TIME/COST TRADE-OFFS IN MACHINE SCHEDULING WITH CONTROLLABLE PROCESSING TIMES

A DISSERTATION 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
DOCTOR OF PHILOSOPHY

By
Sinan Gürel
January, 2008

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

Prof. Dr. M. Selim Aktürk (Supervisor)

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

Prof. Dr. Erdal Erel

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

Prof. Dr. 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 dissertation for the degree of doctor of philosophy.

Asst. Prof. Dr. Alper Şen

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

Asst. Prof. Dr. Hande Yaman

Approved for the Institute of Engineering and Science:

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

ABSTRACT

TIME/COST TRADE-OFFS IN MACHINE SCHEDULING WITH CONTROLLABLE PROCESSING TIMES

Sinan Gürel
Ph.D. in Industrial Engineering
Supervisor: Prof. Dr. M. Selim Aktürk
January, 2008

Processing time controllability is a critical aspect in scheduling decisions since most of the scheduling practice in industry allows controlling processing times. A very well known example is the computer numerically controlled (CNC) machines in flexible manufacturing systems. Selected processing times for a given set of jobs determine the manufacturing cost of the jobs and strongly affect their scheduling performance. Hence, when making processing time and scheduling decisions at the same time, one must consider both the manufacturing cost and the scheduling performance objectives. In this thesis, we have studied such bi-criteria scheduling problems in various scheduling environments including single, parallel and non-identical parallel machine environments. We have included some regular scheduling performance measures such as total weighted completion time and makespan. We have considered the convex manufacturing cost function of CNC turning operation. We have provided alternative methods to find efficient solutions in each problem. We have particularly focused on the single objective problems to get efficient solutions, called the ϵ-constraint approach. We have provided efficient formulations for the problems and shown useful properties which led us to develop fast heuristics to generate set of efficient solutions.

In this thesis, taking another point of view, we have also studied a conic quadratic reformulation of a machine-job assignment problem with controllable processing times. We have considered a convex compression cost function for each job and solved a profit maximization problem. The convexity of cost functions is a major source of difficulty in finding optimal integer solutions in this problem, but our strengthened conic reformulation has eliminated this difficulty. Our reformulation approach is sufficiently general so that it can also be applied to other mixed 0-1 optimization problems with separable convex cost functions.

Our computational results demonstrate that the proposed conic reformulation is very effective for solving the machine-job assignment problem with controllable processing times to optimality.

Finally, in this thesis, we have considered rescheduling with controllable processing times. In particular, we show that in contrast to fixed processing times, if we have the flexibility to control the processing times of the jobs, we can generate alternative reactive schedules in response to a disruption such as machine breakdown. We consider a non-identical parallel machining environment where processing times of the jobs are compressible at a certain cost which is a convex function of the compression on the processing time. When rescheduling, it is critical to catch up the initial schedule as soon as possible by reassigning the jobs to the machines and changing their processing times. On the other hand, one must keep the total cost of the jobs at minimum. We present alternative match-up scheduling problems dealing with this trade-off. We use the strong conic reformulation approach in solving these problems. We further provide fast heuristic algorithms.

Keywords: Scheduling, Controllable processing times, Manufacturing cost, Bicriteria optimization, Convex cost functions, Second-order cone programming, conic integer programming, Rescheduling, Match-up times.

ÖZET

KONTROL EDİLEBİLİR İŞLEM SÜRELERİYLE MAKİNE ÇİZELGELEMEDE MALİYET/ZAMAN İLİŞKİLERİ

Sinan Gürel
Endüstri Mühendisliği, Doktora
Tez Yöneticisi: Prof. Dr. M. Selim Aktürk
Ocak, 2008

Kontrol edilebilir işlem süreleri çizelgeleme kararları verilirken dikkate alınması gereken önemli bir özelliktir. Çünkü pek çok endüstri uygulaması, işlem sürelerinin kontrol edilebilmesine olanak sağlamaktadır. Buna en iyi bilinen örnek bilgisayar sayısal kontrollü (CNC) kesme makineleridir. Verilen bir iş kümesi için seçilen işlem süreleri toplam üretim maliyetini belirlediği gibi çizelgeleme performansını da önemli oranda etkiler. Bu yüzden işlem süreleri ve çizelgeleme kararlarını birlikte verirken hem toplam üretim maliyeti hem de çizelgeleme performans hedeflerini birlikte eniyilemek gerekir. Bu tezde, tek makine, paralel makine gibi değişik çizelgeleme ortamlarında bu çift hedefli problemler üzerinde çalıştık. Toplam iş bitim süresi ve maksimum iş bitim süresi gibi çizelgeleme performans kriterlerini ele aldık. Çalışmada özellikle CNC torna işlemleri için bilinen konveks maliyet fonksiyonunu kullandık. Ele aldığımız her problem için etkin çözüm bulmaya yarayan hızlı metodlar önerdik. Çalışmamızda özellikle tek hedefli problemler çözerek etkin çözüm bulmaya çalıştık. Bu yönteme literatürde ϵ-kısıt yaklaşımı denmektedir. Biz de bu çalışmada etkin problem formülasyonları önerdik ve bu formülasyonlar üzerinde gösterdiğimiz özellikleri kullanarak yaklaşık etkin çözümler üreten sezgisel metodlar geliştirdik.

Bu tezde, bir başka yaklaşımla, kontrol edilebilir işlem süreleriyle iş-makine atama problemi için yeni bir konik karesel formülasyon önerdik. Bu kısımda her iş için konveks bir sıkıştırma maliyeti fonksiyonu ele aldık ve kâr maksimizasyon problemi çözdük. Problemin çözümünü zorlaştıran temel nedenlerden biri maliyet fonksiyonunun konveks olmasıdır. Önerdiğimiz güçlendirilmiş formülasyonla bu zorluğu ortadan kaldırdık. Yaklaşımımız, ayrık konveks maliyet fonksiyonları içeren başka karışık 0-1 eniyileme problemlerinde de kullanılabilecek genel bir yaklaşımdır. Deneysel hesaplamalarımız önerdiğimiz formülasyonun iş-makine

atama problemlerinin eniyi çözümünde çok etkili olduğunu gösterdi.

Son olarak bu tezde, kontrol edilebilir işlem süreleriyle yeniden çizelgeleme üzerine çalıştık. Sabit işlem süreleriyle yeniden çizelgelemeden farklı olarak kontrol edilebilir işlem sürelerinin makine bozulması gibi aksaklıklar karşısında çok farklı alternatif çözümler üretmemize olanak sağladığını gösterdik. Farklı paralel makineler üzerinde konveks sıkıştırma maliyet fonksiyonu varlığında makinelerden birinin bir süre çalışamaması durumunda yeniden çizelgeleme problemi üzerinde çalıştık. Yeniden çizelgelemede amaç iş-makine atamalarını yeniden yaparak ve işlem sürelerini değiştirerek eski çizelgeyi en kısa zamanda yakalamaktır. Öte yandan toplam üretim maliyetini de enazlamak gerekir. Çelişen bu iki hedefi ele alan alternatif yeniden çizelgeleme problemleri önerdik. Bu problemleri güçlendirilmiş konik formülasyon yaklaşımını kullanarak çözdük. Ayrıca hızlı sezgisel tarama algoritmaları önerdik.

Anahtar sözcükler: Çizelgeleme, Kontrol edilebilir işlem süreleri, Konveks üretim maliyeti, İki hedefli eniyileme, İkinci derece konik programlama, Konik tamsayılı programlama, Yeniden çizelgeleme.

Dedicated to
Yeşim and Barış...

Acknowledgement

I would like to sincerely thank to my advisor Prof. M. Selim Aktürk for his valuable and perpetual guidance and encouragement throughout this study. His supervising with patience and interest made this thesis possible.

I am grateful to Assoc. Prof. Alper Atamtürk for kindly guiding me in preparation of this dissertation during my visit to University of California,Berkeley.

I gratefully acknowledge all the members of my committee who have given their time to read this manuscript and offered valuable advice.

I am especially indebted to my wife Yeşim for her love, encouragement and sacrifice which made this thesis possible. I can never thank her enough for taking care of Barış by herself during my visit to Berkeley and during her stay in Bitlis while completing her mandatory government service as a medical doctor.

I would like to thank to TÜBİTAK for providing the financial support for my Ph.D. study and my visit to Berkeley.

Contents

1 Introduction **1**

2 Literature Review **7**

2.1 Machining Parameters Selection 7

2.2 Scheduling with Controllable Processing Times: Time/Cost Trade-off 9

2.2.1 Single Machine Problems 11

2.2.2 Parallel Machine Scheduling Problems 14

2.3 Multi-objective Scheduling 16

2.4 Conic Mixed Integer Programming 18

2.5 Rescheduling 19

3 Single Machine Scheduling **24**

3.1 Problem Definition 26

3.2 Cost Index Based Approximation (CIBA) Method 35

3.3 Total Completion Time Problem 38

3.4 Numerical Example 43

3.5 Computational Results 46

3.6 Conclusion 53

4 Parallel Machine Scheduling 54

4.1 Problem Definition 55

4.2 Optimality Properties 58

4.3 A heuristic method to generate approximate efficient solutions . . 63

4.4 Numerical Example 67

4.5 Computational Analysis 71

4.6 Conclusions 77

5 Machine Job Allocation 78

5.1 Problem Definition 79

5.2 Single Machine Subproblem (P_m) 81

5.3 Cost Lower Bounds for a Partial Schedule 84

5.4 Initial Solution 88

5.5 B&B Algorithm 89

5.6 Beam Search Algorithm (BS) 96

5.7 Improvement Search Heuristic (ISH) 98

5.8 Recovering Beam Search (RBS) 101

5.9 Computational Results 102

5.10 Conclusion 110

6 Conic Quadratic Reformulation 112

6.1 Problem Definition 113

6.2 Conic Reformulations 115

6.2.1 Working with **epi**(**f**) 116

6.2.2 Strengthening the continuous relaxation 118

6.2.3 Conic quadratic representation 121

6.3 Computational Analysis 123

6.4 Conclusion 130

7 Match up Scheduling 131

7.1 Rescheduling with Controllable Processing Times: A Numerical Example 133

7.2 Scheduling Environment and Problem Definitions 138

7.2.1 Minimize Sum of Match up Times 139

7.2.2 Minimize Maximum of Match up Times 141

7.2.3 Minimize Total Manufacturing Cost Subject to a Bound on Sum of Match up Times 142

7.2.4 Minimize Total Manufacturing Cost Subject to a Bound on Maximum Match up Time 143

7.3 Strong Conic Quadratic Formulations for Cost Minimization Problems 144

7.4 Generating A Set of Approximately Efficient Solutions: Heuristic Approach 147

7.4.1 A Subproblem 148

7.4.2 Job Pool 149

7.4.3 1-move Improvement Search 151

7.4.4 2-swap Improvement Search 153

7.5 Computational Study 154

7.6 Conclusions 159

8 Conclusion **161**

8.1 Concluding Remarks 161

8.2 Future Research Directions 163

A Single Machining Operation Problem (SMOP) **176**

List of Figures

3.1 A typical manufacturing cost function for a turning operation. . . 27

3.2 An example set of efficient solutions 29

4.1 A set of efficient solutions for the numerical example 70

4.2 Behavior of R on different regions of the efficient frontier 75

5.1 B&B tree for the numerical example 94

6.1 Surfaces defined by inequalities (6.6) and (6.7). 118

6.2 Binary construction tree for Example 1. 123

7.1 Alternative Reactive Scheduling Approaches 136

7.2 Efficient Solution Set for Total Cost and Sum of Match-up Times Objectives 137

7.3 Efficient Solution Set for Total Cost and Minimum of Maximum Match- up Time Objectives 138

List of Tables

3.1 Specifications of the jobs in the numerical example 43

3.2 Schedules at Z_1 and Z_2 44

3.3 Results of the first 10 iterations by the CIBA method 45

3.4 Schedules generated by different methods when F_2=7.592 46

3.5 Experimental design factors 47

3.6 Technical coefficients of the cutting tools 48

3.7 Performance measures for the weighted case 49

3.8 Performance measures for the total completion time case 50

3.9 Comparison with the global optimal solutions 51

3.10 Comparison of the approximation algorithms for $\Delta = 0.01$ 52

4.1 Schedules at Z_1 and Z_2 68

4.2 Results of the first 7 iterations of MPJ algorithm 69

4.3 Schedules generated by different methods at iteration 1 70

4.4 Experimental Design Factors 72

4.5 Performance measures for different step size levels 73

4.6 Average performance measures for different N and M levels when $\Delta = 0.01$ 74

4.7 Comparison with the global optimal solutions 76

4.8 Comparison of the approximation algorithms 76

5.1 Job data for numerical example 93

5.2 Trial Results for Job Ordering Rules for Step 2 of B&B. 105

5.3 Eliminated and Traversed Tree Sizes 105

5.4 CPU Requirements (in seconds) for different lower bounding methods 106

5.5 Eliminated and Traversed Nodes at different K levels for $N = 20$ and $M = 4$ by LB_{LP} 107

5.6 Deviations from the optimum for IS, BS and RBS heuristics . . . 108

5.7 Deviations from the optimum for ISH algorithm 109

5.8 Average CPU time (sec.) requirements 109

5.9 Performances of Beam Search and Improvement Search Heuristics at different K levels 110

5.10 Performances of IS and ISH 110

6.1 Computational results for the quadratic case: $f(y) = ky^2$. 125

6.2 Alternative formulations for the cubic case: $f(y) = ky^3$. 126

6.3 Computational results for the cubic case: $f(y) = ky^3$. 127

6.4 Conic formulation size for the rational case: $f(y) = ky^{a/b}$. 128

6.5 Computational results for the general case: $f(y) = ky^{a/b}$. 129

7.1 Sum of Match-up Times 156

7.2 Maximum Match-up Time Results 157

7.3 Heuristic Algorithm Performance 159

Chapter 1

Introduction

Most of the studies in the machine scheduling literature assume fixed processing times. However, there are many industry applications where we can control the processing times. A well known example is the turning operation on CNC turning machines. On a CNC turning machine, we can control the processing time of an operation by setting the machining parameters such as the cutting speed and feed rate. For a turning operation, decreasing the processing time by increasing the cutting speed and/or feed rate results in more wear on the tool which implies increased tooling cost for the job. As a result, decreasing the processing time of a job usually requires incurring extra costs. In order to utilize the processing time controllability on a machine, we need to make appropriate processing time decisions which takes the manufacturing cost performance into account.

On the other hand, scheduling problems are extremely sensitive to the processing time data, so we need appropriate processing time decisions to improve the scheduling objectives. When considering a regular scheduling objective, one usually sets the processing time of each job as small as possible and then solves the scheduling problem. This approach only focuses on scheduling performance and ignores the manufacturing cost performance as we have to use more resource to achieve shorter processing times. Therefore, in order to make appropriate processing time and scheduling decisions, we need to investigate the existing

time/cost trade-off between manufacturing cost objective and the scheduling objective under consideration.

The existing CNC machine technology allows us to change the processing times very quickly by just changing few lines in the CNC programming code. Hence, on those machines we can easily execute the scheduling and process planning decisions which balance the manufacturing cost and scheduling performance as required by the decision maker.

In the scheduling literature, most of the studies have focused on problems with a single objective. However, in the real world, we usually face a number of objectives. Process planning or processing time decisions focus on minimizing the manufacturing cost, whereas in the scheduling decisions the main aim is to optimize a scheduling criterion. Usually these two decisions are made independently. Since there is a significant interaction between the schedule performance and cost, in this thesis, we propose models and algorithms that combine these decisions for different scheduling environments and scheduling performance measures.

In this study, we basically focus on CNC turning machines, so we have a well defined and realistic manufacturing cost function of processing time for each job. This manufacturing cost function is nonlinear and convex. In our analysis, we assume the case where the manufacturing cost function might be different for each job due to different operational and surface quality requirements, and its required cutting tool. However, all our results are applicable for the cases where there exists sublots of jobs which are identical. Although we specifically consider manufacturing cost function for the turning operation, our results apply to any problem with nonlinear convex processing cost functions.

In Chapters 3, 4 and 5, we first focus on finding efficient solutions for the objectives of total manufacturing cost and various scheduling performance measures in different machine environments. In order to find efficient solutions, one method we use is the ϵ-approach, i.e. solving a single objective problem after sending other objectives to the constraint set. We also propose heuristic algorithms which generate sets of approximate efficient solutions. We next study a strengthened conic quadratic reformulation for a machine-job assignment problem with controllable

processing times in Chapter 6. Finally, we propose some rescheduling problems under processing time controllability assumption in Chapter 7.

In Chapter 2, we give the literature review on related topics to this thesis. We first review the related literature on process planning problems for turning operation. We then give an extensive review on scheduling with controllable processing times and refer to the multi-objective scheduling literature. We next discuss the advances in second-order cone programming. Chapter 2 ends with a review of rescheduling studies.

In Chapter 3, we consider the situation where both total weighted completion time and cost performance are under consideration for a CNC turning machine. In order to find a set of efficient solutions for this bicriteria problem, we first present a mathematical model for the single objective problem which minimizes total manufacturing cost subject to a given upper bound on total weighted completion time objective. We derive optimality properties for the single objective problem. Then, by utilizing these properties, we propose a new heuristic method to generate a set of approximate efficient solutions. Our results show that by integrating the machine scheduling and process planning decisions, we can generate a set of alternative solutions for the decision maker so that significant time/cost gains can be achieved.

In Chapter 4, we consider identical parallel CNC turning machines on which we have two objectives to minimize: total completion time and total manufacturing cost. We deal with the problem of minimizing total manufacturing cost subject to a given limit on total completion time. This problem is more difficult than minimizing the sum of two objective functions which was usually done in the literature. For this problem, we propose an effective formulation which can be solved by commercial nonlinear programming solvers. Using this formulation, we also give useful properties for the problem which allowed us to develop an algorithm that can generate a large set of approximate efficient solutions in a short computation time.

In the current literature on the loading and scheduling problems of flexible manufacturing systems, the most popular performance measure is balancing the

workload (or minimizing the makespan). This is due to the fact that these systems require a very high investment cost so that the managers would like to fully utilize their capacity. In Chapter 5, we consider both the makespan and total manufacturing cost objectives at the same time for a flexible machining environment of non-identical parallel machines. We solve the problem of minimizing total manufacturing cost subject to a given bound on makespan. We give an exact solution method for the problem and develop several heuristic methods.

Another approach we have taken in analyzing the machine scheduling problems with controllable processing times is developing conic quadratic (second order conic) reformulations. In Chapter 6, we have focused on a machine-job assignment problem with controllable processing times arising in flexible manufacturing systems. In such systems one employs a host of non-identical machines each having different applicable machining power levels. Thus, each job has different cost and different processing time values on different machines.

Different than the analysis in Chapters 3- 5, in Chapter 6, we have considered the case that cost of a job on a machine is determined by the amount of compression on its processing time. We have studied the trade-off between increasing yield and cost of machining, which can be modeled as a nonlinear mixed 0-1 profit maximization problem. We reformulate the problem using a polynomial number of conic quadratic constraints. We construct strong conic reformulations by studying the convex hull description of appropriate mixed integer sets defined by nonlinear inequalities. Our results are applicable to many different problems from different areas such as finance and manufacturing.

As a final step of this thesis, we have considered processing time controllability in rescheduling problems. Controllable processing times is a critical factor to be considered in making reactive decisions against unexpected disruptions to a given schedule. Making processing time decisions simultaneously with scheduling decisions, such as sequencing, allocation, etc., usually complicates the problems. On the other hand, this enables generating alternative schedules with varying manufacturing cost and scheduling performance, hence brings flexibility in making reactive scheduling decisions. In this thesis, we next studied how rescheduling

and processing time decisions can be made at the same time to react against a machine breakdown on a given schedule.

In Chapter 7, we propose alternative rescheduling approaches for a given preschedule in non-identical parallel machine environment. We consider different rescheduling objectives to minimize. The first one is the total manufacturing cost for the jobs not yet started at the time of machine breakdown. The second objective is the sum of match-up times on the machines. Match-up time on a machine is the time point at which the new schedule catches up the preschedule. The third objective to minimize is the maximum match-up time for the new schedule. Since the cost objective and the match-up time related objectives conflict, in order to find efficient solutions, we consider the problems of minimizing total manufacturing cost subject to an upper bound on total match-up time and an upper bound on maximum match-up time. We give formulations for each of these problems. We show that cost minimization problems can be reformulated by using conic quadratic inequalities as shown in Chapter 6. This reformulation is important since it allows us to solve the practical size problems in very short CPU times, which is quite critical in rescheduling. The second approach is devising a heuristic algorithm which generates approximate efficient solutions for the cost and match-up time objectives based on the slope information of cost functions.

As we deal with different forms of manufacturing cost function and different scheduling decisions such as sequencing and allocation in different scheduling environments, we give the related notation at the beginning of each chapter which we believe will make this thesis more readable.

In Chapter 2, we give the related literature. In Chapter 3, we give the results for the total manufacturing cost and weighted completion time objectives in single machine. In Chapter 4, we extend the discussion to the identical parallel machine environment in which total completion time objective is under consideration. In Chapter 5, we explore the manufacturing cost minimization problem on non-identical parallel machine environment with a limit on makespan objective. We give a strong conic-quadratic reformulation for the machine job assignment problem in Chapter 6. In Chapter 7, we introduce rescheduling problems with

controllable processing times and give solution approaches. Finally, we give final remarks and future research directions in Chapter 8.

Chapter 2

Literature Review

In this chapter, we will first give a literature review on process planning decisions for the turning operation. Then, we will discuss on the scheduling literature with controllable processing times and time/cost trade-off. We will then mention the related work on multi-objective scheduling. We will next give a review of second order cone programming and conic mixed integer programming literature. We will end this chapter with rescheduling and match-up scheduling literature.

2.1 Machining Parameters Selection

Trade-off between the cutting parameters and the manufacturing cost or the surface quality of a turning operation have been studied extensively in the literature. Machining parameters selection problem dealing with this trade-off is a well known problem. On a CNC turning machine, increasing the cutting speed and/or feed rate decreases the processing time of an operation whereas it increases the tooling cost. The problem is to select the appropriate cutting speed and feed rate parameters for a given turning operation. For the turning operation, selected machining parameters must satisfy the surface roughness requirement for the part being machined and must take into account the maximum machine power that the machine can apply. These two constraints were defined by Bhattacharya et al.

[14]. The tool life equation, developed by Taylor [82], defines the relationship between cutting tool's life and the machining parameters. It is used to determine the tooling cost which occurs due to loss of tool life by a cutting operation. Also, we can use the tool life equation to define the tool life constraint when formulating problems in which there is a restriction that an operation must be performed within a predefined tool life.

In context of machining parameters selection problem, different objectives like minimizing production cost, maximizing output production rate or maximizing profit rate have been studied. Hitomi [46] discussed various mathematical models and solution methods for different objective functions of machine parameter selection problem for turning operation. Aktürk and Gürel [3] included maintenance cost along with tooling and operating costs in the objective function of machining parameters selection problem. Malakooti and Deviprasad [61] formulated machine parameter selection problem as a multiple objective decision making problem. Three conflicting objectives of minimizing total cost, minimizing production time and minimizing surface roughness were considered and a heuristic approach was discussed. They also gave a list of seminal studies in the machine parameter selection area.

