rel.err=0.05

					FUS-LPB				
T_{PM}	A	B	#	seed	Step 3.3	Step 4.2	Step 5.1	Final	cpu
1	0	0	0	4528957	1872516	1872345	1734536	1734435	36.81
1	0	0	1	3537898	1875253	1873788	1732154	1730743	37.27
1	0	0	2	4953896	1880633	1880633	1737693	1737678	34.21
1	0	0	3	5569231	1874301	1874301	1741194	1741184	33.64
1	0	0	4	4356879	1859061	1857291	1730911	1730170	36.93
1	0	0	5	7546897	1882574	1881417	1733879	1733116	35.59
1	0	0	6	1555626	1884293	1883532	1733135	1732069	34.81
1	0	0	7	3254879	1874936	1874936	1736258	1736223	35.29
1	0	0	8	9856214	1867439	1867402	1735912	1735898	35.99
1	0	0	9	1239876	1883590	1880306	1731361	1730653	37.93
1	0	1	0	4528957	2369138	2369138	2087883	2084329	30.74
1	0	1	1	3537898	2361903	2361903	2080287	2076218	30.6
1	0	1	2	4953896	2370321	2370321	2088203	2084515	30.94
1	0	1	3	5569231	2360861	2360861	2085003	2080401	28.08
1	0	1	4	4356879	2345554	2345554	2085095	2081337	30.55
1	0	1	5	7546897	2364126	2364126	2083978	2080358	31.48
1	0	1	6	1555626	2374526	2373943	2068781	2066156	31.22
1	0	1	7	3254879	2355843	2355843	2071943	2071935	28.94
1	0	1	8	9856214	2365431	2365431	2090595	2085947	28.5
1	0	1	9	1239876	2372235	2371435	2071808	2070921	28.98
1	1	1	0	4528957	2405519	2405519	2125264	2121298	23.14
1	1	1	1	3537898	2395792	2394871	2116354	2114688	22.64
1	1	1	2	4953896	2402469	2401881	2117517	2114996	22.47
1	1	1	3	5569231	2392093	2392093	2121799	2117687	22.17
1	1	1	4	4356879	2381787	2381442	2123727	2121257	23.34
1	1	1	5	7546897	2395899	2395436	2116866	2115602	22.86
1	1	1	6	1555626	2405571	2405571	2122114	2119392	22.49
1	1	1	7	3254879	2395462	2395462	2117680	2116784	22.17
1	1	1	8	9856214	2400706	2400706	2132307	2126347	22.42
1	1	1	9	1239876	2408490	2408490	2121818	2117090	22.92
1	1	0	0	4528957	1916766	1916410	1785070	1784980	24.17
1	1	0	1	3537898	1913712	1913712	1787310	1786091	24.94
1	1	0	2	4953896	1918032	1918032	1788509	1787795	24.23
1	1	0	3	5569231	1909009	1909009	1788056	1787021	23.86
1	1	0	4	4356879	1904132	1903361	1787821	1787131	26.94
1	1	0	5	7546897	1920351	1919361	1787598	1786736	26.24
1	1	0	6	1555626	1922225	1922225	1786757	1786724	23.99
1	1	0	7	3254879	1915076	1915076	1786411	1785793	24.31
1	1	0	8	9856214	1911986	1911986	1790833	1789947	24.24
1	1	0	9	1239876	1914972	1914972	1787019	1786039	24.17

Table A.20: FUS-LPB Results, rel.err=0.05

					FUS			
T_{PM}	A	B	#	seed	Step 3.3	Step 4.2	Final	cpu
0	0	0	0	4528957	1179946	1179946	1141505	28.89
0	0	0	1	3537898	1176487	1176487	1140067	29.28
0	0	0	2	4953896	1176984	1176984	1140030	29.06
0	0	0	3	5569231	1174181	1174181	1139419	28.52
0	0	0	4	4356879	1172396	1172396	1140841	29.08
0	0	0	5	7546897	1175740	1175740	1139861	30
0	0	0	6	1555626	1179919	1179919	1140176	29.66
0	0	0	7	3254879	1179433	1179433	1142280	28.51
0	0	0	8	9856214	1177652	1177652	1143244	29.14
0	0	0	9	1239876	1180146	1180146	1144257	29.73
0	0	1	0	4528957	1430221	1430221	1342662	21.95
0	0	1	1	3537898	1430373	1430373	1342153	21.1
0	0	1	2	4953896	1432383	1432383	1342819	21.3
0	0	1	3	5569231	1425849	1425849	1339950	21.38
0	0	1	4	4356879	1421242	1421242	1341571	22.11
0	0	1	5	7546897	1431521	1431492	1343309	21.14
0	0	1	6	1555626	1433754	1433754	1340738	22.28
0	0	1	7	3254879	1432348	1432348	1342634	21.64
0	0	1	8	9856214	1430102	1430102	1343113	22.06
0	0	1	9	1239876	1431710	1431710	1343940	22.6
0	1	1	0	4528957	1434533	1434533	1348316	19.45
0	1	1	1	3537898	1434950	1434950	1349942	19.35
0	1	1	2	4953896	1437011	1437011	1349130	18.97
0	1	1	3	5569231	1432541	1432541	1348978	19.59
0	1	1	4	4356879	1427025	1427025	1350207	19.93
0	1	1	5	7546897	1434816	1434816	1350276	19.41
0	1	1	6	1555626	1439757	1439757	1348371	20.47
0	1	1	7	3254879	1437289	1437289	1349781	19.18
0	1	1	8	9856214	1434264	1434264	1350103	19.62
0	1	1	9	1239876	1436755	1436755	1351024	19.95
0	1	0	0	4528957	1185246	1185246	1148972	23.36
0	1	0	1	3537898	1183800	1183800	1147840	22.65
0	1	0	2	4953896	1184932	1184932	1149429	22.82
0	1	0	3	5569231	1180449	1180449	1147104	22.29
0	1	0	4	4356879	1176968	1176968	1147307	23.77
0	1	0	5	7546897	1183594	1183594	1149114	23.31
0	1	0	6	1555626	1186486	1186486	1148177	22.5
0	1	0	7	3254879	1183328	1183328	1148371	23.06
0	1	0	8	9856214	1180582	1180582	1147900	23.39
0	1	0	9	1239876	1185049	1185049	1150374	23.99

Table A.21: FUS Results, rel.err=0.05

					FUS			
T_{PM}	A	B	#	seed	Step 3.3	Step 4.2	Final	cpu
1	0	0	0	4528957	1873136	1872963	1735118	33.29
1	0	0	1	3537898	1875889	1874441	1732593	32.23
1	0	0	2	4953896	1881150	1881150	1737872	31.04
1	0	0	3	5569231	1874974	1874974	1741552	30.46
1	0	0	4	4356879	1859665	1857894	1731418	33.93
1	0	0	5	7546897	1883143	1881984	1734277	32.09
1	0	0	6	1555626	1884744	1883982	1733445	31.62
1	0	0	7	3254879	1875439	1875439	1736611	31.79
1	0	0	8	9856214	1867810	1867773	1736339	32.87
1	0	0	9	1239876	1884174	1880885	1731875	32.69
1	0	1	0	4528957	2370793	2370793	2084695	24.34
1	0	1	1	3537898	2363170	2363170	2077012	24.75
1	0	1	2	4953896	2371813	2371813	2085339	25.01
1	0	1	3	5569231	2362445	2362445	2080803	24.34
1	0	1	4	4356879	2346788	2346788	2081910	24.74
1	0	1	5	7546897	2365645	2365645	2081181	25.82
1	0	1	6	1555626	2375380	2374797	2067944	25.75
1	0	1	7	3254879	2356826	2356826	2072488	25.74
1	0	1	8	9856214	2367797	2367797	2087064	24.61
1	0	1	9	1239876	2373410	2372610	2072530	25.69
1	1	1	0	4528957	2406570	2406570	2122164	18.05
1	1	1	1	3537898	2396820	2395899	2115800	17.63
1	1	1	2	4953896	2403190	2402603	2117038	17.48
1	1	1	3	5569231	2393264	2393264	2118506	17.25
1	1	1	4	4356879	2382948	2382603	2123058	18.55
1	1	1	5	7546897	2396536	2396057	2116270	17.74
1	1	1	6	1555626	2406932	2406932	2120059	17.74
1	1	1	7	3254879	2396433	2396433	2117412	17.55
1	1	1	8	9856214	2402036	2402036	2126929	17.49
1	1	1	9	1239876	2409930	2409930	2117646	18.13
1	1	0	0	4528957	1917541	1917183	1785455	21.25
1	1	0	1	3537898	1915384	1915384	1786384	21.56
1	1	0	2	4953896	1919074	1919074	1788444	20.72
1	1	0	3	5569231	1909813	1909813	1787475	20.59
1	1	0	4	4356879	1904748	1903977	1788403	22.45
1	1	0	5	7546897	1921059	1920089	1788253	21.33
1	1	0	6	1555626	1922896	1922896	1787245	20.84
1	1	0	7	3254879	1916241	1916241	1786506	21.11
1	1	0	8	9856214	1912807	1912807	1790384	21.19
1	1	0	9	1239876	1916310	1916310	1786382	21.14