Ermer and Kromordihardjo [30] suggested the combination of separable programming and geometric programming for the conversion of the machine parameter optimization model to a linear programming formulation. Gopalakrishnan and Al-Khayyal [33] provided a geometric programming based method to minimize machining and tooling costs. The method they provided was based on geometric programming and used the complementary slackness conditions to solve the problem. Choi and Bricker [22] discussed the effectiveness of a geometric programming model in machining optimization problems.

There are studies which combine machine level decisions such as tool loading and maintenance planning with process planning decisions in flexible machining environment. Lamond and Sodhi [57] considered the cutting speed selection and tool loading decisions on a single cutting machine so as to minimize total processing time. Sodhi et al. [81] considered determining the optimal processing speeds,

tool loading and part allocations on several flexible machines with finite capacity tool magazines where the objective is to minimize the makespan. Gürel and Aktürk [40] studied making processing time and preventive maintenance planning decisions simultaneously for a CNC turning machine.

Aktürk and Avcı [1] considered the tool life constraint in a geometric programming model which is given in Appendix A. They proved that either the tool life constraint or the surface roughness constraint must be tight at the optimal solution. Kayan and Aktürk [54] later showed that only the surface roughness constraint must be tight at the optimal solution. This result is very important for our analysis in this thesis. The tightness condition for the surface roughness constraint enables us to express the machining cost as a function of processing time. Then, when we make a processing time decision for a scheduling problem, we can easily determine the corresponding cutting speed and feed rate and corresponding machining (manufacturing) cost. We know that this cost function is convex and this property will be very important in our analysis. Kayan and Aktürk [54] also provided a mechanism to determine upper and lower bounds for the processing time of a turning operation. When we consider manufacturing cost and scheduling performance measure simultaneously, process planning decisions and scheduling decisions affect each other. Combining the process planning and various scheduling decisions for CNC turning machines is an important contribution of this thesis. Next, we will give a literature review on scheduling with controllable processing times.

2.2 Scheduling with Controllable Processing Times: Time/Cost Trade-off

In this section, we will give a review of the literature on scheduling with controllable processing times and time/cost trade-off. A recently published extensive survey on this topic is given by Shabtay and Steiner [79]. The initial work on scheduling with controllable processing times dates back to 1980, however, most of the studies in the literature were published in recent years. When discussing

the studies in the literature it is useful to use a similar notation with the recent surveys given by Hoogeven [48] and Shabtay and Steiner [79]. Solving a scheduling problem with controllable processing times requires:

(i) specifying a feasible schedule σ for the jobs, and

(ii) specifying a processing time vector $\mathbf{p}$.

We denote processing time of job j by p_j and corresponding manufacturing cost by $f_j(p_j)$. Then, total manufacturing cost is $F1(\mathbf{p}) = \sum_j f_j(p_j)$ and the scheduling performance measure is $F2(\sigma)$. Then, the following scheduling problems arise:

$P1$: to minimize the total cost, that is $F1(\mathbf{p}) + F2(\sigma)$;

$P2$: to minimize $F1(\mathbf{p})$ under the constraint $F2(\sigma) \leq F$;

$P3$: to minimize $F2(\sigma)$ under the constraint $F1(\mathbf{p}) \leq C$;

$P4$: to identify the efficient frontier for $(F1(\mathbf{p}), F2(\sigma))$.

In the following, we will review the results that have been obtained for the $P1-P4$ versions of different bi-criteria scheduling problems. To state the processing time controllability, we will be using the acronym "*contr*" in the second field of $\alpha|\beta|\gamma$ notation used by Graham et al. [34]. The first field (α) describes the machine environment, the second field (β) describes the processing characteristics or constraints and the third field (γ) gives the objective to be minimized. We will be using the acronym "*lin*" in the second field for the linear cost function problems, and "*conv*" for the convex cost function problems.

Studies assuming controllable processing times mostly deal with two objectives as given above. The first one is the cost of operating the jobs on the machines which is considered as manufacturing cost, or compression cost, or processing cost in different studies in the literature. The second objective is a scheduling performance measure.

There are few survey papers which classify and review the studies in scheduling with controllable processing times area. The recent one is by Shabtay and Steiner [79]. Hoogeven [48] reviews multi-objective scheduling literature and gives a short review on the studies with controllable processing times in multi-objective

scheduling. Another survey on scheduling with controllable processing times is by Nowicki and Zdrzalka [69] which reviews the results achieved till 1990. In this section, we particularly give the results which are related to the various machining environments and scheduling performance measures that we have considered in this thesis.

2.2.1 Single Machine Problems

Firstly, we give the results for the $1|contr|C_{max}, \sum_j f_j(p_j)$ problem. Van Wassenhove and Baker [87] show that $P1 - P4$ versions of the problem $1|contr|g_{max}, \sum_j f_j(p_j)$ are solvable in polynomial time under the assumption that $g_j(t) = w_j t$ for all $j = 1, ..., n$ where g_j is a function of completion time of job j. When w_j's are equal for all j, this result implies that $P1 - P4$ versions of the problem $1|contr, lin|C_{max}, \sum_j f_j(p_j)$ are solvable in polynomial time. Hoogeveen and Woeginger [47] extend these results to the piecewise linear f_j's. Chen et al. [20] consider the problem $1|contr, r_j|C_{max}, \sum_j f_j(p_j)$ where processing times are discretely controllable and the jobs have release dates. They show that $P1$ version of the problem is $\mathcal{NP}$-hard for the discretely controllable case but it is solvable in polynomial time for the continuously controllable case.

For the convex cost function case, solving $P2$ version of the problem $1|contr, conv|C_{max}, \sum_j f_j(p_j)$ is equivalent to solving nonlinear resource allocation problem discussed by Bretthauer and Shetty [15]. They give the optimality properties and a solution method for the problem. Kaspi and Shabtay [53] use a convex nonlinear resource consumption function of processing time for each job. They consider the problem $1|contr, conv, r_j|C_{max}, \sum_j f_j(p_j)$ subject to limited total resource consumption ($P3$). They also consider minimizing total resource consumption subject to limited makespan ($P2$) when all release dates are the same. Kayan and Aktürk [54] consider the problem of $1|contr, conv|C_{max}, \sum_j f_j(p_j)$ on a CNC turning machine and provide methods to solve $P2$ version of the problem. The convex manufacturing cost function $f_j(p_j)$ for the turning operation is known from the process planning literature. In Chapter 5, we will discuss $P2$ version of the problem $1|contr, conv, r_j|C_{max}, \sum_j f_j(p_j)$ in a single CNC turning machine

in detail. Solution method for this problem will lead us in developing solution methods for parallel machine problems.

The second problem to consider is the $1|contr, lin|\sum C_j, \sum_j f_j(p)j)$ where C_j is the completion time of job j. Vickson [89] shows that $P1$ version of the problem is solvable in polynomial time in his work which initiated the area of scheduling with controllable processing times. He observed that compressing the processing time of the job at position j by δ decreases the total completion time by $(n-j+1)\times\delta$, where n is the number of jobs, and increases the cost by $c_j \times \delta$ where c_j is the slope of the linear compression cost function. This observation yielded the conclusion that in an optimal solution a job is either fully compressed or not compressed at all. This decision depends on the position of the job. Then, for each job, we can determine the cost to occur at each position. This allows to formulate the problem as an assignment problem which is easy to solve. Chen et al. [20] show that the discrete controllable case for the same problem is also solvable in polynomial time. Ng et al. [66] additionally consider batching and controllable setup times for the same objective in $P1$ and $P3$ versions of the problem. Ruiz Diaz and French [74] develop an enumerative algorithm for the $P4$ version of the problem and noted that the efficient frontier in general is not convex. In Chapter 3, we will give the results on $P2$ and $P4$ versions of the problem $1|contr, conv|\sum C_j, \sum_j f_j(p_j)$ on a single CNC turning machine. Different than the studies above, we deal with nonlinear convex manufacturing cost function which leads to a non-convex mixed integer nonlinear programming formulation for the $P2$ version of the problem. We will give optimality properties for the problem and a mathematical formulation whose nonlinear programming relaxation gives an integer solution. We will also present an algorithm which generates an approximate efficient solution set in a very short computation time.

Next problem to consider is $1|contr, lin|\sum w_j C_j, \sum_j f_j(p_j)$ where w_j is the weight of job j. Vickson [88] studied $P1$ version of the problem. He proposed several heuristics and a branch and bound algorithm to solve the problem. He conjectured the $\mathcal{NP}$-hardness of the problem. Wan et al. [91] and Hoogeveen and Woeginger [47] showed that the problem is $\mathcal{NP}$-hard in the ordinary sense.

Janiak et al. [50] showed that the problem is a positive half-product minimization problem and presented fully polynomial time approximation schemes for the problem. Shabtay and Kaspi [78] considered a nonlinear relationship between processing times and resource consumption. They considered $P3$ version of the problem which is the problem of scheduling jobs on a single machine to minimize total weighted flow time subject to limited resource. They presented optimality properties for the problem and showed the cases solvable in polynomial time. They also proposed a dynamic programming algorithm. In this thesis, we deal with $P2$ version of the problem for which we will prove some optimality properties in Chapter 3. We will also give an efficient mathematical model for the problem as in the $1|contr, conv|\sum C_j, \sum_j f_j(p_j)$ case. We will give an algorithm to generate an approximate efficient solution set.

There are results for other single machine scheduling problems in the literature regarding due date-related objectives. Vickson [89] considers $P1$ version of the problem $1|contr, lin|T_{max}$ where T_{max} is the maximum tardiness. He gives a polynomial time algorithm to solve the problem. If there is the restriction that a job can either be fully compressed or not compressed at all then the problem becomes $\mathcal{NP}$-hard in the ordinary sense. Shabtay [76] gives polynomial time algorithms for minimizing maximum lateness subject to limited single or two-resource consumption constraint. He again considers nonlinear resource consumption function. Using the same resource consumption function, Yedidsion et al. [94] provide a polynomial algorithm which constructs the trade-off curve between maximal lateness and total resource consumption objectives. Chen et al. [20] show that the discretely controllable case is $\mathcal{NP}$-hard. They also show that the problem $1|contr, lin|\sum w_jU_j + \sum_j f_j(p_j)$ with discretely controllable processing times is $\mathcal{NP}$-hard where U_j is 1 if job j is late and 0 otherwise. They further show that for the common due date case, the problem $1|contr, lin|\sum \alpha E_j + \beta T_j, \sum_j f_j(p_j)$ is solvable in polynomial time where E_j is the earliness of job j. Daniels and Sarin [25] provide some theoretical properties that would aid developing the trade-off curve ($P4$) between number of tardy jobs ($1|contr|\sum U_j$) and the total amount of allocated resource. Panwalkar and Rajagopalan [72] study $P1$ version of the problem $1|contr|\sum E_j + T_j, \sum_j f_j(p_j)$ where the processing times, the common

due date and the job sequence are to be determined. They provide a polynomial time algorithm for the problem. Janiak and Kovalyov [49] consider minimizing weighted compression cost subject to deadlines given for the jobs. For the continuous resource case, they show that the problem is solvable in polynomial time but for the discrete case they prove that it is $\mathcal{NP}$-hard.

In Chapter 3 we present our results on the $P2$ versions of the problems $1|contr, conv|\sum C_j, \sum_j f_j(p_j)$ and $1|contr, conv|\sum w_j C_j, \sum_j f_j(p_j)$. We proposed a heuristic algorithm which generates a set approximate efficient solutions for these problems. To the best of our knowledge, convex cost function case is not studied yet except the resource allocation studies of Kaspi and Shabtay [53], Shabtay and Kaspi [78] and Shabtay [76]. They assumed a nonlinear convex resource consumption function $r_j = w_j p_j^k$ where p_j is the processing time of job j, r_j is the amount of resource allocated to job j, w_j is a job specific constant and k is a negative exponent which is same for all jobs. This resource consumption function corresponds to a special case of our tooling cost term in the manufacturing cost function such that all jobs require the same cutting tool type. However, usually this is not the case in CNC machining, each job may require different cutting tool type and each job could have a different nonlinear manufacturing cost function due to different operational and surface quality requirements. Moreover, we have a lower bound on the processing time of each job due to surface quality and CNC machine power requirements. Therefore, their analysis do not apply directly to the problems which we consider in this thesis. Next, we will give the results on parallel machine problems.

2.2.2 Parallel Machine Scheduling Problems

In the literature, we see that most of the attention in parallel machine scheduling problems with controllable processing times is given for the C_{max} objective. The earliest and the best known work is by Trick [84]. He considered the problem $R_m|contr, lin|C_{max}, \sum_j f_j(p_j)$ where R_m stands for non-identical parallel machines. For the $P1$ version of the problem he proposed an approximation algorithm with $2.816 + \epsilon$ worst case performance. For the same problem Shmoys and

Tardos [80] proposed a 2-approximation algorithm. Trick [84] also considered minimizing total compression cost subject to machine capacity constraints. He assumed that each machine can have different capacities but it corresponds to $P2$ version in our problem classification. He showed the $\mathcal{NP}$-hardness of the problem and gave a mathematical formulation which corresponded to a network structure. He observed some optimality properties and proposed a heuristic algorithm for the problem. Differently, in Chapter 5, we consider a nonlinear convex manufacturing cost function for each job. Moreover, we propose a branch and bound (B&B) algorithm for the problem. We then give a recovering beam search algorithm which can be implemented for the instances where the B&B algorithm is not computationally efficient. We also propose an improvement search algorithm which can be used to improve any given feasible schedule for the problem. Our results for the non-identical machine problem also apply for the identical machine problems.

Jansen and Mastrolilli [51] give polynomial time approximation algorithms for $P1$, $P2$ and $P3$ versions of the problem $P_m|contr, lin|C_{max}, \sum_j f_j(p_j)$ where P_m stands for the identical parallel machines. They also provided exact algorithms for the preemptive versions. Mastrolilli [62] considers $P2$ version of the problem $P_m|r_j, contr|C_{max}, \sum_j f_j(p_j)$. He shows that when the preemption allowed the problem is solvable in polynomial time but it is $\mathcal{NP}$-hard for the non-preemptive case.

Daniels et al. [26] study the $P3$ version of the C_{max} problem on parallel machines. There is limited resource to be allocated to the machines and the resource allocated to a machine determines the processing times of the jobs on that machine. They consider static and dynamic resource allocation cases. They give theoretical results, algorithms and complexity analysis for the problem. The only paper that deals with $P4$ version of the problem $P_m|contr, lin|C_{max}, \sum_j f_j(p_j)$ is by Nowicki and Zdrzalka [68] which gives a polynomial time algorithm to generate the set of Pareto-optimal points when preemption is allowed.

Another problem considered in the literature is $R_m|contr|\sum C_j, \sum_j f_j(p_j)$. The first study dealing with controllable processing times on parallel machines is

by Alidaee and Ahmadian [6] who solved $P1$ version of the problem by extending the approach given by Vickson [88]. They considered linear processing cost functions and their approach was extended to nonlinear convex cost function case by Cheng et al. [21]. Shabtay and Kaspi [77] considered minimizing the total completion time subject to a maximal resource constraint. In Chapter 4, we consider the total completion time objective in identical parallel machines.

Chen [19] studied $P1$ version of the problems $P_m|contr, lin|\sum w_jC_j, \sum_j f_j(p_j)$ and $P_m|contr, lin|\sum w_jU_j, \sum_j f_j(p_j)$. He showed the $\mathcal{NP}$-hardness of both problems and proposed branch and bound algorithms for both discretely and continuously controllable cases. Zhang et al. [95] developed a 3/2-approximation algorithm for the $P1$ version of the $R_m|contr, lin|\sum w_jC_j, \sum_j f_j(p_j)$ problem, which they showed to be $\mathcal{NP}$-hard. Alidaee and Ahmadian [6] showed that $P1$ version of the problem is solvable in polynomial time for the problem $R_m|contr, lin, d = d_j|\sum E_j + T_j, \sum_j f_j(p_j)$ where $d = d_j$ implies all jobs have same due date. In the next section, we give a short review on multi-objective scheduling.

2.3 Multi-objective Scheduling

Quality of a schedule can be evaluated in different dimensions. A production schedule which is good in terms of catching due dates and achieving customer satisfaction may be causing high inventory levels in the system. In the scheduling literature, since 1980's single objective problems were considered. Hoogeven [48] gave a review on multi-objective scheduling. Some of those studies consider fixed processing times. Gupta and Ruiz-Torres [37] considered the objectives of minimizing total flow time and minimizing total number of tardy jobs simultaneously and proposed heuristic algorithms to generate efficient solutions. Gupta and Ho [36] provided solution methods for the problem of minimizing makespan subject to minimum flow time for two parallel machines. Cao et al. [17] considered the machine selection and scheduling decisions together in order to minimize the sum of machine cost and job tardiness. Alagöz and Azizoğlu [5] studied a

problem with the objectives of minimizing total completion time and minimizing number of disrupted jobs in a rescheduling environment. In this thesis, we study bi-criteria scheduling problems in different machining environments with the first objective being the total manufacturing cost, and the second objective being a scheduling performance measure.

One of the methods to solve bi-criteria problems in the literature is representing one of the objectives as a constraint and optimizing over the second objective. By this way, we can search over the different values to generate a set of discrete efficient points to approximate the efficient frontier. Therefore, in this thesis, we consider the problem of minimizing total manufacturing cost objective for a given upper limit on different scheduling objectives. This method known as the ϵ-constraint approach as discussed in T'kindt and Billaut [83] has been used widely in the literature, because it is easy to use in an interactive algorithm. Moreover, the decision maker can interactively specify and modify the bounds and analyze the influence of these modifications on the final solution.

In multi-objective optimization problems, approximation quality of the generated efficient set is important to the decision maker. In the literature, there are different approximation quality evaluation metrics developed. These metrics are useful for comparing different algorithms. Tuyttens et al. [86] consider the classical linear assignment problem with two objectives for which they employ a multi-objective simulated annealing method, and provide two metrics to compare the results with an exact efficient set. Wu and Azarm [92] propose some quality evaluation measures to compare efficient sets generated by different multi-objective optimization methods. A review and discussion on existing metrics is available in Zitzler et al. [98]. In Chapters 3, 4 and 5 we employ different metrics to compare the approximation quality of our proposed methods against solutions obtained by commercial solvers. In the next section, we discuss conic mixed integer programming.

2.4 Conic Mixed Integer Programming

A second-order cone $\mathcal{Q}^m$ in m dimension is defined as:

$$\mathcal{Q}^m = \{x = (x_1, \ldots, x_m) \in \mathbb{R}^m : \|x_1, x_2, \ldots, x_{m-1}\| \leq x_m\}$$

where $\|.\|$ refers to the standard Euclidean norm.

A second-order conic programming (SOCP) problem is an optimization problem which has a linear objective and a set of second order conic constraints as can be written below:

$$\min_x \{c^T x : \|A_i x - b_i\| \leq p_i^T x - q_i, i = 1, \ldots, k\}$$

where A_i are matrices of the same row dimension as x, b_i are vectors of the same dimensions as the column dimensions of the matrices A_i, p_i are vectors of the same dimension as x and q_i are reals. In conic mixed integer programming, a subset of x_i's is assumed to be integer in the problem form given above.

As a part of the progress observed in conic optimization in last two decades, it was shown that SOCP problems can be solved by using polynomial interior point algorithms as given by Nesterov and Nemirovski [65]. It was also shown that convex optimization problems with norms, fractional quadratic functions, hyperbolic functions can be formulated and solved as SOCP problems. Our analysis in Chapter 6 is strongly motivated by the recent advances in conic programming, in particular, second order conic (or conic quadratic) programming discussed by Ben-Tal and Nemirovski [13] and Alizadeh and Goldfarb [7]. An extensive review on SOCP was given by Alizadeh and Goldfarb [7]. Due to the advances in SOCP theory and its potential use, stable SOCP solvers were provided by the commercial optimization software vendors in their recent versions (e.g. ILOG, MOSEK, XPRESS-MP). Availability of efficient SOCP algorithms implemented in branch-and-bound solvers led us to explore the effectiveness of using conic quadratic constraints to formulate the machine-job assignment problem with controllable processing times as a conic mixed-integer program in Chapter 6.

Research on strengthening conic integer programming formulations is so far fairly limited. Çezik and Iyengar [18] describe Chvátal-Gomory and disjunctive

cuts for conic integer programs. Atamtürk and Narayanan [8] give nonlinear conic mixed-integer rounding cuts for conic mixed-integer programming. Whereas these earlier papers develop cuts for general conic mixed-integer programs, in Chapter 6 we exploit the structure of machine-job assignment problem with controllable processing times in order to derive strong conic formulations.

Two recent papers study a similar structure and propose alternative solution approaches to the one given in Chapter 6. Frangioni and Gentile [32] describe an interesting cutting plane procedure based on linear outer approximations of the perspective of convex functions and apply it to the unit commitment problem with separable quadratic cost. Günlük et al. [35] give problem-specific linear and nonlinear cuts for a quadratic cost facility location problem. Although in Chapter 6 we apply conic strengthening to the machine-job assignment problem with controllable processing times, our results are sufficiently general so that they can also be applied to other mixed 0-1 optimization problems with separable convex objective, including those studied in these two recent papers. In the next section, we review the rescheduling literature.

2.5 Rescheduling

Rescheduling has received considerable attention in the scheduling literature. Many different problems, methods and approaches have been presented on rescheduling. Vieira et al. [90] give an extensive discussion on rescheduling theory. Aytug et al. [9] discuss the types and effects of uncertainties that can be faced in the execution of schedules. Both studies include a review of the reactive/predictive approaches in the scheduling literature. Reactive and predictive scheduling approaches have also been studied in the project scheduling literature. Herroelen and Leus [44, 45] review approaches for robust project scheduling and reactive project scheduling under uncertainty.

In the literature, different rescheduling environments were considered with finite or infinite set of jobs in different machine environments. Some studies assume

that all information is given (deterministic) and some of them assume information is uncertain (stochastic). Different approaches were considered to solve rescheduling problems such as dynamic (on-line) scheduling with dispatching rules to apply when jobs arrive or disrupting events occur. In dynamic approach, we make decisions based on the current state of the manufacturing system. Another strategy is predictive scheduling which aims to generate initial schedules in a way to reduce the negative effects of possible disruptions. Repairing the schedule or generating a completely new schedule from scratch are other alternatives. Rescheduling after each disruption or rescheduling periodically at a given frequency can be the alternative policies for making the decision of when to reschedule. There are different methods of rescheduling like right-shifting, partial rescheduling or complete regeneration. Partial rescheduling avoids rescheduling all jobs from scratch since changing the schedule of jobs frequently causes system nervousness.

Leon et al. [59] compare partial scheduling with complete rescheduling and right-shifting methods. Nof and Grant [67] discuss the right-shift and complete rescheduling methods. Kutanoğlu and Sabuncuoğlu [56] compare several dispatching rules of rescheduling in a flexible manufacturing environment. A recent study by Hall and Potts [42] is about rescheduling due to arrival of new set of jobs to a machine. They consider scheduling cost (lateness, total completion time) and disruption cost (change in jobs' positions, change in completion times) simultaneously. Minimizing schedule cost subject to a limit on disruption and minimizing schedule cost plus disruption cost models were considered. They give algorithms and complexity analysis for different models. In Chapter 7, we consider rescheduling in parallel machines after a breakdown occurs on one of the machines.