Table A.22: FUS Results, rel.err=0.05

					FUS-test			
T_{PM}	A	B	#	seed	Step 3.3	Step 4.2	Final	cpu
0	0	0	0	4528957	1179946	1179946	1141505	28.94
0	0	0	1	3537898	1176161	1176161	1139813	30.31
0	0	0	2	4953896	1176984	1176984	1140030	29.18
0	0	0	3	5569231	1174181	1174181	1139419	28.5
0	0	0	4	4356879	1172396	1172396	1140841	28.99
0	0	0	5	7546897	1175322	1175322	1139595	30.72
0	0	0	6	1555626	1179919	1179919	1140176	29.67
0	0	0	7	3254879	1179433	1179433	1142280	28.49
0	0	0	8	9856214	1177167	1177167	1142858	30.21
0	0	0	9	1239876	1179786	1179786	1144046	30.71
0	0	1	0	4528957	1429584	1429584	1342170	23.26
0	0	1	1	3537898	1430373	1430373	1342153	21.05
0	0	1	2	4953896	1431929	1431929	1342483	22.44
0	0	1	3	5569231	1425849	1425849	1339950	21.45
0	0	1	4	4356879	1421242	1421242	1341571	21.88
0	0	1	5	7546897	1431521	1431492	1343309	21.03
0	0	1	6	1555626	1433754	1433754	1340738	22.21
0	0	1	7	3254879	1432348	1432348	1342634	21.67
0	0	1	8	9856214	1429285	1429285	1342457	23.12
0	0	1	9	1239876	1431710	1431710	1343940	22.24
0	1	1	0	4528957	1434533	1434533	1348316	19.48
0	1	1	1	3537898	1434950	1434950	1349942	19.27
0	1	1	2	4953896	1437011	1437011	1349130	18.99
0	1	1	3	5569231	1432541	1432541	1348978	19.63
0	1	1	4	4356879	1427025	1427025	1350207	19.88
0	1	1	5	7546897	1434816	1434816	1350276	19.35
0	1	1	6	1555626	1439757	1439757	1348371	20.6
0	1	1	7	3254879	1437289	1437289	1349781	19.12
0	1	1	8	9856214	1434264	1434264	1350103	19.42
0	1	1	9	1239876	1436755	1436755	1351024	19.77
0	1	0	0	4528957	1185246	1185246	1148972	23.44
0	1	0	1	3537898	1183800	1183800	1147840	22.53
0	1	0	2	4953896	1184932	1184932	1149429	22.98
0	1	0	3	5569231	1180449	1180449	1147104	22.37
0	1	0	4	4356879	1176968	1176968	1147307	23.5
0	1	0	5	7546897	1183594	1183594	1149114	23.08
0	1	0	6	1555626	1186486	1186486	1148177	22.35
0	1	0	7	3254879	1183328	1183328	1148371	23.12
0	1	0	8	9856214	1180582	1180582	1147900	23.3
0	1	0	9	1239876	1185049	1185049	1150374	23.64

Table A.23: FUS-test Results, rel.err=0.05

					FUS-test			
T_{PM}	A	B	#	seed	Step 3.3	Step 4.2	Final	cpu
1	0	0	0	4528957	1873136	1872963	1735118	33.36
1	0	0	1	3537898	1875889	1874441	1732593	32.2
1	0	0	2	4953896	1881150	1881150	1737872	31.07
1	0	0	3	5569231	1874974	1874974	1741552	30.42
1	0	0	4	4356879	1859665	1857894	1731418	33.87
1	0	0	5	7546897	1883143	1881984	1734277	32.02
1	0	0	6	1555626	1884744	1883982	1733445	31.39
1	0	0	7	3254879	1875439	1875439	1736611	31.86
1	0	0	8	9856214	1867810	1867773	1736339	32.58
1	0	0	9	1239876	1884174	1880885	1731875	32.73
1	0	1	0	4528957	2369138	2369138	2087883	26.36
1	0	1	1	3537898	2362780	2362780	2081043	24.86
1	0	1	2	4953896	2370321	2370321	2088203	26.98
1	0	1	3	5569231	2361307	2361307	2085515	24.45
1	0	1	4	4356879	2346081	2346081	2085703	24.74
1	0	1	5	7546897	2365645	2365645	2081181	25.67
1	0	1	6	1555626	2374526	2373943	2068781	27.7
1	0	1	7	3254879	2355843	2355843	2071943	27.26
1	0	1	8	9856214	2366584	2366584	2091784	24.61
1	0	1	9	1239876	2373410	2372610	2072530	25.64
1	1	1	0	4528957	2406442	2406442	2126041	18.33
1	1	1	1	3537898	2396743	2395821	2116939	17.82
1	1	1	2	4953896	2403120	2402532	2118083	17.65
1	1	1	3	5569231	2392748	2392748	2122377	17.37
1	1	1	4	4356879	2382887	2382542	2124338	18.49
1	1	1	5	7546897	2396486	2396020	2117521	17.98
1	1	1	6	1555626	2406468	2406468	2122750	17.89
1	1	1	7	3254879	2396403	2396403	2118204	17.68
1	1	1	8	9856214	2401492	2401492	2132811	17.72
1	1	1	9	1239876	2409930	2409930	2117646	18.11
1	1	0	0	4528957	1917541	1917183	1785455	21.37
1	1	0	1	3537898	1914395	1914395	1787748	21.74
1	1	0	2	4953896	1919074	1919074	1788444	20.65
1	1	0	3	5569231	1909813	1909813	1787475	20.54
1	1	0	4	4356879	1904674	1903903	1788201	22.51
1	1	0	5	7546897	1920995	1920025	1788111	21.71
1	1	0	6	1555626	1922896	1922896	1787245	20.83
1	1	0	7	3254879	1915922	1915922	1786911	21.36
1	1	0	8	9856214	1912475	1912475	1791341	21.15
1	1	0	9	1239876	1915677	1915677	1787479	21.17

Table A.24: FUS-test Results, rel.err=0.05

VITA

Sinan Gürel was born on January 16, 1977 in Kütahya, Turkey. He attended Ankara Fen Lisesi in 1992. He graduated from Bilkent University Industrial Engineering Department with degree of honor in 1999. He worked as a Maintenance Management System Specialist in implementation projects of MAXIMO, a world-wide used maintenance management system software, for two years. In 2001, he attended Industrial Engineering Department of Bilkent University as a Research Assistant. Since then, he has been working with Dr. M. Selim Aktürk on his graduate study.