In the rescheduling literature, to the best of our knowledge controllable processing times have not been considered except two studies. The first one is by Türkcan [85] in which reactive scheduling decisions on non-identical parallel CNC machines were considered. They consider the sum of earliness and tardiness and the manufacturing costs along with a stability measure defined as the absolute difference between completion times of jobs in the new schedule and the initial

schedule. They provide a heuristic approach to find the new schedule after a disruption such as machine breakdown or new job job arrival. In the second work, Yang [93] considers the arrival of new jobs in a single machine rescheduling problem where the objective is to minimize the total cost after rescheduling. The objective includes schedule disruption cost which is a function of deviations on start times of the jobs. Another cost term is the time compression cost which comes from the compression on the processing times. The third term in the objective is the cost of scheduling performance measures such as total completion time and weighted tardiness. Proposed solution approach is a heuristic algorithm based on very large scale neighborhood search. Processing time controllability allows alternative approaches in rescheduling after a disruption occurs, so in Chapter 7, we deal with rescheduling with controllable processing times and present alternative rescheduling problems and solution approaches.

In the literature, there are studies which propose methods to generate robust schedules with respect to disruptions. A predictive approach proposed by Mehta and Uzsoy [63] is including inserted idle times in a job shop schedule so as to reduce the impact of disruptions. They first find a job sequence by using the shifting bottleneck algorithm and then apply a heuristic approach to insert idle times into the schedule. Similarly O'Donovan et al. [70] describe methods for constructing robust schedules with respect to machine breakdown for a single machine environment. The objective is to minimize the expected deviation in completion times due to the breakdown. Their experiments show that inserted idle time approach improve schedule robustness with little impact on other performance measures. For the maximum tardiness problem, Jensen [52] proposed minimizing maximum lateness instead, so that achieved schedule has improved rescheduling performance due to the idle time left at the end of the schedule. In a recent work, Leus and Herroelen [60] considered minimizing expected weighted deviation between actual and planned job starting times in a single machine scheduling problem with a common deadline for all jobs. They find the optimal job sequence and the optimal length of idle time following each job in the schedule. However, if we insert idle times in a schedule but no disruption occurs, then the jobs will finish too early and machining capacity will not be fully-utilized.

In case of scheduling with controllable processing times, if a disruption occurs then we have the flexibility to repair the schedule by compressing the processing times of remaining jobs. Inserting idle times implies additional compression on the processing times of the jobs and hence requires higher manufacturing cost. In Chapter 7 we propose rescheduling approaches which repair a parallel machine schedule after a machine breakdown by assigning new processing times to the jobs and making new machine-job assignment decisions.

Match-up scheduling is a rescheduling approach which aims to catch up the preschedule within a certain time after disruption occurs. Match-up scheduling examples in the literature are given by Bean et al. [12] and Aktürk and Görgülü [2]. Those two studies propose heuristic approaches to find match-up times. Match-up scheduling studies in the literature assume planned idle time periods in preschedules so that the disruption can be absorbed. With controllable processing times, even if the preschedule is a non-delay schedule, i.e. no idle time exists in the schedule, we can still reschedule to match up with the preschedule. It may be possible to match up soon after the disruption occurs by compressing the jobs which immediately succeed the breakdown. However, catching up sooner implies incurring more compression cost. Therefore, when we consider the match-up time and the cost objectives at the same time, it is critical to make appropriate rescheduling and processing time decisions. In Chapter 7, we present match-up scheduling problems with controllable processing times which demonstrate the trade off between match-up time and manufacturing cost objectives. We give exact and efficient solution approaches for the considered problems.

Azizoğlu and Alagöz [10] study a rescheduling problem on parallel machines where an unavailability period occurs on one of the machines. They consider the total flow time objective and a stability measure of number of jobs reassigned to another machine than its original machine in the preschedule. They show that efficient solution set for the considered objectives can be generated in polynomial time. Curry and Peters [24] consider a dynamic scheduling environment on parallel machines where the arrival of new jobs to the system is handled. They observed that by considering a machine reassignment cost in the scheduling

problem in addition to the scheduling performance measure of stepwise increasing tardiness cost, solutions with few reassignments could be achieved without a statistically significant change in tardiness cost. Lee et al. [58] studied a two-machine scheduling environment where they penalized machine-job reassignments made after a disruption as transportation cost in their model. They considered transportation cost with different scheduling performance measures and changes in completion times of jobs. Number of reassignments is an important stability measure, which we have not considered in our study, but in Chapter 7 our models and heuristic algorithm can be easily extended to include a limit on number of reassigned jobs. In the next chapter, we discuss the total manufacturing cost and total weighted completion time objectives in a single machine environment.

Chapter 3

Time/Cost Trade-offs in Single Machine Scheduling

In this chapter, we study the trade-off between the total manufacturing cost and the total weighted completion time objectives in single machine environment. The aim is to find efficient solutions for these two objectives. We use the ϵ-approach and formulate a single objective problem to find efficient solutions. We solve the problem of minimizing total manufacturing cost subject to a given bound on total weighted completion time. In Section 3.1, we give the problem definition and propose a mathematical model to find an efficient solution for a given total weighted completion time level. We give optimality properties for the model. Based on these properties, we develop an approximate efficient solution set generating method which is presented in Section 3.2. Then, in Section 3.3, we discuss an alternative model for the total completion time problem. In Section 3.4, we give a numerical example. Next, in Section 3.5 we provide the computational results on the performance of proposed methods. Finally, we conclude the chapter in Section 3.6.

The notation used throughout the chapter is as follows:

Decision variables:

p_i: processing time of job i.

X_{ij}: binary variable to state if job i precedes job j in the sequence.

v_i : cutting speed for operation i (fpm).

f_i : feed rate for operation i (ipr).

U_i : usage rate of the required cutting tool to process operation i.

Parameters:

p_i^l: processing time lower bound for job i.

p_i^u: processing time level that gives minimum manufacturing cost for job i.

w_i: weight of job i.

$f_i(p_i)$: manufacturing cost function of processing time for job i.

α, β, γ : speed, feed, depth of cut exponents for the required cutting tool of job i.

C_i^{TL} : Taylor's tool life constant for the required cutting tool of job i.

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

C_s, g, h, l : specific coefficients of the surface roughness constraint of job i.

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

C_{t_i} : cost of cutting tool required to process job i (\$/tool).

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

d_i : depth of cut for job i (in).

H : maximum available machine power (hp).

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

S_i : maximum allowable surface roughness for job i (μin).

3.1 Problem Definition

We have N jobs to be processed, and each job corresponds to a metal cutting operation that will be performed by a given cutting tool on a single CNC turning machine. Each job differs in terms of its manufacturing properties such as diameter, length, depth of cut and maximum allowable surface roughness and its cutting tool, and a positive weight which shows its importance relative to the other jobs. The CNC turning machine can process one job at a time. We also assume that setup time and the tool change times are negligible. We have two objectives, minimizing the total manufacturing cost of jobs and minimizing their total weighted completion time. Therefore, we have to determine a job sequence and the corresponding processing times simultaneously. In order to solve this bicriteria problem, we have to consider process planning and scheduling problems simultaneously. One way to integrate these two decision making problems is through the proper selection of job processing times. Assuming a single pass operation, the processing time of job i on a CNC turning machine can be calculated as follows:

$$p_i = \frac{\pi D_i L_i}{12 v_i f_i}$$

On the other hand, the tool usage rate of a job, U_i, is simply the ratio of its processing time to the tool life. If we use extended Taylor's tool life equation, T_i, to describe the tool life then

$$U_i = \frac{p_i}{T_i} = \frac{(\pi D_i L_i)/(12 v_i f_i)}{C_i^{TL}/(v_i^{\alpha} f_i^{\beta} d^{\gamma})}$$

The most commonly used objective function for the manufacturing cost of job i is the sum of the operating and tooling costs. Operating cost of job i is the cost of running the machine for p_i. We assume that C_o is constant and independent of selected machining parameters. Tooling cost for job i is the cost of its tool usage U_i. The optimum machining parameters of the cutting speed (v_i) and the feed rate (f_i) for each job can be found by solving machining conditions optimization problem subject to the tool life, surface roughness and machine power constraints as discussed in Appendix A. Appendix A gives the geometric programming model for the problem and the optimality properties (Theorems A.1, A.2) which were

shown by Aktürk and Avcı [1] and Kayan and Aktürk [54], respectively. Using Theorem A.2, the manufacturing cost of job i can be expressed as a function of p_i as follows:

$$f_i(p_i) = C_o p_i + C_{t_i} U_i = C_o p_i + C_{t_i} \frac{d_i^{\gamma}}{C_i^{TL}} \left(\frac{\pi D_i L_i}{12} \right)^{\left(\frac{\alpha h - \beta g}{h-g}\right)} \left(\frac{C_s d_i^l}{S_i} \right)^{\left(\frac{\alpha-\beta}{h-g}\right)} p_i^{\left(\frac{(1-\alpha)h-(1-\beta)g}{h-g}\right)}$$

Furthermore, Kayan and Aktürk [54] showed that the nonlinear manufacturing constraints that limit the allowable ranges of the processing times can be replaced by a linear bound of $p_i^l \leq p_i \leq p_i^u$ for each job i when there is a regular scheduling performance measure, such as makespan or total completion time. For the determination of p_i^l and p_i^u values, we refer to Kayan and Aktürk [54]. A typical manufacturing cost function behavior for a job is given in Figure 3.1. Since jobs have different manufacturing properties, they will have different nonlinear convex manufacturing cost functions and different bounds on their processing times.

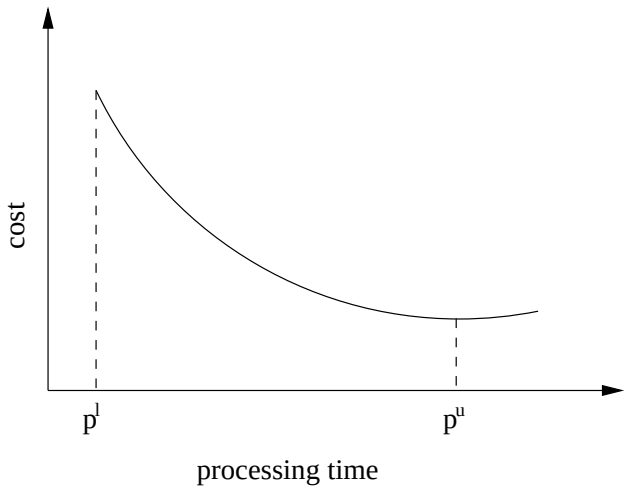

Figure 3.1: A typical manufacturing cost function for a turning operation.

The mathematical model for the problem is as below:

$$\min \; F_1 : \sum_{i=1}^{N} f_i(p_i) = \sum_{i=1}^{N} (C_o p_i + C_{t_i} U_i)$$

$$\min \; F_2 : \sum_{j=1}^{N} \sum_{i \neq j}^{N} w_j p_i X_{ij} + \sum_{j=1}^{N} w_j p_j$$

$$\text{s.t.} \qquad X_{ij} + X_{ji} = 1 \qquad i = 1, \ldots, N,\ j = i+1, \ldots, N \tag{3.1}$$

$$X_{ij} + X_{jk} + X_{ki} \geq 1 \qquad i, j, k = 1, \ldots, N,\ i \neq j \neq k \tag{3.2}$$

$$p_i^l \leq p_i \leq p_i^u \qquad i = 1, \ldots, N \tag{3.3}$$

$$X_{ij} \in \{0, 1\} \qquad i, j = 1, \ldots, N\ i \neq j \tag{3.4}$$

In the mixed integer nonlinear programming (MINLP) model above, the first objective function (F_1) is the total manufacturing cost. The second objective function (F_2) is the total weighted completion time. Constraint set (3.1) is the precedence constraints to ensure that two jobs cannot precede each other at the same time. Constraint set (3.2) satisfies the triangular inequality among the jobs such that if job j precedes job i and job k precedes job j then job k precedes job i. We have constraint set (3.3) that sets the upper and lower bounds on the processing time of each job.

For the weighted completion time problem, the minimum value is attained when we set the processing times to their lower bounds, p_i^l. On the other hand, the manufacturing cost decreases when we increase the processing times, and the minimum manufacturing cost is attained at p_i^u for each job i. That means if we increase the processing time of a job, the manufacturing cost decreases but completion time of the job itself and all the following jobs increase. Therefore, we cannot minimize both objectives F_1 and F_2 at the same time, and hence the overall problem is to generate an efficient solution set for the decision maker. A solution x $(F_1(x), F_2(x))$ is said to be efficient with respect to the given bicriteria if there does not exist another solution y $(F_1(y), F_2(y))$ such that $F_1(y) \leq F_1(x)$ and $F_2(y) \leq F_2(x)$ with at least one holding as a strict inequality. The following lemma states that there are infinitely many efficient solutions for the problem.

Lemma 3.1. *The efficient solution set for the problem includes infinitely many points.*

Proof. This is due to the fact that the processing times of jobs are continuous and can take any value satisfying $p_i^l \leq p_i \leq p_i^u$. If we slightly decrease the processing time of any job i that will increase the total manufacturing cost (as shown in

Figure 3.1), but at the same time it will decrease the total weighted completion time. Hence, there are infinitely many possible F_2 (or F_1) levels for the problem and we can find infinitely many efficient solutions. Furthermore, efficient frontier can be represented as a continuous function on a (F_1, F_2) plot. □

In Figure 3.2, an example for a set of efficient solutions is given. Solution Z_1 is the ideal solution for F_2 where $F_2(Z_1) = K^l$ where the superscript l implies that K^l is achieved by setting $p_i = p_i^l$ for each job i and applying the weighted shortest processing time first (WSPT) rule by Smith [1956]. According to the WSPT rule the jobs are ordered in the decreasing order of w_i/p_i to minimize total weighted completion time. At Z_1, total manufacturing cost is $F_1(Z_1) = \sum_{i=1}^{N} f_i(p_i^l)$. On the other hand, solution Z_2 is the ideal solution for F_1 where $F_1(Z_2) = \sum_{i=1}^{N} f_i(p_i^u)$ and it is achieved by setting $p_i = p_i^u$ for each job i and jobs are ordered by the WSPT rule. At Z_2, total weighted completion time is $F_2(Z_2) = K^u$, where the superscript u implies that the solution is achieved where all jobs are machined at processing time upper bounds.

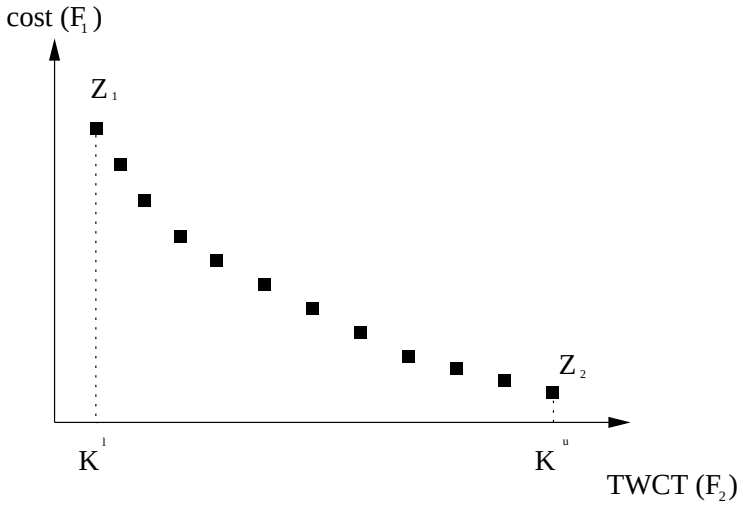

Figure 3.2: An example set of efficient solutions

In order to find a set of efficient solutions other than Z_1 and Z_2, we can consider F_2 as a constraint as in the formulation below and solve the resulting

Single Criterion Problem (SCP) for different values of K.

$$\begin{aligned} \min\ & F_1 : \sum_{i=1}^{N} f_i(p_i) \\ & \text{s.t.} \sum_{j=1}^{N}\sum_{i=1}^{N} w_j p_i X_{ij} + \sum_{j=1}^{N} w_j p_j \leq K \\ & \text{and } (3.1), (3.2), (3.3), (3.4) \end{aligned} \tag{3.5}$$

Constraint (3.5) guarantees that the total weighted completion time (F_2) of the schedule is less than or equal to a predefined value K. We can solve this model by the MINLP solvers like GAMS/BARON [16]. To generate an efficient solution set of n points between Z_1 and Z_2 we can employ the following algorithm denoted as the SCP-based method.

The SCP-based method finds a set of efficient points by solving the SCP model iteratively, so we investigated the SCP model and found some useful properties for the problem.

Lemma 3.2. *When $K \leq K^u$, constraint* (3.5) *on F_2 must be tight at the optimal solution. This implies that the optimal schedule must satisfy the WSPT rule.*

Proof. When $K > K^u$, total manufacturing cost can be minimized by setting all jobs to their minimum cost processing times (p^u) and ordering them by the WSPT rule. In this case, constraint (3.5) can be loose at optimality. When $K \leq K^u$, if constraint (3.5) is loose, then by increasing the processing time of a job, we can decrease the total manufacturing cost of the schedule. Therefore, a solution cannot be optimal if constraint (3.5) is loose. As a consequence of this result we can also state that the WSPT rule must be satisfied by an optimal schedule. Otherwise, we can reorder the jobs and find a better solution which violates the optimality. □

We proved that an optimal solution for the SCP must satisfy constraint (3.5) as an equality and it must also satisfy the WSPT rule. The next property for the problem is about the non-cycling constraint set (3.2).

Lemma 3.3. *The non-cycling constraints, constraint set* (3.2)*, are redundant for the SCP.*

Proof. Consider the SCP model without constraint set (3.2). We can easily see that Lemma 3.2 still holds for the new problem, otherwise we can improve the solution by resequencing the jobs and increasing the processing times. Lemma 3.2 states that the optimal solution must satisfy the WSPT rule which implies that the optimal solution cannot have cycles and always satisfies constraint set (3.2). □

By using this result we can eliminate constraint set (3.2) when solving the SCP model in SCP-based method. Since constraint set (3.2) is defined for each job triple, removing it reduces most of the constraints in the model so that MINLP solvers can solve the problem more efficiently. From this point on, SCP will denote the single criterion problem without constraint set (3.2). We next consider the relaxation of the problem where the integrality assumption of X_{ij}'s is relaxed. Relaxation results a nonlinear programming (NLP) model for which we can state the following two properties.

Lemma 3.4. *For the relaxed SCP, in a local optimal solution, if* $w_i/p_i > w_j/p_j$ *for any job pair* (i, j)*, then job* i *must precede job* j*, i.e. the solution must have* $X_{ij} = 1$*. This implies that a local optimal solution to the relaxed SCP must have binary* X_{ij}*'s except the cases including jobs with equal* w_i/p_i *ratios.*

Proof. Suppose that there is a local optimal solution S which has non-integer X_{ij} values. We will show that by modifying S in a way to achieve an integer solution, we can improve F_2.

Consider a job pair (i, j) in S for which $w_i/p_i > w_j/p_j$ holds. Suppose that the precedence variables for the job pair (i, j) are as follows: $X_{ij} = \lambda,\ X_{ji} = 1 - \lambda$, where $0 \leq \lambda < 1$. Then, $\sum w_k C_k$ for S is as below:

$F_2(S) = \Phi + w_i \times p_j \times (1 - \lambda) + w_j \times p_i \times \lambda$, where Φ is a constant value.

If we form a new solution S' from S by setting $X_{ij} = 1$ and $X_{ji} = 0$, we get: $F_2(S') = \Phi + w_j \times p_i$.

Obviously, $F_2(S') < F_2(S)$ and S' is a feasible solution to the SCP. This proves that a solution S with non-integer X_{ij} variables cannot be a local optimal solution. □

Considering Lemma 3.2 and the arguments above together, we conclude that in a local optimal solution to the relaxed problem all X_{ij} variables are binary. However, to be rigorous, we must also point out that in case $w_i/p_i = w_j/p_j$ for some job pair (i, j), we may have alternative optimal solutions where X_{ij} and X_{ji} are non-integer. This is because when $w_i/p_i = w_j/p_j$, $X_{ij} = \lambda$ and $X_{ji} = 1 - \lambda$, whatever value λ takes such that $0 \leq \lambda \leq 1$, $\sum w_i C_i$ calculated by the mathematical model remains the same. In practice, it is highly unlikely to face this situation since in our case p_i's are controllable variables that take real values.

Lemma 3.4 is an extremely important result to reduce the computational burden since we do not need MINLP solvers to solve the SCP. The problem can be solved by an NLP solver. However, NLP solvers can only guarantee to achieve local optimal solutions. From nonlinear programming theory we know that if the objective function of a problem is convex and the feasible region for the problem is a convex set, then a local optimum is a global optimum. Therefore, we investigated if the feasible region for the relaxed SCP is a convex set or not to see if NLP solvers could give us the global optimum.

Lemma 3.5. *The feasible region for the relaxed SCP is not a convex set, i.e. NLP solvers cannot guarantee global optimality for the problem.*

Proof. Consider two jobs i_1 and i_2 such that i_1 immediately precedes i_2, $X_{i_1 i_2} = 1$, in a solution called A_1. Let's suppose that $p_{i_1} = s_1$ and $p_{i_2} = s_2$ with weights w_1 and w_2, respectively. We also have $\frac{w_1}{s_1} > \frac{w_2}{s_2}$. $F_2(A_1) = Q + w_2 s_1 + w_1 s_1 + w_2 s_2 = K$, where Q is a constant. Next, consider another solution A_2 which is identical to A_1

except that i_1 and i_2 were pairwise interchanged and processing times were set to q_1 and q_2, respectively, and $\frac{w_2}{q_2} > \frac{w_1}{q_1}$ holds. $F_2(A_2) = Q + w_1q_2 + w_1q_1 + w_2q_2 = K$.

Now consider a convex combination of A_1 and A_2 as the solution $A = \lambda A_1 + (1-\lambda)A_2$. At A, $X_{i_1i_2} = \lambda$, $X_{i_2i_1} = 1-\lambda$, $p_{i_1} = \lambda s_1 + (1-\lambda)q_1$ and $p_{i_2} = \lambda s_2 + (1-\lambda)q_2$. Then,

$$\begin{aligned}
F_2(A) &= Q + w_1X_{i_2i_1}p_{i_2} + w_2X_{i_1i_2}p_{i_1} + w_1p_{i_1} + w_2p_{i_2} \\
&= Q + (w_1(1-\lambda) + w_2)(\lambda s_2 + (1-\lambda)q_2) \\
&\quad +(w_2\lambda + w_1)(\lambda s_1 + (1-\lambda)q_1) \\
&= Q + \lambda(s_1(w_1 + \lambda w_2) + s_2(w_2 + (1-\lambda)w_1)) \\
&\quad +(1-\lambda)(q_1(w_1 + \lambda w_2) + q_2(w_2 + (1-\lambda)w_1) \\
&= Q + \lambda(s_1w_1 + s_2w_2 + \lambda s_1w_2 + (1-\lambda)s_2w_1) \\
&\quad +(1-\lambda)(q_1w_1 + q_2w_2 + \lambda q_1w_2 + (1-\lambda)q_2w_1) \\
&> Q + \lambda(K-Q) + (1-\lambda)(K-Q) \\
&> K
\end{aligned}$$

since $s_2w_1 > s_1w_2$ and $q_1w_2 > q_2w_1$. This shows that the feasible region for the relaxed SCP is not a convex set. □

Lemma 3.5 implies the potential existence of multiple local optimal solutions for the relaxed SCP. Although it does not prove $\mathcal{NP}$-hardness of the SCP, we know that in general, computing a global minimum in a non-convex NLP is an $\mathcal{NP}$-hard problem due to Murty and Kabadi [64].

Lemma 3.5 is a direct consequence of the nonlinear terms p_iX_{ij} in constraint (3.5), so that the MINLP solvers may terminate with an integer local optimal solution. In order to find the global optimum for the SCP model, we propose the following linearized single criterion problem (LSCP) model below. In this model, we replace the constraint (3.5) with a set of linear constraints (3.6)-(3.10) and the term p_iX_{ij} is replaced with the variable Y_{ij}, where M is a large positive number.

$$\min \; F_1 : \sum_{i=1}^{N} f_i(p_i)$$

$$\text{s.t.} \sum_{j=1}^{N} \sum_{i=1}^{N} w_j Y_{ij} + \sum_{j=1}^{N} w_j p_j \leq K \tag{3.6}$$

$$Y_{ij} \geq p_i + (X_{ij} - 1)M \qquad i, j = 1, \ldots, N, \;\; i \neq j \tag{3.7}$$

$$Y_{ij} \leq p_i + (1 - X_{ij})M \qquad i, j = 1, \ldots, N, \;\; i \neq j \tag{3.8}$$

$$Y_{ij} \leq M X_{ij} \qquad i, j = 1, \ldots, N, \;\; i \neq j \tag{3.9}$$

$$Y_{ij} \geq 0 \qquad i, j = 1, \ldots, N, \;\; i \neq j \tag{3.10}$$

$$\text{and } (3.1), (3.3), (3.4)$$

In this model, constraint (3.6) is the constraint on F_2. Constraint sets (3.7)-(3.10) assure that if $X_{ij} = 0$ then $Y_{ij} = 0$ and if $X_{ij} = 1$ then $Y_{ij} = p_i$, so that $Y_{ij} = p_i X_{ij}$ always holds. Unfortunately, the computational performance of this linearization is very poor because Lemma 3.4 no longer holds so we cannot relax the integrality constraint of X_{ij}. Therefore, we have to use computationally less efficient GAMS/BARON solver instead of GAMS/MINOS.

In this section, we defined the problem and proposed the SCP-based method that can utilize commercial NLP solvers to solve the SCP and generate an approximation for the efficient frontier for the problem. We also gave the LSCP model which can be solved to global optimum by the MINLP solvers. However, since the MINLP and NLP solvers are not yet widely used and they are not as much computationally efficient as LP solvers, we also aimed to develop an effective approximation method to find a set of efficient points. In the next section, we define a heuristic method to approximate the efficient frontier for the problem.

3.2 Cost Index Based Approximation (CIBA) Method

In Section 3.1, we showed some optimality properties and simplifying characteristics for the SCP. In this section, we further present another very important optimality property in Lemma 3.6. This property will be helpful to design a heuristic method. This property basically tells us that if a solution is locally optimal then we cannot improve the total manufacturing cost by only changing the processing times of the jobs. It can also be explained as follows: for a given job sequence there is a unique optimal processing time vector p that minimizes the total manufacturing cost for a given total weighted completion time level K. This property is as follows:

Lemma 3.6. *For any job pair i, j, a local optimal solution must satisfy the following conditions:*
i. If $p_i > p_i^l$ and $p_j > p_j^l$ then $\frac{\partial f_i(p_i)}{\partial p_i}\frac{1}{W_i} = \frac{\partial f_j(p_j)}{\partial p_j}\frac{1}{W_j}$,
where $W_i = w_i + \sum_{k=1}^{N} X_{ik}w_k$ (i.e. the sum of the weights of job i and jobs that job i precedes)
ii. If $p_i = p_i^l$ and $p_j > p_j^l$ then $\frac{\partial f_i(p_i)}{\partial p_i}\frac{1}{W_i} \geq \frac{\partial f_j(p_j)}{\partial p_j}\frac{1}{W_j}$.

Proof. Suppose that $\frac{\partial f_i(p_i)}{\partial p_i}\frac{1}{W_i} < \frac{\partial f_j(p_j)}{\partial p_j}\frac{1}{W_j}$ for some i,j. Then,

$$\lim_{\Delta p\to 0}\left(\frac{f_i(p_i+\Delta p)-f_i(p_i)}{W_i\Delta p} - \frac{f_j(p_j)-f_j(p_j-\frac{W_i}{W_j}\Delta p)}{W_j\frac{W_i}{W_j}\Delta p}\right) < 0,$$

$$\lim_{\Delta p\to 0}\left(\frac{f_i(p_i+\Delta p)-f_i(p_i)+f_j(p_j)-f_j(p_j-\frac{W_i}{W_j}\Delta p)}{\Delta p}\right) < 0.$$

Then, $\exists\ \Delta p > 0$ s.t. $f_i(p_i+\Delta p) - f_i(p_i) + f_j(p_j) - f_j(p_j - \frac{W_i}{W_j}\Delta p) < 0$, which means the current solution can be improved without violating the total weighted completion time constraint. This proves that if there exists a job pair that does not satisfy the given conditions, then the solution is not locally optimal. □

The heuristic approach (CIBA) starts with the solution Z_1 (Figure 3.2) and generates new approximate efficient points by using the information in

Lemma 3.6. After taking the solution Z_1, by slightly increasing the processing times of the jobs one at a time at each iteration, we decrease F_1 while increasing F_2. The critical issue is which job to choose to perturb at each iteration so that the achieved decrease in F_1 over the increase in F_2 is at the maximum (i.e. the 'biggest bang for the buck'). To make the most appropriate choice, we propose a new cost index r_i for each job i, such that

$$r_i = \frac{\partial f_i(p_i)}{\partial p_i} \frac{1}{W_i}.$$

It is the estimated manufacturing cost change per unit total weighted completion time loss to be achieved by increasing the processing time of job i. Note that this index definition comes from Lemma 3.6. Next, we choose the job with the minimum r_i value. It is important to note that $f_i(p_i)$ is decreasing and $r_i < 0$ when $p_i^l \leq p_i < p_i^u$ for all i. Then, the processing time of the selected job is increased by a predefined Δ amount. This may result in a schedule that violates Lemma 3.2. Then, we check if the WSPT rule is violated or not and if so we reorder the jobs according to the WSPT rule. After reordering, F_2^{new} and F_1^{new} are calculated. For the updated and re-sequenced jobs, their r_i's are updated. This job selection and update process is applied until F_2^{new} reaches to K^u. Each schedule achieved at the end of an iteration is kept as an approximate efficient solution. By keeping Δ as small as possible we can achieve solutions which are more close to the optimality property defined by the Lemma 3.6. Having discussed the basic approach, the proposed cost index based approximation (CIBA) method is given below:

Step 1. Find the non-dominated solutions Z_1 and Z_2.

Step 2. Start with the solution Z_1, set $F_1^{new} = F_1(Z_1)$ and $F_2^{new} = F_2(Z_1)$. While $F_2^{new} < K^u$ do the following:

Step 2.1. Select the job m with the minimum r_m. If there are more than one such jobs, select the last one in the sequence.

Step 2.2. Set $p_m = p_m + \Delta$.

Step 2.3. If the WSPT rule is violated, re-sequence the jobs by the WSPT rule.

Step 2.4. Update r_i indices for job m and for all other jobs whose position in sequence is changed in Step 2.3.

Step 2.5. Update $F_1^{new} = F_1^{new} - [f_m(p_m) - f_m(p_m + \Delta)]$ and recalculate F_2^{new}.

Step 2.6. If $F_2^{new} < K^u$, report the current schedule with F_1^{new} and F_2^{new} as an approximate efficient solution.

In each iteration of the CIBA method, we want to improve total manufacturing cost by slightly losing from the total weighted completion time. Since we want to minimize both criteria at the same time, we always prefer to update the job with the maximum manufacturing cost gain per unit increase in the total weighted completion time (Step 2.1). This is due to Lemma 3.6 which states the optimality conditions on p_i's. At the end of each iteration, the schedule achieved is reported as an approximate efficient solution. An important property that holds for the solution set generated by the CIBA is the following:

Lemma 3.7. *Each iteration of the CIBA method, generates a new approximate efficient solution.*

Proof. As discussed earlier, we have a nonlinear convex manufacturing cost function and the processing times can take any real value between $p_i^l \leq p_i \leq p_i^u$. At each iteration of the CIBA method, the processing time of a selected job is increased by a Δ amount, that means the total manufacturing cost is strictly decreasing in each iteration as shown in Figure 3.1. Moreover, the total completion time strictly increases even if the job sequence changes. Therefore, in each iteration we generate a new approximate efficient solution that cannot dominate previously generated solutions. □

The solution quality of the CIBA method depends on the selected Δ value. Since we are making decisions based on $\frac{\partial f_i(p_i)}{\partial p_i}$'s, if we choose a small Δ value, we get a better approximation to an efficient solution. Furthermore, a smaller Δ leads to more solution points to be generated which implies a better approximation of the efficient frontier.

3.3 Total Completion Time Problem

All the models, properties and algorithms that we have given above also apply for the total completion time problem, which is a special case where the weights of jobs are equal. Instead, we present a new model for the total completion time problem. In this section, we proved that this new model holds the same properties as the previous one. Moreover, we performed a set of trial runs and showed that the new model is computationally more efficient in terms of the CPU times in solving the total completion time problem. In this model, we introduce a new binary decision variable X_{ij} to control if job i is assigned to position j in the sequence. The new formulation for the SCP is as below:

$$\min \; F_1 : \sum_{i=1}^{N} f_i(p_i)$$

$$\text{s.t.} \sum_{i=1}^{N}\sum_{j=1}^{N} \quad (N-j+1)X_{ij}p_i \leq K \tag{3.11}$$

$$\sum_{j=1}^{N} X_{ij} = 1 \qquad i = 1,\ldots,N \tag{3.12}$$

$$\tag{3.13}$$

$$\sum_{i=1}^{N} X_{ij} = 1 \qquad j = 1,\ldots,N \tag{3.14}$$

$$\tag{3.15}$$

$$p_i^l \leq p_i \leq p_i^u \qquad i = 1,\ldots,N \tag{3.16}$$

$$X_{ij} \in \{0,1\} \; i,j = 1,\ldots,N \tag{3.17}$$

In the model above, the objective function corresponds to the total manufacturing cost. Constraint (3.11) gives the total completion time. Constraints (3.12) and (3.14) are the assignment constraints which guarantee that each job is assigned to a position in the schedule and each position has a job assigned. Constraints (3.16) and (3.17) are same as in the previous model.

We could have used a similar model with the assignment variables X_{ij} to solve the weighted total completion time problem. In that case, we will not be able to use the property stated in Lemma 3.4, and hence the relaxed SCP model cannot be solved by the NLP solvers. It turns out that the way we model the problem affects the solver to be used and the computational requirements. We now check the characteristics of this new formulation as we did for the previous model before.

Lemma 3.8. *When $K \leq K^u$, the total completion time constraint* (3.11) *must be tight at the optimal solution. This implies that the optimal schedule must satisfy the shortest processing time first (SPT) rule.*

The proof for Lemma 3.8 is similar to the proof of Lemma 3.2, so we skip it due to the space limitations. We next consider the NLP relaxation of the new SCP formulation and look for the optimality conditions for the relaxed problem.

Lemma 3.9. *For the relaxed SCP, in a local optimal solution, if $p_{i_1} < p_{i_2}$ for any job pair (i_1, i_2), then job i_1 must be assigned to an earlier position than i_2. This implies that a local optimal solution to the relaxed SCP must have binary X_{ij}'s except the cases that include jobs with equal processing times.*

Proof. Suppose that we have a solution S for the relaxed problem where we consider two jobs i_1 and i_2 with processing times $p_{i_1} < p_{i_2}$. Further consider two positions j_1 and j_2 in the schedule such that $j_1 < j_2$. Suppose that assignment variables in S are as follows:
$X_{i_1 j_1} = \rho_1$ and $X_{i_2 j_1} = \rho_2$,
$X_{i_1 j_2} = \xi_1$ and $X_{i_2 j_2} = \xi_2$, where $\rho_1,\ \rho_2,\ \xi_1$ and ξ_2 are positive.
Since we relaxed the integrality constraint, a solution to the relaxed problem may contain non-binary X_{ij} values. This means, the same job could be allocated to multiple positions in the schedule, which is infeasible for our original problem.

Total completion time value calculated for S is as below:

$$\begin{aligned} F_2(S) &= \sum_{i=1}^{N}\sum_{j=1}^{N}(N-j+1)X_{ij}p_i \\ &= \Phi + (N-j_1+1)[\rho_1 p_{i_1} + \rho_2 p_{i_2}] + (N-j_2+1)[\xi_1 p_{i_1} + \xi_2 p_{i_2}] \\ &= \Phi + [(N-j_1+1)\rho_1 + (N-j_2+1)\xi_1]p_{i_1} + \\ &\quad [(N-j_1+1)\rho_2 + (N-j_2+1)\xi_2]p_{i_2} \end{aligned}$$

where Φ is a constant value. Suppose that without changing the processing times, we change the assignment variables to get a new solution S'. Setting $\delta = \min(\xi_1, \rho_2)$, new values for the assignment variables are as follows:
$X_{i_1 j_1} = \rho_1 + \delta$, $X_{i_2 j_1} = \rho_2 - \delta$,
$X_{i_1 j_2} = \xi_1 - \delta$ and $X_{i_2 j_2} = \xi_2 + \delta$.
By this arrangement we reallocate these two jobs to positions j_1 and j_2 such that we increase job i_1's ratio in the preceding position j_1 without disturbing the assignment constraints.

Total completion time of the solution after this arrangement is:

$$\begin{aligned} F_2(S') &= \Phi + [(N-j_1+1)(\rho_1+\delta) + (N-j_2+1)(\xi_1-\delta)]p_{i_1} \\ &+ [(N-j_1+1)(\rho_2-\delta) + (N-j_2+1)(\xi_2+\delta)]p_{i_2} \end{aligned}$$

Then, $F_2(S') - F_2(S) = \delta(j_2 - j_1)(p_{i_1} - p_{i_2}) < 0$, because $j_2 > j_1$ and $p_{i_1} < p_{i_2}$.

This proves that there is always an integer optimal solution for the relaxed problem. □

According to Lemma 3.9, a local optimal solution must have integer X_{ij}'s. Although for the case where $p_{i_1} = p_{i_2}$, we could have alternative non-integer local optimal solutions. As in the previous formulation for the weighted case we do not need a MINLP solver to solve the relaxed problem, and an NLP solver would be sufficient.

Lemma 3.10. *The feasible region for the relaxed SCP is not a convex set.*

Proof. Consider two jobs i_1 and i_2 in a schedule called A_1. They are assigned at positions k and $k+1$, respectively. Their processing times are $p_{i_1} = s_1$ and $p_{i_2} = s_2$ where $s_1 < s_2$.
$F_2(A_1) = Q + (N-k+1)s_1 + (N-k)s_2 = K$, where Q is a constant.

Consider another schedule A_2 which is identical to A_1 except that job i_1 is at position $k+1$ and i_2 is at position k with processing times $p_{i_1} = q_1$ and $p_{i_2} = q_2$ where $q_2 < q_1$. $F_2(A_2) = Q + (N-k+1)q_2 + (N-k)q_1 = K$.

Next, define a point A which is a convex combination of A_1 and A_2 as follows: $A = \lambda A_1 + (1-\lambda)A_2$ where $0 < \lambda < 1$. At point A, $p_{i_1} = \lambda s_1 + (1-\lambda)q_1$ and $p_{i_2} = \lambda s_2 + (1-\lambda)q_2$. Also, $X_{i_1 k} = \lambda$, $X_{i_1(k+1)} = (1-\lambda)$, $X_{i_2 k} = (1-\lambda)$ and $X_{i_2(k+1)} = \lambda$.

$$\begin{aligned} F_2(A) &= Q + [(N-k+1)\lambda + (N-k)(1-\lambda)]p_{i_1} \\ &+ [(N-k+1)(1-\lambda) + (N-k)\lambda]p_{i_2} \\ &= Q + \lambda^2[(N-k+1)s_1 + (N-k)s_2] \\ &+ (1-\lambda)^2[(N-k+1)q_2 + (N-k)q_1] \\ &+ \lambda(1-\lambda)[(N-k+1)q_1 + (N-k)q_2] \\ &+ \lambda(1-\lambda)[(N-k+1)s_2 + (N-k)s_1] \\ &> K \end{aligned}$$

since $s_1 < s_2$ and $q_2 < q_1$.

This example shows that the feasible region for the problem is not convex and this implies that a local optimum found by an NLP solver may not be the global optimum. The complexity discussion for the weighted case in Section 3.1 holds for this case, too. □

The SCP model for the total completion time case also includes nonlinear terms in constraint (3.11). We can linearize constraint (3.11) in the model by replacing the nonlinear term $p_i X_{ij}$ in constraint (3.11) with a variable Y_{ij} and

adding constraints (3.7)-(3.10) to the model as we did in Section 3.1. By this way we can achieve the LSCP model for the total completion time case and solve it to global optimum by using the MINLP solver GAMS/BARON.

Next, we give another property analogous to Lemma 3.6.

Lemma 3.11. *For any job pair i, j a local optimal solution must satisfy the following conditions:*
i. If $p_i > p_i^l$ and $p_j > p_j^l$ then $\frac{\partial f_i(p_i)}{\partial p_i}\frac{1}{n_i} = \frac{\partial f_j(p_j)}{\partial p_j}\frac{1}{n_j}$, where $n_i = \sum_{j=1}^{N} X_{ij}(N-j+1)$.
ii. If $p_i = p_i^l$ and $p_j > p_j^l$ then $\frac{\partial f_i(p_i)}{\partial p_i}\frac{1}{n_i} \geq \frac{\partial f_j(p_j)}{\partial p_j}\frac{1}{n_j}$.

Proof. Suppose that in a solution $\frac{\partial f_i(p_i)}{\partial p_i}\frac{1}{n_i} < \frac{\partial f_j(p_j)}{\partial p_j}\frac{1}{n_j}$ for jobs i,j. Then;
$\lim_{\Delta p \to 0} \left(\frac{f_i(p_i+\Delta p)-f_i(p_i)}{n_i \Delta p} - \frac{f_j(p_j)-f_j(p_j-\frac{n_i}{n_j}\Delta p)}{n_j \frac{n_i}{n_j}\Delta p} \right) < 0,$
$\lim_{\Delta p \to 0} \left(\frac{f_i(p_i+\Delta p)-f_i(p_i)+f_j(p_j)-f_j(p_j-\frac{n_i}{n_j}\Delta p)}{n_i \Delta p} \right) < 0.$
Then, $\exists\ \Delta p > 0$ s.t. $f_i(p_i + \Delta p) - f_i(p_i) + f_j(p_j) - f_j(p_j - \frac{n_i}{n_j}\Delta p) < 0$, which means the current solution can be improved by increasing p_i by Δ and decreasing p_j by $\frac{n_i}{n_j}\Delta$. This proves that if a solution does not satisfy the conditions above, then it is not locally optimal. □

As we have shown that similar properties as the weighted case apply to the alternative formulation for the total completion time case we can employ similar approaches to find an approximation for the efficient frontier. We can generate approximate efficient solution set by solving the new model in the SCP-based method as described in Section 3.1. We can also employ our CIBA method by just modifying the cost index r_i as $r_i = \frac{\partial f_i(p_i)}{\partial p_i}\frac{1}{n_i}$ where n_i is the number of jobs that job i precedes including itself. In the next section, we discuss the properties and the solution methods achieved so far on an example problem.

3.4 Numerical Example

In this section, we give a numerical example for the total weighted completion time problem and apply the SCP-based method and the CIBA method to generate an approximate efficient frontier. In this problem we have 5 jobs and the design attributes (D_i, L_i, d_i and S_i) of each job along with the required cutting tool type are given in Table 3.1. We consider a machine with $C_o = 0.25$ and $H = 5$. We first calculated the p_i^l and p_i^u values and formed the manufacturing cost function ($f_i(p_i)$) for each job as follows:

$$
\begin{aligned}
f_1(p_1) &= 0.25p_1 + 0.26p_1^{-1.32} \quad \text{where} \quad 0.29 \leq p_1 \leq 1.15 \\
f_2(p_2) &= 0.25p_2 + 0.21p_2^{-1.43} \quad \text{where} \quad 0.44 \leq p_2 \leq 1.09 \\
f_3(p_3) &= 0.25p_3 + 0.02p_3^{-1.71} \quad \text{where} \quad 0.29 \leq p_3 \leq 0.52 \\
f_4(p_4) &= 0.25p_4 + 0.18p_4^{-1.32} \quad \text{where} \quad 0.20 \leq p_4 \leq 0.97 \\
f_5(p_5) &= 0.25p_5 + 0.02p_5^{-1.71} \quad \text{where} \quad 0.25 \leq p_5 \leq 0.47
\end{aligned}
$$

Job	D_i	L_i	d_i	S_i	Tool	w_i
1	1.9	4.6	0.211	222	5	1.2
2	2.0	4.9	0.151	173	1	1.3
3	1.6	4.3	0.204	269	9	1.1
4	1.9	4.6	0.138	176	5	1.9
5	1.6	4.2	0.170	250	9	1.0

Table 3.1: Specifications of the jobs in the numerical example

To find the solution Z_1 and the corresponding schedule, we first set all jobs' processing times to their lower bounds and apply the WSPT rule as shown in Table 3.2. Total manufacturing cost, F_1, for this initial schedule is 4.27 and the corresponding optimal total weighted completion time, F_2, is 4.752. The initial schedule gives us the ideal total weighted completion time and the nadir manufacturing cost. We also have the schedule (Z_2) corresponding to the minimum

manufacturing cost in Table 3.2. For the minimum cost settings, $F_1 = 1.77$ and $F_2 = 14.288$. This schedule gives us the ideal manufacturing cost and the nadir total weighted completion time.

	Z_1					Z_2				
Position	Job	p	w/p	r_i	$f_i(p_i)$	Job	p	w/p	r_i	$f_i(p_i)$
1	4	0.20	9.50	-1.49	1.56	5	0.47	2.13	0.0	0.19
2	1	0.29	4.14	-1.26	1.40	3	0.52	2.12	0.0	0.19
3	5	0.25	4.00	-0.36	0.28	4	0.97	1.96	0.0	0.43
4	3	0.29	3.79	-0.30	0.24	2	1.09	1.19	0.0	0.46
5	2	0.44	2.95	-1.51	0.79	1	1.15	1.04	0.0	0.50

Table 3.2: Schedules at Z_1 and Z_2

In this example, we set the step size $\Delta = 0.1$. In Table 3.3, we present the perturbed job (j), and after each perturbation the new p_j, w_j/p_j, sequence and r_i's along with the F_1 and F_2 values for the first 10 iterations of the CIBA method. As stated in the proposed CIBA method, we start with the solution Z_1, and perturb the job with the minimum r_i in each iteration, such as job 2 at the first iteration. At each iteration we re-sequence the jobs if the WSPT rule is violated. As an example, after we perturb job 1 in iteration 3, we re-sequence the jobs to satisfy the WSPT rule. The algorithm progresses in this way by perturbing one job at a time until we reach to solution Z_2. As can be seen in Table 3.3, at each iteration we improve the total manufacturing cost (F_1) while losing from the total weighted completion time (F_2) (i.e. generate a new efficient solution) as discussed in Lemma 3.7.

Using the CIBA method, we generated 27 approximate efficient solutions for the example problem. As discussed before, we could also use a commercial NLP solver to find the minimum manufacturing cost for a given total weighted completion time level. In this study, we model the problem in GAMS and use the MINOS solver as the SCP-based method. In Table 3.4, there are two schedules generated for the total weighted completion time of 7.592. The first schedule is

Iter.	Perturbed Job (j)	p_j	w_j/p_j	Sequence	r_i 1	2	3	4	5	F_2	F_1
0				4 1 5 3 2	-1.26	-1.51	-0.30	-1.49	-0.36	4.752	4.27
1	2	0.54	2.407	4 1 5 3 2	-1.26	-0.84	-0.30	-1.49	-0.36	4.882	4.12
2	4	0.30	6.333	4 1 5 3 2	-1.26	-0.84	-0.30	-0.56	-0.36	5.532	3.52
3	1	0.39	3.077	4 5 3 1 2	-1.12	-0.84	-0.20	-0.56	-0.26	5.821	3.12
4	1	0.49	2.449	4 5 3 1 2	-1.62	-0.84	-0.20	-0.56	-0.26	6.071	2.91
5	2	0.64	2.031	4 5 3 1 2	-0.62	-0.49	-0.20	-0.56	-0.26	6.201	2.83
6	1	0.59	2.034	4 5 3 1 2	-0.37	-0.49	-0.20	-0.56	-0.26	6.451	2.71
7	4	0.40	4.750	4 5 3 1 2	-0.37	-0.49	-0.20	-0.27	-0.26	7.101	2.45
8	2	0.74	1.757	4 5 3 1 2	-0.37	-0.28	-0.20	-0.27	-0.26	7.231	2.40
9	1	0.69	1.739	4 5 3 2 1	-0.47	-0.15	-0.20	-0.27	-0.26	7.472	2.33
10	1	0.79	1.519	4 5 3 2 1	-0.28	-0.15	-0.20	-0.27	-0.26	7.592	2.28

Table 3.3: Results of the first 10 iterations by the CIBA method

found by the CIBA method at iteration 10 given in Table 3.3. We also solved the same problem by MINOS. It can be seen from Table 3.4 that two schedules are different in terms of job sequences and job processing times. When we consider the schedule found by the SCP-based method using MINOS, we see that it satisfies Lemma 3.2, which states that total weighted completion time must be tight at optimality and it also satisfies the WSPT rule. The MINOS solution has integer X_{ij}'s, so that it gives a feasible schedule (Lemma 3.4). For this particular solution, the CIBA method gives a slightly better solution ($F_1 = 2.28$) than the GAMS/MINOS solver ($F_1 = 2.30$), which is an example for the case that a solution found by MINOS may not be a global optimal (Lemma 3.5). Finally, when we check the r_i values of the MINOS solution, we see that they satisfy Lemma 3.6 which states that a local optimal solution cannot be improved by just changing the processing times of jobs. In order to find the global optimum of F_1 for a given $F_2 = 7.592$, we solve the LSCP model by GAMS/BARON, which gives $F_1 = 2.265$ with the job sequence of 4-5-3-2-1 (the same sequence with the CIBA), and the processing times (0.413, 0.277, 0.290, 0.647, 0.820), respectively.

	schedule by CIBA method					schedule by SCP-based method				
Position	job(i)	p_i	w_i/p_i	r_i	$f_i(p_i)$	job(i)	p_i	w_i/p_i	r_i	$f_i(p_i)$
1	4	0.40	4.75	-0.27	0.703	4	0.414	4.589	-0.245	0.68
2	5	0.25	4.00	-0.26	0.277	5	0.256	3.906	-0.245	0.27
3	3	0.29	3.79	-0.20	0.239	3	0.290	3.793	-0.202	0.24
4	2	0.74	1.76	-0.15	0.508	1	0.672	1.786	-0.245	0.50
5	1	0.79	1.52	-0.28	0.552	2	0.769	1.691	-0.245	0.61

Table 3.4: Schedules generated by different methods when F_2=7.592

3.5 Computational Results

In this chapter, we considered two bicriteria production optimization problems. The first problem deals with minimizing the total manufacturing cost and total weighted completion time objectives. The second one deals with minimizing the total manufacturing cost and total completion time objectives. For each problem, we provided two different efficient frontier approximation algorithms: namely the SCP-based method and the CIBA method. The SCP-based method is modeled in GAMS 2.5 using solver MINOS 5.51. GAMS is a well known commercial mathematical modeling tool with many different solvers attached to it ([16]). MINOS is the oldest NLP solver available with GAMS and is still the NLP solver that is used the most frequently. MINOS has been developed at the Systems Optimization Laboratory at Stanford University and it is still being improved. The CIBA method is coded in C and compiled with Gnu C compiler. All codes were run on a computer with 1294 MB memory and Pentium III 1133 MHz. CPU. In this section, we discuss the results of the computational study.

There are three experimental factors that can affect the efficiency of the proposed methods as listed in Table 3.5. The number of jobs (N) is an important factor that affects the solution quality and computational requirements. The machine type is considered since different machines have different cutting power levels and different operating costs. Machines with higher maximum cutting

power, H, are more expensive to buy so operating cost of them, C_o, is higher. C_o and H affect the p^l and p^u levels as well as the shape of the manufacturing cost function. C_t, the tooling cost level, also affects the p^u and the shape of the manufacturing cost function. We also consider 10 different cutting tool types with the specific coefficients given in Table 3.6. Each job can be manufactured by one of these cutting tools. For each experimental setting $(3 * 3 * 2)$, we took five replications resulting in 90 different problem settings. Furthermore, we generated jobs' technical specifications as follows: D_i are selected from U[1,4], L_i from U[4,6], d_i from U[0.05,0.3], S_i from U[150,250], where U[a,b] is a uniform distribution in the interval [a,b]. For the weighted case we generated a weight for each job from U[1,10]. Finally, we used two different levels of step size Δ, such as 0.01 and 0.03.

Factor	Definition	Level 1	Level 2	Level 3
N	Number of jobs	50	100	150
m/c	Machine type	$C_o = 1, H = 5$	$C_o = 2, H = 10$	$C_o = 4, H = 20$
C_t	Tooling cost level	U[6,10]	U[15,19]	

Table 3.5: Experimental design factors

We first present the results for the total weighted completion time problem. The number of approximate efficient points generated by the CIBA method depends on the experimental factors, job specifications and Δ. We cannot determine the number of points to be generated by the CIBA method in advance. Therefore, to compare these two approaches we first run the CIBA method and generate a set of approximate efficient points. Then, we choose a subset of this solution set and run the SCP-based model for this subset. As discussed earlier, we generate points with total weighted completion time values in $[K^l, K^u]$. Out of these points, we chose 50 points other than Z_1 and Z_2 such that each successive point pair has equal (or almost equal) separation. This is because we want to test the algorithms at different total weighted completion time levels along the efficient frontier.

Tool	α	β	γ	K	b	c	e	C_m	g	h	l	C_s
1	4.0	1.40	1.16	40960000	0.91	0.78	0.75	2.394	-1.52	1.004	0.25	204620000
2	4.3	1.60	1.20	37015056	0.96	0.70	0.71	1.637	-1.60	1.005	0.30	259500000
3	3.7	1.30	1.10	13767340	0.90	0.75	0.72	2.315	-1.45	1.015	0.25	202010000
4	3.7	1.28	1.05	11001020	0.80	0.75	0.70	2.415	-1.63	1.052	0.30	205740000
5	4.1	1.26	1.05	48724925	0.80	0.77	0.69	2.545	-1.69	1.005	0.40	204500000
6	4.1	1.30	1.10	57225273	0.87	0.77	0.69	2.213	-1.55	1.005	0.25	202220000
7	3.7	1.30	1.05	13767340	0.83	0.75	0.73	2.321	-1.63	1.015	0.30	203500000
8	3.8	1.20	1.05	23451637	0.88	0.83	0.72	2.321	-1.55	1.016	0.18	213570000
9	4.2	1.65	1.20	56158018	0.90	0.78	0.65	1.706	-1.54	1.104	0.32	211825000
10	3.8	1.20	1.05	23451637	0.81	0.75	0.72	2.298	-1.55	1.016	0.18	203500000

Table 3.6: Technical coefficients of the cutting tools

We measured the relative difference between the F_1 values for a given F_2 value, $R = (F_1(\text{CIBA}) - F_1(\text{SCP}))/F_1(\text{SCP})$. Another critical issue to consider is the computational requirements of both methods. In Table 3.7, we summarize the R level and the required CPU's. The given CPU's in this table are measured for the entire solution sets. The data shows that the relative difference between two methods on the average is very small in favor of the CIBA. We can say that on the average the CIBA performs slightly better than the commercial NLP solver MINOS in terms of the solution quality. The maximum R values show that there are cases where the SCP-based method performs better. Moreover, we conclude that the CIBA method can find significantly higher number of efficient points than the SCP-based method in a much shorter computational time. For $\Delta = 0.01$, the SCP-based method used 352.91 CPU seconds on the average to solve for 50 solution points, but the CIBA method just spent 29.56 CPU seconds on the average to generate 354,597 points. In an 8% of MINOS' CPU requirement, the CIBA method can generate 7000 times more alternatives. It is important to note that we originally had a mixed-integer nonlinear programming (MINLP) formulation. Due to Lemmas 3.3 and 3.9, we were able to solve these problems by using the MINOS solver. Still, the CPU needed to solve the SCP-model is quite high. As expected, when we increase the step size, Δ, we loose from the solution

quality at each point but gain from the CPU time. Moreover, for a higher Δ value, the size of the approximate efficient solution set decreases. When we check the results for different levels of N, we observe that increase in the number of jobs increases the SCP-based method's CPU much more than the CIBA method, but the relative difference between two methods, R, slightly improves in favor of the CIBA method.

Δ	Measure	Min	Max	Mean	Std. Dev.
0.01	R	-0.003970	0.001559	-0.000108	0.000665
	SCP-based CPU	16.86	961.59	352.91	356.88
	CIBA CPU	0.78	125.59	29.56	35.17
	CIBA set size	32,084	1,066,094	354,597	296,026
0.03	R	-0.000936	0.000216	-0.000041	0.000177
	SCP-based CPU	17.30	955.57	354.19	355.70
	CIBA CPU	0.26	41.24	9.76	11.56
	CIBA set size	10,693	355,344	118,188	98,674

Table 3.7: Performance measures for the weighted case

Next, we discuss the computational results for the total completion time case. In Table 3.8, we present the results for the total completion time case for different Δ values. We see that the overall performance of CIBA is very close to the SCP-based method but the SCP-based method performs slightly better than the CIBA in terms of solution quality. The minimum R values indicate that there are cases where the CIBA performs better. Results show that both methods require less computation time for the unweighted case compared to the weighted case as expected. Moreover, the CIBA method generated less number of efficient points for the unweighted case. This is because when the weights of completion times are selected from the interval [1,10] we achieve a larger $[K^l, K^u]$ interval so that the CIBA can generate more points for the weighted case if the same Δ value is used as in the total completion time case.

Another important question is the absolute performance of the SCP-based

Δ	Measure	Min	Max	Mean	Std. Dev.
0.01	R	-0.000076	0.001232	0.000313	0.000309
	SCP-based CPU	12.70	509.43	201.60	194.72
	CIBA CPU	0.01	0.14	0.06	0.03
	CIBA set size	1324	9731	4471.7	2286.3
0.03	R	-0.019599	0.010221	0.002610	0.003642
	SCP-based CPU	11.42	522.29	202.65	195.51
	CIBA CPU	0.01	0.05	0.021	0.01
	CIBA set size	444	3,247	1,494.6	763.2

Table 3.8: Performance measures for the total completion time case

method from the global minimum due to Lemmas 3.5 and 3.10. We could solve the LSCP model only for 5 and 8 jobs within reasonable CPU times by using the GAMS/BARON solver version 7.2.3. BARON (Branch And Reduce Optimization Navigator) is a well known MINLP solver and it is hooked up to GAMS modeling system. BARON combines constraint propagation, interval analysis, and duality for efficient range reduction with rigorous relaxations constructed by enlarging the feasible region and/or underestimating the objective function. We applied the same experimental settings as above. We took five replications for each setting. For each replication, we applied the SCP-based approach for 5 efficient points generated by the CIBA method. Then, we solved a total of 300 MINLP problems by BARON and MINOS. Table 3.9 shows the relative difference values for the SCP-based methods using MINOS versus BARON and the CIBA versus the SCP-based method using BARON. Results show that both MINOS and CIBA find solutions very close to global optimal. There are some cases where MINOS finds the global optimal. The computational requirements of MINOS and CIBA are negligible for the considered number of jobs levels. We observe that when we increase the number of jobs from 5 to 8, the CPU time required by BARON increases by 300 times.

N	Measure	Min	Max	Mean	Std. Dev.
5	R(CIBA-BARON)	0.000014	0.005895	0.001007	0.001433
	R(MINOS-BARON)	0.0	0.006673	0.001258	0.002132
	BARON CPU	5.51	13.32	10.07	1.79
8	R(CIBA-BARON)	0.000010	0.004125	0.000105	0.000109
	R(MINOS-BARON)	0.0	0.004552	0.000092	0.001348
	BARON CPU	993.90	4627.83	3010.53	943.08

Table 3.9: Comparison with the global optimal solutions

Up to this point we have compared the pointwise quality of individual solutions generated by different methods. However, since this is a multi-objective optimization problem, we also check the approximation quality of solution sets generated by the CIBA and SCP-based methods. In the literature, there are different metrics used to compare the approximation quality of solution sets generated by different methods. In this chapter, we will use three of them. The first one is the area method proposed by Zitzler and Thiele [97], which measures the size of the objective value space covered by a solution set. The second metric is the coverage difference of two sets, $CD(A, B)$, by Zitzler [96], such that $CD(A, B) = Area(A \cup B) - Area(B)$. This measure shows the contribution of solution set A to the area covered by solution set B. These two metrics use the ideal and nadir values of the objective functions in order to normalize the objective values of solutions and calculate the corresponding areas. In our problem, ideal and nadir values are achieved at solutions Z_1 and Z_2. The last metric we use is the probability, $P(A, B)$, that an algorithm, A, gives a better solution than another algorithm, B. This metric is proposed by Hansen and Jaszkiewicz [43]. It is calculated as

$P(A,B) = \int_{u\in[0,1]} C(A(u),B(u))\,du$, where

$$C(A(u),B(u)) = \begin{cases} 1 & f(A(u)) < f(B(u)) \\ 1/2 & f(A(u)) = f(B(u)) \\ 0 & f(A(u)) > f(B(u)) \end{cases}$$

and $f(A(u)) = min_{x\in A}\{max(uF_1'(x),(1-u)F_2'(x))\}$ where $F_1'(x) = \frac{F_1(x)-F_1(Z_2)}{F_1(Z_1)-F_1(Z_2)}$ which is a normalization of F_1. Here u is the weight of the normalized objective function F_1 for the decision maker. The method basically tries a number of u values between 0 and 1 and estimates the decision maker's probability to choose a solution generated by method A. The results in Table 3.10 show that on the average area covered by the CIBA algorithm is larger than the area covered by the SCP-based method. Although there is a small difference, paired t-test results show that CIBA is significantly better than SCP-based method in terms of the area measure. When we check the coverage difference results, we see that the contribution of CIBA to the area covered by the SCP-based method is much more than the opposite measure. Finally, we check the probability measure, which shows that for the 99.5% of the cases on the average, the decision maker would prefer to implement a solution achieved by the CIBA method.

Metric	Mean	Min	Max
Area(CIBA)	0.843	0.816	0.867
Area(SCP)	0.833	0.805	0.856
CD(CIBA,SCP)	0.011	0.010	0.012
CD(SCP,CIBA)	0.000002	0.0	0.000029
P(CIBA,SCP)	0.995	0.975	1.000

Table 3.10: Comparison of the approximation algorithms for $\Delta = 0.01$

Computational results show that the CIBA method has almost same pointwise cost quality with the SCP-based approach despite the much less computational time requirement. More than that, the CIBA method can generate significantly

higher number of efficient solutions than the SCP-based methods in a short computation time. As a result, the approximation quality measures that we calculated for both methods show that CIBA achieves significantly better approximations of the efficient frontiers than the SCP-based method.

3.6 Conclusion

In this chapter, we considered the bicriteria problem with the objectives of minimizing the manufacturing cost and the total weighted completion time on a single CNC turning machine. We used ϵ-approach method and proposed a very effective single criterion model to find efficient solutions. The problem is to minimize total manufacturing cost for a given set of jobs for different levels of total weighted completion time. This model can be solved to integer local optimality by just using a commercial NLP solver. Additionally, we have derived important optimality properties for the model which led us to design a very quick and effective algorithm which generates an approximate efficient solution set.

Although we have focused on CNC turning machines for practical purposes, our results are valid for any nonlinear convex processing cost function. Furthermore, all the properties and methods that we derive below apply to the linear cost function case as well. This chapter has been published as a paper (Gürel and Aktürk [41]). In the next chapter, we will extend the discussion to parallel machine environment.

Chapter 4

Time/Cost Trade-offs in Parallel Machine Scheduling

In this chapter, we consider a time/cost trade-off problem in identical parallel machines environment. The term "identical" implies that the technical specifications of the machines are the same so that a job has the same manufacturing cost function on different machines. We have two objectives to minimize: total manufacturing cost and total completion time. As we have discussed in Chapter 3, we cannot minimize both objectives at the same time. Therefore, as we have done in Chapter 3 our focus will be on finding efficient solutions for this bicriteria problem. Our results in this chapter, will be analogous to the results we have achieved in Chapter 3.

In Section 4.1, we give the problem definition and by using the ϵ-approach, we propose a mathematical formulation for the problem of minimizing total manufacturing cost subject to a given bound on the total completion time. In Section 4.2, we prove some useful properties for the problem. In Section 4.3, we propose an algorithm which generates approximate efficient solutions for the problem. In Section 4.4, we discuss our findings on a numerical example, and report the computational results in Section 4.5. Finally, we give concluding remarks in Section 4.6.

4.1 Problem Definition

The notation used throughout this chapter is as follows:

p_i:	processing time of job i.
p_i^l:	processing time lower bound for job i.
$f_i(p_i)$:	manufacturing cost function of processing time for job i.
p_i^u:	processing time level that gives the minimum manufacturing cost ($f_i(p_i)$) for job i.
C_o :	unit operating cost for CNC turning machines (\$/min).
T_i, e_i :	tooling cost multiplier and exponent for job i.

We have N jobs to be machined on M identical parallel CNC turning machines. Each job corresponds to a different turning operation to be performed on one of the machines so that each job has different cutting properties such as diameter, length, allowable surface roughness and cutting tool. Therefore, each job has a different $f_i(p_i)$, and different p_i^l and p_i^u. Since the machines are identical, $f_i(p_i)$, p_i^l and p_i^u for job i are same for all machines. Each CNC machine can perform one job at a time. The problem is to find the best schedule and processing times for each job in order to minimize total manufacturing cost and total completion time.

As given in Chapter 3, manufacturing cost function of job i as a function of p_i can be expressed as below:

$$f_i(p_i) = C_o p_i + T_i p_i^{e_i}$$

As discussed in Chapter 3, p_i^l is the lower bound for p_i and p_i^u is the processing time level that minimizes the manufacturing cost for job i. Similar to the previous chapter, since $T_i > 0$ and $e_i < 0$ always hold, $f_i(p_i)$ is a nonlinear convex function.

If we consider the minimization of total completion time as a single objective on identical parallel machines with fixed processing times we can solve the problem by using the following properties.

Property 4.1. The shortest processing time first (SPT) rule is optimal.

The SPT rule as given in Pinedo [73] is to schedule the smallest job on machine 1 at time zero, schedule the second smallest job on machine 2, and so on; the $(M+1)$th smallest job follows the smallest job on machine 1, $(M+2)$th smallest job follows the second smallest job on machine 2, and so on. The second property is due to Conway et al. [23] as below.

Property 4.2. One can interchange jobs in equivalent positions in sequence on different machines without having any effect on the total completion time.

Two positions on different machines are equivalent if the number of jobs succeeding these positions on the same machines are equal. Property 4.2 implies the existence of many alternative optimal schedules for the problem.

Using Property 4.1 and 4.2, we can determine how many jobs will be scheduled on each machine in the optimal solution. Then, we can find number of positions on each machine and determine which positions on different machines will be equivalent so that we can form sets of equivalent positions. Using this observation for the total completion time problem with fixed processing times, we can formulate our bicriteria problem with controllable processing times as an assignment problem of jobs to sets of equivalent positions.

In an optimal solution for the total completion time problem with fixed processing times, l machines will have $\left(\lfloor \frac{N}{M} \rfloor + 1\right)$ jobs where $l \equiv (N \bmod M)$ with minimal $l \geq 0$. The rest of the machines will have $\lfloor \frac{N}{M} \rfloor$ jobs. We define $S(j)$ as the number of positions in set j. We define $N(j)$ as the number of jobs succeeding each job in set j on the same machine plus the job itself. We assume that position 1 is the first position on machine 1, position 2 is the first on machine 2 and M is the first on machine M, position $M+1$ is the second position on machine 1, $M+2$ is the second on machine 2 and so on. If $l > 0$ then there will be $\left(\lfloor \frac{N}{M} \rfloor + 1\right)$ sets and set 1 will include position 1 to position l so that $S(1) = l$ and $N(1) = \lceil \frac{N}{M} \rceil$. Set 2 will include position $(l+1)$ to position $(l+M)$ so that $S(2) = M$ and $N(2) = \lceil \frac{N}{M} \rceil - 1$ and, so on. If $l = 0$, then each machine will have equal number of jobs and there will be $\frac{N}{M}$ sets. Each set j will include M positions from position $((j-1)M+1)$ to jM. Then, we can determine $N(j)$

for each set j as follows:

$$N(j) = \left\lceil \frac{N}{M} \right\rceil - j + 1$$

Then, a mathematical formulation for the bicriteria problem with controllable processing times is as below, where X_{ij} is the binary variable which controls if job i is placed in one of the positions in set j.

$$\begin{aligned}
\min \ F_1 &: \sum_{i=1}^{N} f_i(p_i) = \sum_{i=1}^{N} C_o p_i + T_i p_i^{e_i} \\
\min \ F_2 &: \sum_{i=1}^{N} \sum_{j} N(j) X_{ij} p_i \\
\text{s.t.} \quad & \sum_{j} X_{ij} = 1 && i = 1, \ldots, N && (4.1) \\
& \sum_{i=1}^{N} X_{ij} = S(j) && \forall j && (4.2) \\
& p_i^l \leq p_i \leq p_i^u && i = 1, \ldots, N && (4.3) \\
& X_{ij} \in \{0, 1\} && i = 1, \ldots, N, \forall j && (4.4)
\end{aligned}$$

In the mixed integer nonlinear programming (MINLP) model above, the first objective function F_1 is the total manufacturing cost. F_1 equals the sum of N nonlinear convex manufacturing cost functions, so it is a convex function. The second objective function F_2 is the total completion time. Total completion time is a nonlinear function since it is a sum of nonlinear terms. Constraint set (4.1) forces each job to be assigned to a position set. Constraint set (4.2) fixes the number of jobs to be assigned to each position set. In each set there are certain number of positions and the number of jobs assigned to a set must be equal to the number of positions in the set. Constraint set (4.3) sets the processing time lower and upper bounds for each job. In the next section, we define a single objective problem and give optimality properties on it.

4.2 Optimality Properties

As discussed earlier, we cannot minimize both objectives at the same time. Therefore, we need to find efficient solutions for the problem. A solution Z to a bi-criteria problem is efficient if there exists no other solution which is better than Z in one of the criteria and not worse in the other. Since we have the SPT rule as an optimal strategy for the total completion time problem, we can determine the efficient solution Z_1 with the minimum total completion time K^l and the maximum manufacturing cost by setting $p_i = p_i^l$ for all i and by applying the SPT rule. Similarly, we can find another efficient solution Z_2 with the minimum manufacturing cost by setting $p_i = p_i^u$ for all i and by applying the SPT rule. We denote the total completion time at Z_2 as K^u. Two solutions, Z_1 and Z_2, are the points that we can find by just using the SPT rule on processing time lower and upper bounds, respectively. However, a decision maker may find the manufacturing cost for Z_1 too high to pay or may find the total completion time for Z_2 too high. In order to find the efficient solutions other than Z_1 and Z_2, we can consider F_1 as a single objective to minimize subject to an F_2 constraint and formulate a single objective problem which we call SOP as below.

$$\begin{aligned} \min F_1 : \quad & \sum_{i=1}^{N} f_i(p_i) \\ \text{(SOP) s.t.} \quad & \sum_{i=1}^{N} \sum_{j} N(j) X_{ij} p_i \leq K \\ & \text{and } (4.1), (4.2), (4.3), (4.4) \end{aligned} \tag{4.5}$$

Constraint (4.5) guarantees that total completion time (F_2) of the schedule is less than or equal to a predefined value K. An alternative way of modeling the SOP is formulating it as an assignment problem of jobs to individual positions as commonly done in the literature. Such a formulation would not be using the result in Property 4.2. It would require more variables and constraints in the model. We can solve the SOP model by using MINLP solvers. Then, we can generate a set of n efficient solutions between Z_1 and Z_2 by using the following

algorithm denoted as the solver based approach (SBA).

SBA algorithm
Step 1. Find solutions Z_1 and Z_2, and calculate K^u and K^l by using the SPT rule for p^u and p^l values, respectively.
Step 2. Set $\Delta = (K^u - K^l)/(n+1)$.
Step 3. For $k = 1$ to n, solve the SOP for $K = K^l + k\Delta$.

We found the following useful properties for the SOP formulation above. These properties improved the SBA method and provided a clearer interpretation of the problem.

Lemma 4.1. *When $K \leq K^u$, constraint* (4.5) *on F_2 must be tight at the optimal solution.*

Proof. When $K > K^u$, total manufacturing cost can be minimized by setting $p_i = p_i^u$ for all i and constraint (4.5) can be satisfied by applying the SPT rule. In such a case, constraint (4.5) is loose. When $K \leq K^u$, if constraint (4.5) is loose, then, it is sure that we have at least one job i such that $p_i < p_i^u$ and by increasing p_i we can decrease f_i so that we could improve F_1. Therefore, when $K \leq K^u$, a solution cannot be optimal if constraint (4.5) is loose. □

In our single objective problem, we model the total completion time as a resource constraint to be used to minimize total manufacturing cost. Therefore, Lemma 4.1 states that when this resource is scarce ($K \leq K^u$), we must fully utilize it. Lemma 4.1 also implies that an optimal solution for the SOP must have the minimum total completion time for the optimal processing times. This implies an optimal solution must satisfy the rules in Property 4.1 and 4.2. To further explore the problem we next consider the relaxed SOP where X_{ij}'s are allowed to take any values in the interval [0,1]. The next property is an extension of Lemma 4.1 for locally optimal solutions to the relaxed SOP.

Corollary 4.1. For the relaxed SOP, when $K \leq K^u$ constraint (4.5) on F_2 must be tight at locally optimal solutions.

Next property is an important one which states that any local optimal solution to the relaxed problem has binary X_{ij}'s. Non-integer local optimal solutions may exist only if there are multiple jobs having identical processing times in the solution. However, such non-integer solutions are alternative solutions for existing integer local optimal solutions.

Lemma 4.2. *When $K \leq K^u$, in a local optimal solution for the relaxed SOP, a job cannot be assigned to multiple position sets, i.e. a local optimal solution must have integer X_{ij}'s.*

Proof. Consider two jobs i_1 and i_2 placed to positions in different sets j_1 and j_2, respectively. Suppose that $p_{i_1} < p_{i_2}$ and $N(j_1) > N(j_2)$. Consider a local optimal solution Z for the relaxed problem. The assignment variables for jobs i_1 and i_2 and positions j_1 and j_2 in Z are as follows:
$X_{i_1 j_1} = \rho_1$ and $X_{i_2 j_1} = \rho_2$,
$X_{i_1 j_2} = \xi_1$ and $X_{i_2 j_2} = \xi_2$, where $\rho_1,\ \rho_2,\ \xi_1$ and ξ_2 are positive.

Total completion time calculated for Z is as below:

$$\begin{aligned} F_2(Z) &= \sum_{i=1}^{N} \sum_{j} N(j) X_{ij} p_i \\ &= \Phi + N(j_1)[\rho_1 p_{i_1} + \rho_2 p_{i_2}] + N(j_2)[\xi_1 p_{i_1} + \xi_2 p_{i_2}] \\ &= \Phi + [N(j_1)\rho_1 + N(j_2)\xi_1] p_{i_1} + [N(j_1)\rho_2 + N(j_2)\xi_2] p_{i_2} \end{aligned}$$

where Φ is a constant value. Suppose that without changing the processing times, we change the assignment variables to get a new solution Z'. Setting $\delta = \min(\xi_1, \rho_2)$, new values for the assignment variables are as follows:
$X_{i_1 j_1} = \rho_1 + \delta$, $X_{i_2 j_1} = \rho_2 - \delta$,
$X_{i_1 j_2} = \xi_1 - \delta$ and $X_{i_2 j_2} = \xi_2 + \delta$.
By this arrangement, we reallocate these two jobs to position sets j_1 and j_2 such that we increase job i_1's ratio in the preceding position set j_1 without disturbing the assignment constraints.

Total completion time of the solution after this arrangement is:

$$\begin{aligned} F_2(Z') &= \Phi + [N(j_1)(\rho_1 + \delta) + N(j_2)(\xi_1 - \delta)]p_{i_1} \\ &+ [N(j_1)(\rho_2 - \delta) + N(j_2)(\xi_2 + \delta)]p_{i_2} \end{aligned}$$

Then, $F_2(Z') - F_2(Z) = \delta(N(j_1) - N(j_2))(p_{i_1} - p_{i_2}) < 0$, since $N(j_1) > N(j_2)$ and $p_{i_1} < p_{i_2}$. Then, since $K \leq K^u$ we can improve total manufacturing cost by increasing processing times. This proves that there exists an improving feasible direction for solution Z so that Z cannot be a local optimal solution. When we generalize this result for all jobs and position sets, we conclude that a solution with non-integer X_{ij}'s cannot be a local optimal solution for the problem. However, for the cases of $p_{i_1} = p_{i_2}$ we may have alternative non-integer local optimal solutions. □

This result is an extremely important one since it shows that although our problem is a MINLP problem, we can employ nonlinear programming (NLP) solvers to solve its relaxed form and achieve integer solutions. This is critical since NLP solvers are computationally more efficient than MINLP solvers. If the objective function is convex, in general, NLP solvers can only guarantee to find local optimal solutions. However, if the feasible region for the problem is a convex set, a local optimal solution is globally optimal. We next check if the feasible region for the problem is a convex set to see whether NLP solvers can guarantee to find the global optimum for the problem.

Lemma 4.3. *The feasible region for the relaxed SOP is not a convex set.*

Proof. Consider two jobs i_1 and i_2 in a schedule called A_1. They are assigned at positions in sets k and $k+1$, respectively. Suppose that $N(k) = r + 1$ and $N(k+1) = r$ and the processing times are $p_{i_1} = s_1$ and $p_{i_2} = s_2$ where $s_1 < s_2$. Further, suppose that $F_2(A_1) = Q + (r+1)s_1 + rs_2 = K$ where Q is a constant.

Consider another schedule A_2 which is identical to A_1 except that job i_1 is assigned to the position set $k+1$ and i_2 is assigned to the position set k with

processing times $p_{i_1} = q_1$ and $p_{i_2} = q_2$ where $q_2 < q_1$. Suppose that $F_2(A_2) = Q + (r+1)q_2 + rq_1 = K$.

Next, define a point A which is a convex combination of A_1 and A_2 as follows: $A = \lambda A_1 + (1-\lambda)A_2$ where $0 < \lambda < 1$. At point A, $p_{i_1} = \lambda s_1 + (1-\lambda)q_1$ and $p_{i_2} = \lambda s_2 + (1-\lambda)q_2$. Also, $X_{i_1 k} = \lambda$, $X_{i_1(k+1)} = (1-\lambda)$, $X_{i_2 k} = (1-\lambda)$ and $X_{i_2(k+1)} = \lambda$.

$$\begin{aligned} F_2(A) &= Q + [(r+1)\lambda + r(1-\lambda)]p_{i_1} + [(r+1)(1-\lambda) + r\lambda]p_{i_2} \\ &= Q + \lambda^2[(r+1)s_1 + rs_2] + (1-\lambda)^2[(r+1)q_2 + rq_1] \\ &+ \lambda(1-\lambda)[(r+1)q_1 + rq_2] + \lambda(1-\lambda)[(r+1)s_2 + rs_1] \\ &> K \end{aligned}$$

since $s_1 < s_2$ and $q_2 < q_1$. This shows that feasible region for the relaxed problem is not a convex set. □

This lemma indicates that NLP solvers may not be able to find global optimum, so they only guarantee to achieve local optimal solutions. Although the complexity of the problem is open, this lemma supports the difficulty of the problem again due to Murty and Kabadi [64].

We showed that NLP solvers guarantee to find integer local optimal solutions for the SOP. A way of solving this problem to global optimum is to solve for processing times for all possible job-position allocations which is computationally inefficient except for small instances. In order to find the global optimal solution for the single objective problem by using MINLP solvers, we next present a linearized single objective problem LSOP model below. In this model, constraint (4.5) is replaced with the constraint set 4.6- 4.10 in which the term $X_{ij}p_i$ is replaced with the variable Y_{ij}. The parameter B in the model below denotes a large positive number.

$$\min F_1 : \sum_{i=1}^{N} f_i(p_i)$$

$$\text{(LSOP) s.t.} \quad \sum_{i=1}^{N} \sum_{j} N(j) Y_{ij} \leq K \tag{4.6}$$

$$Y_{ij} \geq p_i + (X_{ij} - 1)B \qquad i = 1, \ldots, N, \ \forall j \tag{4.7}$$

$$Y_{ij} \leq p_i + (1 - X_{ij})B \qquad i = 1, \ldots, N, \ \forall j \tag{4.8}$$

$$Y_{ij} \leq BX_{ij} \qquad i = 1, \ldots, N, \ \forall j \tag{4.9}$$

$$Y_{ij} \geq 0 \qquad i = 1, \ldots, N, \ \forall j \tag{4.10}$$

$$\text{and} \qquad (4.1), (4.2), (4.3), (4.4)$$

By replacing the terms $X_{ij}p_i$ with the variable Y_{ij} in constraint (4.5) we obtain constraint 4.6 in the above model. To assure the equivalence of the term $X_{ij}p_i$ and the variable Y_{ij} we add the constraint sets 4.7- 4.10 so that if $X_{ij} = 0$ then $Y_{ij} = 0$ and if $X_{ij} = 1$ then $Y_{ij} = p_i$. However, Lemma 4.2 is no longer valid for this linearized model since the linearization is only possible for binary X_{ij}'s, so we cannot use NLP solvers to solve the LSOP, instead we can use a MINLP solver to find the global optimal solution.

In this section, we introduced the SOP model and proposed the SBA method to generate a set of efficient solutions. We showed that NLP solvers can also be employed in SBA method to obtain approximate efficient solutions. We also gave the LSOP model which can be solved to global optimum by the MINLP solvers. In the next section, we propose a heuristic method which generates approximate efficient solutions for the bicriteria problem. Proposed method achieves almost equal solution quality compared to commercial NLP and MINLP solvers although it spends much less computation time.

4.3 A heuristic method to generate approximate efficient solutions

In this section, we first state a very important optimality property for the single objective problem. Different than the properties in Section 4.2, this optimality

property states a relationship that must hold between positions and processing times of the jobs in a local optimal solution. Based on this property we propose a heuristic algorithm which generates a set of approximate efficient solutions. A solution may be viewed as approximately efficient if it is efficient with respect to a large set of known solutions for a given problem. Since we are using a heuristic approach to find the minimum manufacturing cost value, F_1, for a given total completion time value, we denote these solutions as approximate efficient solutions. That means these solutions do not dominate each other but there can be another solution generated by an exact algorithm that could dominate any one of them.

Lemma 4.4. *Let i_1, i_2 be a pair of jobs in a local optimum and $n_{i_k} = \sum_j X_{i_k j} N(j)$, $k \in \{1, 2\}$. Then the optimal processing times p_{i_1}, p_{i_2} must satisfy the following conditions:*

i. If $p_{i_1} > p^l_{i_1}$ and $p_{i_2} > p^l_{i_2}$ then

$$\frac{1}{n_{i_1}} \frac{\partial f_{i_1}(p_{i_1})}{\partial p_{i_1}} = \frac{1}{n_{i_2}} \frac{\partial f_{i_2}(p_{i_2})}{\partial p_{i_2}}$$

ii. If $p_{i_1} = p^l_{i_1}$ and $p_{i_2} > p^l_{i_2}$ then

$$\frac{1}{n_{i_1}} \frac{\partial f_{i_1}(p_{i_1})}{\partial p_{i_1}} \geq \frac{1}{n_{i_2}} \frac{\partial f_{i_2}(p_{i_2})}{\partial p_{i_2}}$$

Proof. Suppose that in a solution $\frac{1}{n_{i_1}} \frac{\partial f_{i_1}(p_{i_1})}{\partial p_{i_1}} < \frac{1}{n_{i_2}} \frac{\partial f_{i_2}(p_{i_2})}{\partial p_{i_2}}$ for jobs i_1 and i_2. Then,

$$\lim_{\Delta p \to 0} \left(\frac{f_{i_1}(p_{i_1} + \Delta p) - f_{i_1}(p_{i_1})}{n_{i_1} \Delta p} - \frac{f_{i_2}(p_{i_2}) - f_{i_2}(p_{i_2} - \frac{n_{i_1}}{n_{i_2}} \Delta p)}{n_{i_2} \frac{n_{i_1}}{n_{i_2}} \Delta p} \right) < 0$$

$$\lim_{\Delta p \to 0} \left(\frac{f_{i_1}(p_{i_1} + \Delta p) - f_{i_1}(p_{i_1}) - f_{i_2}(p_{i_2}) + f_{i_2}(p_{i_2} - \frac{n_{i_1}}{n_{i_2}} \Delta p)}{n_{i_1} \Delta p} \right) < 0$$

Then, $\exists\ \Delta p > 0$ s.t. $f_{i_1}(p_{i_1} + \Delta p) + f_{i_2}(p_{i_2} - \frac{n_{i_1}}{n_{i_2}} \Delta p) - f_{i_1}(p_{i_1}) - f_{i_2}(p_{i_2}) < 0$, which means the current solution can be improved by increasing p_{i_1} by Δp and decreasing p_{i_2} by $\frac{n_{i_1}}{n_{i_2}} \Delta p$. This proves that a local optimal solution must satisfy the conditions above. □

This lemma states that there is a unique solution for the processing times of jobs which is optimal for a given job-position set allocation. When a job-position set allocation is given, we can find the processing times of the jobs by employing a search algorithm and using Lemma 4.4. Lemma 4.4 also implies that we cannot move from a local optimal solution to another one without changing the job-position set allocation since there is a unique local optimal solution for a certain job-position set allocation. It implies that we can find the global optimal solution for the problem by trying all possible job-position set allocations and solve each case for optimal processing times.

Using the information in Lemma 4.4 we propose an approximation algorithm, which we call as the most profitable job first (MPJ) algorithm, to find a set of approximate efficient solutions other than Z_1 and Z_2. MPJ algorithm starts with the efficient solution Z_1 which has the minimum total completion time but the maximum manufacturing cost. To generate a new approximate efficient solution, we want to select a job and increase its processing time. This will decrease the total manufacturing cost but increase the total completion time. Selecting the job to be perturbed is a very critical decision, and hence we propose a new measure t_i as given below:

$$t_i = \frac{\partial f_i(p_i)/\partial p_i}{N(j)}$$

We can interpret t_i as the estimated cost change per estimated unit total completion time change when processing time of job i is increased by one unit. This makes sense since we want to find efficient solutions by achieving maximum cost decrease per unit total completion time increase. As we know from Lemma 4.4, in an optimal solution t_i values must be equal for the jobs whose processing times are higher than their lower bounds. By selecting the job with the minimum t_i value, we want to satisfy or at least be very close to satisfying Lemma 4.4 at each step. In each iteration of MPJ, we increase the processing time of a selected job by a predefined value Δ so that we can improve total manufacturing cost while giving up from the total completion time. After increasing the processing time of the selected job, we check whether the SPT rule, specified in Property 4.1, is satisfied for the new schedule or not. If not, the SPT rule is applied. At the end of each iteration, we achieve a new solution with a better manufacturing cost but

a higher total completion time than the previous solution. The proposed MPJ algorithm generates a set of approximate efficient solutions as outlined below:

MPJ algorithm
Step 1. Find the efficient solutions Z_1 and Z_2.
Step 2. Start with the solution Z_1, set $F_1^{new} = F_1(Z_1)$ and $F_2^{new} = F_2(Z_1)$. While $F_2^{new} < K^u$ do the following:
Step 2.1. Select job i with the minimum t_i. If there are more than one such jobs, select the one with the longest processing time.
Step 2.2. Set $p_i = p_i + \Delta$.
Step 2.3. If the SPT rule is violated then re-sequence the jobs by the SPT rule.
Step 2.4. Update t_i indices for the perturbed job and for all other jobs whose position in sequence is changed in Step 2.3.
Step 2.5. Update $F_1^{new} = F_1^{new} - [f_i(p_i) - f_i(p_i + \Delta)]$ and recalculate F_2^{new}.
Step 2.6. If $F_2^{new} < K^u$, report the current schedule with F_1^{new} and F_2^{new} as a new approximate efficient solution.

The MPJ algorithm first finds two efficient solutions Z_1 and Z_2 in Step 1, then generates a new approximate efficient solution in between these two points in each iteration as shown below.

Lemma 4.5. *In each iteration of the MPJ algorithm, we generate a new approximate efficient solution.*

Proof. At each iteration of the MPJ algorithm, processing time of a selected job is increased by Δ amount. Since the total completion time is a regular scheduling measure, total completion time of new schedule is strictly higher than the previous one. Similarly, this increase will strictly decrease the total manufacturing cost. This means new solution cannot dominate previously generated solutions. Therefore, no two solutions generated by the MPJ algorithm can dominate each other. □

The MPJ algorithm generates a set of approximate efficient solutions which cannot dominate each other as shown above. We then utilize the set of these

discrete points to approximate the efficient frontier. In the MPJ algorithm, Δ is a very critical parameter which affects the quality of the solutions achieved. By using a smaller Δ value, the MPJ algorithm can generate more solutions each of which is more close to satisfy the conditions given in Lemma 4.4. In the next section, we will discuss the given properties and the algorithms on a numerical example.

4.4 Numerical Example

In this section, we give a 5 jobs-2 machines problem as an example to illustrate the properties and algorithms that we have discussed above. As we have discussed before, due to different job and tool properties each job has different p_i^l and p_i^u levels. In this numerical example we use the following manufacturing cost functions and processing time bounds.

$$
\begin{aligned}
f_1(p_1) &= 0.25p_1 + 3.30p_1^{-1.29} \quad \text{where} \quad 1.65 \leq p_1 \leq 3.45 \\
f_2(p_2) &= 0.25p_2 + 0.02p_2^{-1.71} \quad \text{where} \quad 0.20 \leq p_2 \leq 0.48 \\
f_3(p_3) &= 0.25p_3 + 0.20p_3^{-1.22} \quad \text{where} \quad 0.42 \leq p_3 \leq 0.99 \\
f_4(p_4) &= 0.25p_4 + 0.03p_4^{-1.22} \quad \text{where} \quad 0.18 \leq p_4 \leq 0.43 \\
f_5(p_5) &= 0.25p_5 + 0.20p_5^{-1.40} \quad \text{where} \quad 0.36 \leq p_5 \leq 1.05
\end{aligned}
$$

Since we have 5 jobs and 2 machines, we have 3 sets of equivalent positions. The first position is for the shortest job which will be succeeded by two positions on the same machine. The second and third positions are equivalent positions so they form a position set. Each one is succeeded by a single position. The fourth and fifth positions form the last set of equivalent positions and each one is the last position on its machine. We can assume that positions 1, 3 and 5 are on machine 1 and positions 2 and 4 are on machine 2.

We can find the solutions Z_1 and Z_2 given in Table 4.1 as discussed in Section 4.1. At Z_1, total completion time is 3.73 (ideal total completion time) and total manufacturing cost is 4.40. At Z_2, total completion time is 8.79 and total

manufacturing cost is 2.81 (ideal cost). Z_1 and Z_2 have different job-position allocations. In Table 4.1, we also give the t_i values for each job. At Z_1, $\frac{\partial f_5(p_5^l)}{\partial p_5} = -3.0$ and it is succeeded by a single job, then $t_5 = \frac{-3.0}{2} = -1.5$. Since $\frac{\partial f_i(p_i^u)}{\partial p_i} = 0$ for all i at Z_2, we see that $t_i = 0$ for all i.

	Z_1				Z_2			
Position	Job i	p_i	$f_i(p_i)$	t_i	Job i	p_i	$f_i(p_i)$	t_i
1	4	0.18	0.29	-0.56	4	0.43	0.19	0.0
2	2	0.20	0.34	-1.22	2	0.48	0.19	0.0
3	5	0.36	0.93	-1.50	3	0.99	0.45	0.0
4	3	0.42	0.68	-1.42	5	1.05	0.45	0.0
5	1	1.65	2.14	-1.10	1	3.45	1.53	0.0
		$F_1(Z_1) =$	4.40			$F_1(Z_2) =$	2.81	

Table 4.1: Schedules at Z_1 and Z_2

In Table 4.2, we give the first 7 iterations of the MPJ algorithm for the example problem. In the first iteration we consider the efficient solution Z_1. At Z_1 in Table 4.2, t_5 is minimum so we select job 5 to perturb (Step 2.1). We increase p_5 by $\Delta = 0.1$ from 0.36 to 0.46 (Step 2.2). Since the SPT order is violated we resequence the jobs and this results pairwise interchanging jobs 3 and 5 (Step 2.3). We update the t_i's for jobs 3 and 5 (Step 2.4). For the new sequence, $t_5 = -1.56$. Next, we report the achieved solution at the end of iteration 1 as given in Table 4.2 (Step 2.5). Still t_5 is the minimum, we increase p_5 again in iteration 2 and achieve a new schedule. The algorithm continues until it meets the total completion time level of K^u.

We use the estimated cost change and total completion time data to make decisions, so using smaller Δ would always give a better approximation. If we use a smaller Δ in our example, the sequence change that occurred in iteration 1 might occur in the next iteration or a different job may be selected in iteration 2 which may imply achieving different schedules. Δ also affects the number of iterations such that smaller Δ implies more iterations which means generating

Iter.	Job i	p_i	Sequence	t_1	t_2	t_3	t_4	t_5	F_2	F_1
0			4 2 5 3 1	-1.1	-1.22	-1.42	-0.47	-1.50	3.73	4.40
1	5	0.46	4 2 3 5 1	-1.1	-1.22	-0.71	-0.47	-1.56	3.89	4.18
2	5	0.56	4 2 3 5 1	-1.1	-1.22	-0.71	-0.47	-0.88	3.99	4.07
3	2	0.30	4 2 3 5 1	-1.1	-0.32	-0.71	-0.47	-0.88	4.19	3.93
4	1	1.75	4 2 3 5 1	-0.93	-0.32	-0.71	-0.47	-0.88	4.29	3.83
5	1	1.85	4 2 3 5 1	-0.79	-0.32	-0.71	-0.47	-0.88	4.39	3.75
6	5	0.66	4 2 3 5 1	-0.79	-0.32	-0.71	-0.47	-0.51	4.49	3.68
7	1	1.95	4 2 3 5 1	-0.67	-0.32	-0.71	-0.47	-0.51	4.59	3.61

Table 4.2: Results of the first 7 iterations of MPJ algorithm

more solutions.

To compare the solution that MPJ achieved at the end of iteration 1, we solved the single objective problem by the NLP solver GAMS/MINOS for $K = 3.89$. The result achieved by MINOS and the result achieved by MPJ are given in detail in Table 4.3. Total manufacturing cost achieved by MINOS is 4.20. From Table 4.2, we know that MPJ achieved the total manufacturing cost of 4.18. This is a case where MPJ performed better than MINOS. The reason for this situation can be seen in Table 4.3. Two solutions are different in terms of job sequence and processing times. This means MINOS stuck to a local optimal solution, but MPJ achieved a better solution at a different job sequence. If the sequences were the same, MINOS would achieve a better solution since it can change all jobs' processing times while MPJ changes the processing time of a single job at a time. Furthermore, we solved the LSOP model for the example for $K = 3.89$ by using the MINLP solver GAMS/BARON and achieved the global optimum which is the same as the solution achieved by MPJ. We solved LSOP model for all the F_2 levels of solutions achieved by MPJ. In Figure 4.1, we present the set of efficient solutions (including Z_1 and Z_2) found by the MPJ and corresponding efficient (global optimal) solutions achieved by GAMS/BARON for the example problem. For this small example, F_1 levels of solutions achieved by MPJ are very close to

global optimum and there are many cases in which they are equal.

schedule by MPJ					schedule by MINOS				
Position	job i	p_i	t_i	$f_i(p_i)$	Pos. Set	job i	p_i	t_i	$f_i(p_i)$
1	4	0.18	-0.47	0.29	1	4	0.18	-0.47	0.29
2	2	0.20	-1.22	0.36	2	2	0.21	-1.10	0.34
3	3	0.42	-0.71	0.68	2	5	0.41	-1.10	0.8
4	5	0.46	-1.56	0.71	3	3	0.46	-1.10	0.63
5	1	1.65	-1.10	2.14	3	1	1.65	-1.10	2.14

Table 4.3: Schedules generated by different methods at iteration 1

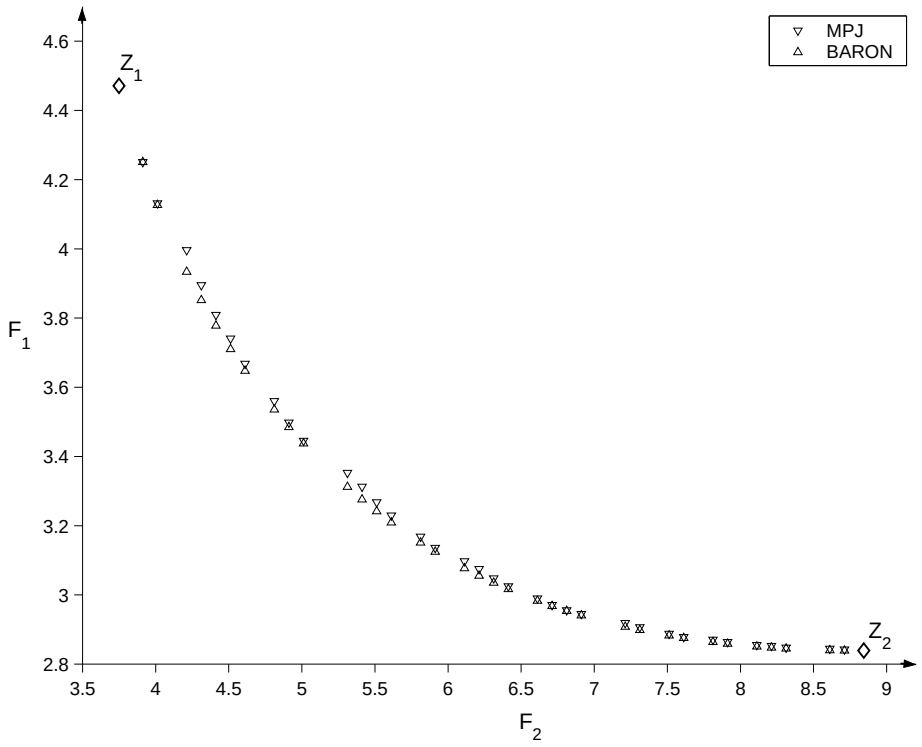

Figure 4.1: A set of efficient solutions for the numerical example

In Table 4.3, we see that solution by MINOS satisfies Lemma 4.1 and gives an integer local optimal solution to the NLP model as stated in Lemma 4.2. This example further shows that it is possible to achieve better solutions than MINOS since MINOS cannot guarantee global optimality due to Lemma 4.3 which states

the non-convex nature of the problem. When we check the MINOS solution in Table 4.3, we see that t_i values are equal for all jobs that have $p_i > p_i^l$, except t_1 for which $p_1 = p_1^l$. This is an example that illustrates the optimality conditions stated in Lemma 4.4.

4.5 Computational Analysis

In this chapter, we presented an efficient formulation for the single objective problem of minimizing total manufacturing cost subject to a total completion time constraint. We proposed two approximation algorithms SBA and MPJ to generate a set of approximate efficient solutions. We coded the SBA algorithm in GAMS 2.5 and implemented it by using the solver MINOS 5.51. We coded the MPJ algorithm in C language and compiled with Gnu C compiler version 3.2. All codes were run on the operating system Mandrake 10.0 with Linux 2.6.3 on a computer with 1294 MB memory and Pentium III 1133 MHz. CPU. In this section, we discuss the results of the computational study.

We considered 4 experimental factors which are given in Table 4.4. The first two factors, number of jobs and number of machines, define the problem size. The third factor is the machine type. We consider three types of machines with different H and C_o levels. The last experimental factor is the tooling cost level C_t, which affects the multiplier T_i in the tooling cost term of the manufacturing cost function. We generated C_t for each tool type as given in Table 4.4 where $U[a, b]$ is a uniform distribution in interval [a,b]. For each experimental setting, we solved for 5 replications resulting in 240 randomly generated problems. For each replication we generated cutting specifications (diameter, length, depth of cut and required surface roughness) of jobs randomly. For each job we randomly used one of the tool types out of ten types of cutting tools with different technical coefficients given in 3.6. Furthermore, to test the effect of Δ on the results we tried two levels of Δ, namely 0.01 and 0.03.

	Factors			
Level	N	M	Machine Type	Tool Cost (C_t)
1	50	3	$C_o = 1, H = 5$	U[6,10]
2	100	6	$C_o = 2, H = 10$	U[15,19]
3	150		$C_o = 4, H = 20$	
4	200			

Table 4.4: Experimental Design Factors

For each replication we first apply the MPJ method to the problem and generate a set of efficient solutions. Out of this set, we select 50 solutions, other than Z_1 and Z_2. To be able to compare solution quality of MPJ and SBA, we applied the SBA method for the F_2 levels of the selected solutions. In order to test the algorithms on different regions of the efficient frontier, we selected those 50 points such that each successive point pair has equal or almost equal separation. We considered the performance measure $R = \frac{F_1^{MPJ} - F_1^{SBA}}{F_1^{SBA}}$ which is the relative difference between F_1 level achieved by the MPJ algorithm and F_1 level achieved by the SBA method for the same K value.

In Table 4.5, we give the results for R, CPU time required by SBA to solve 50 instances and CPU time required by MPJ to generate a set of efficient solutions for both levels of Δ. We also include the size of the solution set generated by MPJ. We would like to emphasize the mean R level, which is less than 0.0003 and indicates that MPJ achieves almost equal cost performance as MINOS. Even for $\Delta = 0.03$, the mean R is around 0.3%. The minimum R is negative for $\Delta = 0.01$ which shows that MPJ can achieve better results than MINOS due to Lemma 4.3. The maximum level and standard deviation show that R values do not deviate much from the mean, and even the worst R is 0.1%. The next important criterion to compare two approaches is the computational requirements. Despite employing MINOS for just 50 points out of thousands generated by MPJ, in Table 4.5, we see that CPU time requirement of MINOS is incomparably high with respect to MPJ. On the average MPJ produces 5559 solutions, so MPJ can generate

many alternative solutions in a very short computation time. When we increase Δ, we see that R is increased but CPU time required by MPJ is decreased as expected. However, even for $\Delta = 0.01$, the required CPU time is 0.37 seconds on the average, so we can decrease Δ even further and it would still not require much CPU time and would give a better approximation in terms of solution quality and number of alternative solutions.

Δ	Measure	Mean	Min	Max	Std. Dev.
0.01	R	0.000282	-0.002119	0.001213	0.000318
	MINOS CPU sec.	78.92	1.96	323.35	91.37
	MPJ CPU sec.	0.08	0.01	0.20	0.05
	MPJ set size	5559.16	1323	12459	3012.14
0.03	R	0.002764	0.000178	0.010918	0.002654
	MINOS CPU sec.	79.45	2.26	327.91	92.13
	MPJ CPU sec.	0.03	0.00	0.08	0.02
	MPJ set size	1857.03	443	4159	1005.58

Table 4.5: Performance measures for different step size levels

Table 4.6 gives R and CPU time measures for different levels of number of jobs and number of machines. As N increases, we do not observe a significant decrease or increase on R. The CPU time required by MINOS is strongly affected by N, which increases rapidly from 4.89 to 206.53 (almost 40 times), when N is increased from 50 to 200. On the other hand, the CPU seconds required by MPJ increased slightly from 0.03 to 0.14. This shows that as N increases, CPU time increase rate for MINOS is higher than CPU time increase rate for MPJ. Although we tried 200 jobs at most, we can solve larger instances by using MPJ within acceptable CPU time levels. When we check the change on R with respect to M, we see that it increases for 50 jobs case and decreases for the others. Also, the CPU time required by MPJ is not affected by the level of M. However, CPU time required by MINOS is significantly affected by M. As M is increased for a given N, CPU time required by MINOS is reduced. This is because as we increase the number of machines, the number of equivalent position sets decrease

N	M	R	MINOS CPU	MPJ CPU
50	3	0.000216	6.16	0.03
	6	0.000308	3.62	0.03
	Total	0.000262	4.89	0.03
100	3	0.000298	34.44	0.06
	6	0.000285	16.76	0.05
	Total	0.000291	25.60	0.05
150	3	0.000294	108.88	0.09
	6	0.000266	48.47	0.09
	Total	0.000280	78.67	0.09
200	3	0.000298	292.62	0.14
	6	0.000291	120.43	0.14
	Total	0.000294	206.53	0.14

Table 4.6: Average performance measures for different N and M levels when $\Delta = 0.01$

and this makes the problem easier for MINOS since it copes with less alternatives of job-position set allocations.

Another observation we obtain from the computational results is related to the relative performance of MPJ on different parts of the efficient frontier. In our experiments, we evaluated the R values for 50 efficient solutions except Z_1 and Z_2 for each replication. Let us denote the first found efficient solutions right after Z_1 as Group 1, and so on, then, we have 50 solution groups for each replication. When we compare the average R value for each group, we observe that the average R is decreasing as we go from the first group to the last group as shown in Figure 4.2. For example, R is higher for the total completion time levels closer to Z_1. As we get closer to Z_2, R gets smaller and smaller. This relationship is also shown to be statistically significant by the ANOVA results. We think that this is related to the behavior of the manufacturing cost functions. As we get closer to Z_2, we work on flatter parts of the cost functions where the decision

making based on slope information as in the MPJ algorithm is more reliable.

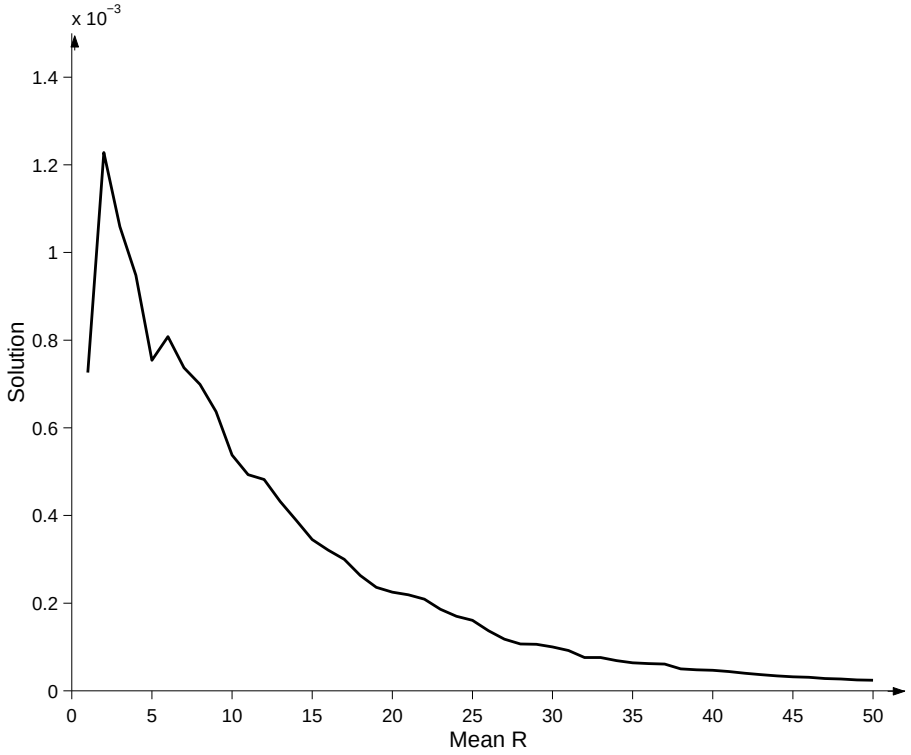

Figure 4.2: Behavior of R on different regions of the efficient frontier

We next analyzed the absolute performance of MPJ and SBA against the global optimal solutions achieved by the MINLP solver BARON 7.2.3 by solving the LSOP models given in Section 4.2. Due to CPU time requirement we could only solve 7 jobs and 10 jobs cases for 3 machines for the same experimental settings. For each replication, we selected 5 efficient points out of the set generated by the MPJ algorithm and solved the LSOP and the relaxed SOP models by the solvers BARON and MINOS, respectively. By this way, we tested these three approaches on 300 points. Table 4.7 shows the relative difference values for the MPJ method versus SBA using BARON, and SBA using MINOS versus SBA using BARON. Results show that both MPJ and MINOS find solutions very close to global optimum. There are cases where MINOS achieves the global optimal solution. We also see from Table 4.7 that when we increase the number of jobs from 7 to 10, the CPU time required by BARON increases by a factor of 140.

Next, we compare the multi-objective performance measure $P(A, B)$ which

N	Measure	Min	Max	Mean	Std. Dev.
7	R(MPJ-BARON)	0.000007	0.009724	0.001373	0.002429
	R(MINOS-BARON)	0.0	0.004609	0.000610	0.001310
	BARON CPU sec.	11.86	29.24	19.48	4.40
10	R(MPJ-BARON)	0.000025	0.004111	0.000693	0.000853
	R(MINOS-BARON)	0.0	0.009073	0.001682	0.002630
	BARON CPU sec.	875.87	2377.93	1412.34	404.34

Table 4.7: Comparison with the global optimal solutions

gives the probability that an algorithm A, gives a better solution than some other algorithm B as discussed in Chapter 3. Table 4.8 includes $P(MPJ, MINOS)$ results for different values of Δ. When $\Delta = 0.01$ we see that probability of decision maker to prefer a solution generated by MPJ is 98% on the average. This is due to the fact that the MPJ method generates significantly higher number of efficient solutions than the GAMS/MINOS solver. When Δ is increased this falls to 82% since pointwise solution quality and number of generated points decreases. The results show that MPJ is a powerful method from the multi-objective optimization point of view.

	$P(MPJ, MINOS)$		
Δ	Mean	Min	Max
0.01	0.975	0.870	0.999
0.03	0.816	0.430	0.970

Table 4.8: Comparison of the approximation algorithms

In this section, we gave the computational results of proposed approaches and discussed the results from several points of view. We saw that by using the optimality properties for the problem we could develop an algorithm which can compete with a commercial software in terms of solution quality but gives better results in terms of computational requirements. This highlights the importance

of extracting and using the problem specific information when dealing with a problem.

4.6 Conclusions

We first considered the single objective problem of minimizing total manufacturing cost subject to a total completion time constraint. For this MINLP problem, we provided an efficient formulation whose relaxed form can be solved to integer by NLP solvers. We showed that NLP solvers can just guarantee to find local optimal solutions for the problem. We also proved optimality conditions that must hold between processing times of jobs at optimality. By the help of these properties, we developed a heuristic algorithm which generates a set of approximate efficient solutions for the problem. Computational results proved that the heuristic algorithm performs as good as well known commercial GAMS/MINOS NLP solver with significantly less computational effort. The discussion we have presented in this chapter was published as Gürel and Aktürk [38]. So far, we have given our results on total completion time objective in single and parallel machines, and total weighted completion time objective in single machine. In the next chapter, we will consider total manufacturing cost and makespan objectives in non-identical parallel machine environment.

Chapter 5

Minimizing Total Manufacturing Cost and Makespan in Non-identical Parallel Machine Scheduling

In this chapter, we consider non-identical parallel CNC turning machines environment. We try to handle a situation where we need to make two decisions at the same time, which are scheduling jobs on the given machines and making appropriate processing time decisions, so as to minimize total manufacturing cost. In practice, when the workload on a machine is high, the time related performance measure (makespan) becomes more important and we need to consider manufacturing cost and makespan objectives at the same time. In this study, we study finding the optimum processing times (equivalently, machining parameters) that will minimize the total manufacturing cost ($\bar{F}$) subject to a makespan C_{max} constraint. We provide optimality properties that will be used in an exact solution method. We also propose a recovering beam search method, and develop an improvement search algorithm for the problem.

In Section 5.1, we give the problem definition. In Section 5.2, we define the single machine subproblem and give a solution method for the problem. In Section 5.3, we prove cost lower bounds for the partial schedules which we will employ in our exact and beam search algorithms. In Section 5.4 we provide a heuristic method which constructs a starting solution for the B&B algorithm. We give the B&B algorithm in Section 5.5. Then, in Section 5.6, we discuss the beam search algorithm for the problem. We proposed an improvement search heuristic in Section 5.7. In Section 5.8, we provide a recovering procedure that improves the beam search algorithm. We computationally test all these methods in Section 5.9 and finally give concluding remarks in Section 5.10.

5.1 Problem Definition

The notation used throughout the chapter is as follows:

Parameters:

J: set of jobs to be processed.

$f_{jm}(p_{jm})$: manufacturing cost function for job j on machine m.

p^l_{jm}: processing time lower bound for job j on machine m.

p^u_{jm}: processing time level at which $f_{jm}(p_{jm})$ takes its minimum value.

C_m : operating cost for CNC turning machine m (\$/min).

H_m : maximum applicable power by CNC turning machine m (hp).

T_j, e_j : tooling cost multiplier and exponent for job j.

Decision variables:

p_{jm}: processing time of job j on machine m.

X_{jm} : decision variable, that controls if job j is assigned to machine m.

In set J, we have N jobs to be processed. Each job corresponds to a metal cutting (turning) operation that will be performed by a given cutting tool on one of the M CNC turning machines. Each job is different in terms of its length and diameter, depth of cut, maximum allowable surface roughness, and its cutting tool type. Also, each machine is different in terms of its maximum applicable cutting power H_m, and its unit operating cost, C_m(\$/min.). Each job must be performed

on a single machine without preemption and each machine can perform one job at a time. We also assume that setup and tool change times are negligible.

In this chapter, we will be using the previously defined convex manufacturing cost function below:

$$f_{jm}(p_{jm}) = C_m p_{jm} + T_j p_{jm}^{e_j}$$

which is minimized at a processing time level p_{jm}^u. Since each job has different manufacturing properties and each machine has a different unit operating cost C_m, manufacturing cost function f_{jm} is different for each job on each machine. Furthermore, there exists a processing time lower bound p_{jm}^l which is determined by the manufacturing properties of job j and the maximum applicable cutting power of machine m, therefore p_{jm}^l is also different for each job on each machine.

The problem is to schedule N jobs on M machines and find the optimum processing time of each job so as to minimize total manufacturing cost ($\bar{F}$) under the constraint that the makespan (C_{max}) of the schedule being upper bounded by a known value of K. The technique used is the ϵ-constraint approach and is written in the γ field of the $\alpha|\beta|\gamma$ scheduling notation as $\epsilon(\bar{F}/C_{max})$. Therefore, our bicriteria problem is presented as $R_m|contr|\epsilon(\bar{F}/C_{max})$ and can be formulated as follows:

$$\min \bar{F} : \sum_{j=1}^{N} \sum_{m=1}^{M} X_{jm} f_{jm}(p_{jm})$$

$$\text{s.t.} \quad \sum_{j=1}^{N} X_{jm} p_{jm} \leq K \qquad m = 1, \ldots, M \tag{5.1}$$

$$\sum_{m=1}^{M} X_{jm} = 1 \qquad j = 1, \ldots, N \tag{5.2}$$

$$p_{jm}^l \leq p_{jm} \leq p_{jm}^u \quad j = 1, \ldots, N,\ m = 1, \ldots, M \tag{5.3}$$

$$X_{jm} \in \{0, 1\}\ j = 1, \ldots, N,\ m = 1, \ldots, M \tag{5.4}$$

In the mixed integer nonlinear programming (MINLP) model above, the objective function is the total manufacturing cost of the jobs, ($\bar{F}$). Constraint set

(5.1) guarantees that the makespan for each machine is less than or equal to K so that the $C_{max} \leq K$ for the schedule. Constraint set (5.2) forces each job to be assigned to a machine. Constraint set (5.3) applies the processing time lower and upper bounds for each job. The objective function is non-convex due to existing $(X_{jm} f_{jm}(p_{jm}))$ terms. Moreover, due to the bilinear terms $(X_{jm} p_{jm})$ in the constraint set (5.1), the feasible space of the problem is also non-convex, so it is a non-convex MINLP model. The well known Outer Approximation and Generalized Benders Decomposition algorithms for MINLP problems are only valid under the convex objective function and convex feasible space assumptions (Floudas [31]). The non-convexities constitute the major difficulty in finding the global optimal solutions in MINLP problems and non-convex MINLP problems receive increasing attention in nonlinear optimization theory (Kesavan et al. [55]). In this chapter, by exploiting the structure of our non-convex MINLP problem, we give an exact algorithm and propose efficient heuristic algorithms for the problem.

The formulation above shows that we have to make two types of decisions to solve the problem, the first one is machine/job allocation decisions and the second one is processing time decisions. For a given machine/job allocation, we can find the optimal processing times by solving the single machine manufacturing cost minimization problem subject to the makespan constraint for each machine. This is a subproblem of our original problem and we denote it as P_m where m stands for machine m. In the next section we give a solution method for P_m.

5.2 Single Machine Subproblem (P_m)

In P_m, we find the optimal processing times for the set of jobs assigned to machine m so that total manufacturing cost is minimized and the makespan for the machine does not exceed K. The problem of finding the optimal processing times on a single machine under a makespan constraint has been studied in the literature. The problem for the case of linear decreasing cost functions is shown to be polynomially solvable by Van Wassenhove and Baker [87]. This result was extended to the case of piecewise linear cost functions by Hoogeveen and Woeginger

[47]. Here, we solve the problem for nonlinear convex cost functions. Bretthauer and Shetty [15] provided optimality properties for the nonlinear resource allocation problem which also applies to our problem. Assuming that J_m is the set of jobs assigned to machine m, the single machine problem for machine m can be formulated as follows:

$$\begin{aligned} & \min \quad \sum_{j \in J_m} f_{jm}(p_{jm}) \\ (P_m) \text{ s.t.} \quad & \sum_{j \in J_m} p_{jm} \leq K & (5.5) \\ & p^l_{jm} \leq p_{jm} \leq p^u_{jm} \quad \forall j \in J_m & (5.6) \end{aligned}$$

In Lemma 5.1, we give an optimality property for P_m. This property is very important since it states the relationship between processing times of the jobs assigned on the same machine in an optimal solution.

Lemma 5.1. (Optimality Property for P_m) *In an optimal solution to the single machine problem P_m, let p^* be the optimal processing times vector, and $J^1_m = \{j : p^*_{jm} > p^l_{jm}\}$ and $J^2_m = \{j : p^*_{jm} = p^l_{jm}\}$ where $J_m = J^1_m \cup J^2_m$. Then the following conditions holds:*

i. $(\partial f_{jm}/\partial p_{jm})(p^*_{jm}) = \lambda$ *for* $j \in J^1_m$.

ii. $(\partial f_{jm}/\partial p_{jm})(p^*_{jm}) \geq \lambda$ *for* $j \in J^2_m$.

Proof. Since makespan is a regular scheduling measure, increasing the processing time of a job will not improve the makespan. Consequently, any processing time value greater than p^u_{jm} will lead to an inferior solution because both the manufacturing cost and makespan of the schedule will get worse. Therefore, we can replace $p^l_{jm} \leq p_{jm} \leq p^u_{jm}$ in constraint set (5.6) with $p_{jm} \geq p^l_{jm}$. Next, we assume that there exists at least one job j on machine m such that $p_{jm} > p^l_{jm}$, then the optimal processing times vector p^* is a regular point. Such a point must satisfy the Karush-Kuhn-Tucker (KKT) conditions. Then the Lagrangian

function for a processing time vector p is as follows:

$$L(p,\lambda,\mu) = \sum_{j\in J_m} f_{jm}(p_{jm}) - \lambda\left(K - \sum_{j\in J_m} p_{jm}\right) + \sum_{j\in J_m} \mu_j(p^l_{jm} - p_{jm})$$

At the optimal solution p^*, $\nabla_p(L_p,\lambda,\mu) = 0$ must be satisfied, then for each $j \in J_m$, the following equation must hold:

$$(\partial f_{jm}/\partial p_{jm})(p^*_{jm}) - \lambda - \mu_j = 0$$

where $\lambda \leq 0$ and $\mu_j \geq 0$ for each $j \in J_m$. If $j \in J^1_m$, then $\mu_j = 0$, so $(\partial f_{jm}/\partial p_{jm})(p^*_{jm}) = \lambda$ which proves part (i) of the lemma. If $j \in J^2_m$, then $(\partial f_{jm}/\partial p_{jm})(p^*_{jm}) = \lambda + \mu_j$. Since $\mu_j \geq 0$, $(\partial f_{jm}/\partial p_{jm})(p^*_{jm}) \geq \lambda$ holds, which proves part (ii) of the lemma. The only case that $J^1_m = \emptyset$ holds for an optimal solution is the case in which $K = \sum_{j\in J_m} p^l_{jm}$. Then, the optimal solution is $p^*_{jm} = p^l_{jm}\ \forall j \in J_m$ which falls into the case described in part (ii) of the lemma. □

At the optimal solution, the value of λ in Lemma 5.1 gives us the change rate of the optimal manufacturing cost if K is increased. Property in Lemma 5.1 implies that at optimality, we cannot improve the manufacturing cost by changing the processing times of the jobs. We know that $\lambda \leq 0$ always holds since $f_{jm}(p_{jm})$ is non-increasing in the interval $[p^l_{jm}, p^u_{jm}]$ for all j and m. An immediate extension of this property for P_m to non-identical parallel machines case is as follows:

Corollary 5.1. For the $R_m|contr|\epsilon(\bar{F}/C_{max})$ problem, an optimal solution must satisfy the property in Lemma 5.1 for each machine, individually.

For the non-identical parallel machines problem, property in Lemma 5.1 must hold for each machine m individually so we need to solve the subproblem P_m for each machine. The optimality property in Lemma 5.1 allows us to solve the problem P_m. However, since each job may use a different tool type we may have different e_j's for each job. Then, it is not possible to derive a closed form expression to determine the optimal processing time p^*_{jm}. Therefore, in order to

solve the problem we need to employ a line search algorithm, which will search over the Lagrangian dual variable of the makespan constraint, λ. According to Lemma 5.1, for each value of λ we can determine a corresponding processing time for each job and a corresponding makespan level. Then, given a makespan level K, we can search over possible values of λ to achieve the makespan level K and corresponding optimal processing times.

P_m **algorithm:**

Step 1. Set the upper bound for λ, $\lambda^u = 0$.

Step 2. Set the lower bound for λ, $\lambda_l = \min_j\{(\partial f_{jm}/\partial p_{jm})(p^l_{jm})\}$.

Step 3. Calculate corresponding makespan levels $K^u = \sum_{j\in J_m} p^u_{jm}$ and $K^l = \sum_{j\in J_m} p^l_{jm}$.

Step 4. If $K^l > K$, then the problem is infeasible.
Else if $K^u \leq K$, then $p^*_{jm} = p^u_{jm}$ for all $j \in J_m$.
Else, apply the bisection method (Bazaraa et al. [11]) over all possible values of λ in $[\lambda^l, \lambda^u]$ in order to find a solution which has a makespan level within ϵ-neighborhood of K.

P_m algorithm is pseudo-polynomial since it applies the bisection method. We will use P_m algorithm in the B&B algorithm, beam search, and improvement search heuristics. In the next section, we will prove useful properties on cost lower bounds for partial schedules. Optimality property in Lemma 5.1 will also be useful for proving these properties.

5.3 Cost Lower Bounds for a Partial Schedule

In a partial schedule, denoted as S_p, a subset of jobs (J^p) is assigned to machines, but the remaining jobs are not yet assigned. For S_p, we denote the set of jobs assigned to machine m as J^p_m and assume that optimal p_{jm} decisions were made

by solving the P_m for the currently scheduled jobs on machine m. Considering an arbitrary S_p, we will prove lower bounds for the manufacturing costs of complete schedules achievable by adding the unscheduled jobs to S_p. We assume that when we add an unscheduled job to S_p, processing times of previously scheduled jobs may change, but the machine/job assignments in S_p stay the same. Since the processing time decisions for the jobs in J^p were made previously by solving the subproblem P_m for each machine m, we have at hand the optimal dual price λ_m for each machine m. When we add an unscheduled job j to a machine m of the partial schedule S_p, it is sure that the new schedule will have a higher manufacturing cost. Assume that adding an unscheduled job j to machine m does not violate the makespan constraint (5.1), i.e., $\sum_{i \in J_{pm}} p^l_{im} + p^l_{jm} \leq K$. Then, we give a lower bound for the cost increase that will occur by adding a single job j to a machine m of S_p in Lemma 2.

Lemma 5.2. (Cost Lower Bound for adding job j to machine m (lb_{jm})) *A lower bound, lb_{jm}, on the cost change that occurs by adding job j to machine m can be found as follows:*

$$lb_{jm} = f_{jm}(p^{ub}_{jm}) - \lambda_m \cdot p^{ub}_{jm}$$

where $p^{ub}_{jm} = \max(p^l_{jm}, (\partial f_{jm}/\partial p_{jm})^{-1}(\lambda_m))$.

Proof. Suppose that we first set the processing time of job j to $p^{ub}_{jm} = \max(p^l_{jm}, (\partial f_{jm}/\partial p_{jm})^{-1}(\lambda_m))$ and then add the job to machine m. p^{ub}_{jm} is the processing time that satisfies the optimality property (Lemma 5.1) for machine m. It is obvious that optimal processing time p^*_{jm}, to be achieved after solving P_m, cannot be higher than p^{ub}_{jm}, therefore we will definitely incur an additional cost of at least $f_{jm}(p^{ub}_{jm})$. The new schedule on machine m may be infeasible since the makespan of the jobs on machine m, $\sum_{i \in J^p_m} p_{im} + p^{ub}_{jm}$, may exceed K. In such a case, the jobs on machine m must be compressed to make the schedule feasible. We will estimate the cost of compressing the jobs on machine m to K. The marginal cost of decreasing the makespan of the schedule is $-\lambda_m$ and we need to compress the jobs by p^{ub}_{jm}. Then, the second term is $-\lambda_m \cdot p^{ub}_{jm}$ which is

a lower bound on the compression cost to be incurred. Then, the additional cost of the new schedule achieved will be at least lb_{jm}. □

Lemma 5.2 gives us a lower bound for the cost of adding an unscheduled job to a specified machine. We want to determine a cost change lower bound for adding all unscheduled jobs to S_p. By using the lb_{jm} values for $j \in J \setminus J^p$, we can formulate an integer program (IP) that gives us a lower bound on the manufacturing cost increase due to adding all unscheduled jobs in $J \setminus J^p$ to S_p. A lower bound for the cost increase to be caused by adding all unscheduled jobs, or equivalently forming a complete schedule, can be found by solving the following integer program:

$$\min \sum_{j \in J \setminus J^p} \sum_{m=1}^{M} X_{jm} lb_{jm}$$

$$(IP) \text{ s.t.} \sum_{j \in J \setminus J^p} X_{jm} p^l_{jm} \leq K - \sum_{j \in J_{pm}} p^l_{jm} \qquad m = 1, \ldots, M \quad (5.7)$$

$$\sum_{m=1}^{m=M} X_{jm} = 1 \qquad j \in J \setminus J^p \quad (5.8)$$

$$X_{jm} \in \{0, 1\} \qquad j \in J \setminus J^p \text{ and } m = 1, \ldots, M \quad (5.9)$$

In the IP model above, the objective function is the sum of the cost change lower bounds (lb_{jm}) for the possible assignments of unscheduled jobs to the machines. Constraint set (5.7) is the makespan constraint that guarantees that sum of the processing time lower bounds of jobs assigned to a machine does not exceed K. Constraint set (5.8) assigns each unscheduled job to a machine. Finally, there exists binary constraints (5.9) for the decision variable X_{jm}. In the following lemma, we prove that the IP model gives a lower bound for S_p.

Lemma 5.3. **(LB_{IP})** *For a partial schedule S_p, an optimal solution of the IP gives a lower bound for the cost increase in order to form a complete schedule.*

Proof. As discussed in Lemma 5.2, lb_{jm} gives a cost increase lower bound of adding job j to machine m of S_p. The IP model looks for a feasible machine/unscheduled job assignment that gives the minimum sum of lb_{jm}'s, given

that each unscheduled job is assigned to a machine and makespan constraint is satisfied for each machine. A complete schedule achievable from S_p will obviously have one of the feasible machine/job assignments of the IP model. Then, forming any complete schedule will bring more cost than the optimal value of the IP model. □

We denote the lower bound found by solving the IP model as LB_{IP}. If an IP for S_p turns out to be infeasible, this means no complete schedule can be achieved from S_p. Next property gives the computational complexity of the IP problem.

Lemma 5.4. ($\mathcal{NP}$-hardness) *Solving the IP model is $\mathcal{NP}$-hard.*

Proof. As our IP model is looking for an optimal machine/job assignment subject to makespan constraints, it is a Generalized Assignment Problem (GAP) model. Then, finding a cost increase lower bound by solving IP is $\mathcal{NP}$-hard. □

Since we will need to find a cost increase lower bound for each partial schedule (node) in our B&B algorithm, solving an $\mathcal{NP}$-hard problem for each node, of course, may not be efficient in terms of computation time. Therefore, we propose two other practical methods. The first one is to solve the LP relaxation of the IP model. The following lemma is obvious:

Lemma 5.5. (LB_{LP}) *An optimal solution of the LP relaxation of IP gives a cost increase lower bound for S_p.*

A lower bound (LB_{LP}) to be found by the LP relaxation is obviously smaller than LB_{IP}. However, it requires much less computation time which is critical for our B&B algorithm. Another approach to find a lower bound for S_p would be relaxing the makespan constraint (5.7) of IP. In such a case we still get a lower bound for the cost increase, which is denoted as LB_R, and given below:

Lemma 5.6. (LB_R) *A lower bound for the cost increase required to form a complete schedule from S_p is given by: $LB_R = \sum_{j \in J \setminus J^p} \min_m lb_{jm}$, where lb_{jm} is*

the cost increase lower bound of adding job j to machine m of S_p as defined in Lemma 5.2.

Proof. The lower bound on the additional cost to be incurred by adding a job j to S_p (without knowing the machine to which it will be assigned) can be found by $\min_m lb_{jm}$. When we have multiple unscheduled jobs, we can find an overall lower bound by summing the corresponding lower bounds for all unscheduled jobs. □

The lower bound LB_R assumes that each job can be assigned to the machine that gives the best cost change lower bound for it regardless of the makespan. Obviously, such a machine/job assignment may be infeasible due to constraint set (5.7), but computing LB_R is much more simpler than LB_{IP} or LB_{LP}. Between three given lower bounding methods, it is easy to see the relationship given below:

Lemma 5.7. $LB_R \leq LB_{LP} \leq LB_{IP}$.

Computational requirements will also have the same relationship. In order to achieve a better lower bound we need to solve a harder problem. In this section, we proposed three methods to find a cost increase lower bound for an arbitrary partial schedule S_p. In the next section, we describe a construction heuristic to find an initial solution for the B&B algorithm.

5.4 Initial Solution

In order to find an initial solution for the problem, we propose a heuristic algorithm denoted as IS. This initial solution will serve as an upper bounding solution for our B&B algorithm. The IS algorithm starts with a list of jobs where jobs were ordered in ascending order of their minimum cost ($\min_m f_{jm}(p^u_{jm})$). Then, starting with the first job in the list and all machines being empty at the beginning, in each iteration, the algorithm adds a new job to the schedule. For each job, the algorithm first selects a suitable machine which gives the minimum cost

increase lower bound as discussed in Lemma 5.2, and then adds the job to that machine by solving the subproblem P_m for the machine.

IS algorithm

Step 1. List the jobs in ascending order of their $min_m f_{jm}(p^u_{jm})$.

Step 2. Starting from the first job in the list, for each job do Steps 3 to 5.

Step 3. Calculate lb_{jm} for each machine m, and choose the best machine: $m' = \arg\min_m lb_{jm}$.

Step 4. Check the feasibility of assigning job j to machine m'. If it is not feasible, choose the next best machine and update m'. Repeat this until finding a suitable machine or finding out that no machine is feasible. If no feasible machine exists, then stop.

Step 5. Assign job j to machine m' and determine the optimal processing times by solving the single machine subproblem P_m for machine m'.

IS algorithm schedules the minimum cost job first, so it is a greedy approach in a sense. At each iteration, a new job is scheduled on the machine which gives the minimum cost increase lower bound given in Lemma 5.2. IS algorithm either ends with a feasible schedule for the problem or fails to find a feasible schedule and stops. The algorithm performs at most $N \times M$ iterations. In the next section, we will give the B&B algorithm.

5.5 B&B Algorithm

In this section, we first explain the B&B search tree. Then, we discuss the node elimination rules. We next give a step-by-step description of the algorithm.

Search Tree

At the root node of the search tree at level 0, all jobs are unscheduled. Each node in the search tree corresponds to an assignment where the jobs in a subset

of J are assigned to the machines. At each level of the search tree, the B&B algorithm assigns an unscheduled job. Then, a node at level k corresponds to a partial schedule with k jobs being assigned to the machines and similarly a node at level N corresponds to a complete schedule where all jobs in J are scheduled.

The algorithm uses a job list $(j_1, \ldots, j_N)$ to assign job j_k in the k^{th} level of the tree. The root node has M child nodes: one distinct node for scheduling job j_1 to each machine m for $m = 1, \ldots, M$. Then, each node at level 1 corresponds to an assignment where job j_1 is assigned on a different machine. Similarly, a node at level k corresponds to a partial schedule with jobs $(j_1, \ldots, j_k)$ assigned on the machines. Each node at level $k < N$ has at most M child nodes so that there is one child node for the assignment of job j_{k+1} to machine m, for $m = 1, \ldots, M$.

For each node, we find the optimal cost and the optimal processing time decisions for each machine by solving the subproblem P_m for the given machine/job assignments of the partial schedule. This will allow us to use the lower bounding methods discussed in Lemmas 2-6 so that we will be able to reduce the tree size by fathoming some parts of the tree. Obviously, by traversing the search tree defined above, we can find an optimal solution for the problem.

Node Elimination

Having a search tree that enumerates all possible solutions for the problem and finds an optimal schedule, the question is how to reduce the size of the search tree by discarding nodes. There are two ways of eliminating nodes from our B&B tree. One is by feasibility and the other is by optimality. As discussed above, we generate a child node from a parent node by adding a new job j to a machine m of the partial schedule represented by the parent node. When opening a child node, we first check if it is feasible to add job j to machine m. If the child node is not feasible ($\sum_{i \in J_m} p^l_{im} + p^l_{jm} > K$), we eliminate the child node, so that ignore all subtree growing from that child node.

If a child node turns out to be feasible, then we solve the single machine subproblem P_m for machine m and make optimal processing time decisions for the partial schedule so that Lemma 5.1 is satisfied for the jobs on machine m. After checking the feasibility of the node and solving the subproblem for machine m,

the next step is to find a lower bound for the new partial schedule. In Section 5.3, we derived three different ways of calculating manufacturing cost increase lower bounds for achieving complete schedules from a given partial schedule. We can employ one of those methods (LB_{IP}, LB_{LP}, LB_R) to find a cost increase lower bound. The cost increase lower bound of the node plus the cost of the partial schedule of the node itself gives us a lower bound for a complete schedule achievable from the node. If the lower bound of a node is higher than or equal to the cost upper bound, then the node is eliminated due to optimality.

Another alternative for eliminating nodes by feasibility is detecting the infeasibility of a partial schedule when finding its lower bound by using LB_{IP} or LB_{LP}. When solving the IP model (or its LP relaxation) for finding LB_{IP} (LB_{LP}), the solver may find out that the model is infeasible. This shows that no feasible complete schedule can be achieved from the considered partial schedule. Then, we eliminate the node by feasibility. Having described the search tree and the ways of eliminating nodes from the tree we give a stepwise presentation of our B&B algorithm below:

B&B Algorithm

Step 1. Find an initial upper bounding solution by using the IS algorithm. Set upper bounding cost UB^c to the cost of the solution found by IS. If IS cannot find a feasible solution, set $UB^c = \infty$.

Step 2. Form a list of jobs $(j_1, \ldots, j_N)$ in descending order of $(\max_m p^l_{jm})$.

Step 3. Start with the root node as the parent node, set the level of the parent node $level_p$ to 0.

Step 4. For $i = 1, \ldots, m$, do the following:

Step 4.1. Generate child node i of the parent node by adding job j_{level_p+1} to machine i.

Step 4.2. Check the feasibility of child node i. If child node i is not feasible, eliminate it.

Step 4.3. Else, solve the subproblem P_m for machine i and calculate cost (F^p) of the partial schedule.

Step 4.4. If child node i is a complete schedule, i.e. $level_p + 1 = N$, then, if $F^p < UB^c$, set $UB^c = F^p$. Else, apply Steps 4.5 and 4.6.

Step 4.5. Find the cost increase lower bound LB^p. If it turns out that no complete feasible schedule can be achieved from child node i, then eliminate it. Else, calculate lower bound for child node i by $LB^C = F^p + LB^p$.

Step 4.6. If $LB^c \geq UB^c$ eliminate child node i.

Step 5. Find the next parent node and update $level_p$ and go to Step 4. If no parent node is available, then stop.

In the B&B algorithm, Step 1 finds an initial schedule to be an upper bounding solution by using the IS algorithm. In Step 2, jobs are ordered to form a list which determines which job to be scheduled at what level of the B&B search tree. Step 3 sets the root node as the first parent node to be considered in the following steps. Step 4 and its sub-steps branches on the parent node and generates its child nodes. At each child node, job j_{level_p+1} is added to a different machine of the partial schedule represented by the parent node. Step 4.2 checks the feasibility of adding job j_{level_p+1} to machine i and if not feasible eliminates the node. If it turns out to be feasible, in Step 4.3, single machine subproblem P_m is solved for machine i. If child node i represents a complete schedule and if $F^p < UB^c$, then the schedule on child node i is the best solution found so far and the UB^c value is updated. If child node i represents a partial schedule, a lower bound is calculated for the partial schedule in Step 4.5. At this step, we may conclude that no feasible complete solution can be achieved from this partial schedule, then we eliminate this node. In Step 4.6, if the lower bound found in step 4.5 is greater than UB^c, then we eliminate child node i. In Step 5, we either find a new parent node or stop.

In the B&B algorithm, we implemented a modified depth-first strategy. When we branch on a parent node, we generate all its child nodes and out of these child nodes we select the one with the minimum lower bound as the new parent node

	Machine 1				Machine 2				Cost Coef.	
Job	p_{j1}^l	$f_{j1}(p_{j1}^l)$	p_{j1}^u	$f_{j1}(p_{j1}^u)$	p_{j2}^l	$f_{j2}(p_{j2}^l)$	p_{j2}^u	$f_{j2}(p_{j2}^u)$	T_j	e_j
0	0.66	4.21	1.53	2.70	0.31	10.24	1.14	4.00	2.05	-1.32
1	1.15	1.95	1.21	1.94	0.48	4.29	0.93	2.99	1.00	-1.64
2	0.20	0.63	0.31	0.56	0.08	1.47	0.22	0.82	0.06	-1.22
3	0.22	0.98	0.43	0.77	0.09	2.44	0.31	1.12	0.12	-1.22

Table 5.1: Job data for numerical example

and branch on this node next. If the complete subtree growing from the selected parent node is traversed, we branch on the next best child node of its parent node. This is a depth-first strategy supported with a greedy node selection approach.

Numerical Example

Next, we illustrate the B&B algorithm on a numerical example. We consider a 4 jobs 2 machines example. Machine 1 has a unit operating cost $C_1 = 1\$/min.$ and maximum applicable cutting power $H_1 = 5hp.$ and machine 2 has $C_2 = 2\$/min.$ and $H_2 = 10hp.$ The upper limit K on the makespan objective is 1.3. Each of four jobs has different length, diameter, depth of cut and surface roughness requirement and each requires a different cutting tool type. The resulting processing time bounds and tooling cost coefficients are given in Table 5.1.

The B&B algorithm first (Step 1) finds an initial upper bounding solution by applying the IS algorithm. IS algorithm orders the jobs by their minimum costs (Step 1) so the job order is 2, 3, 1, 0. Both machines are empty at this stage, so $lb_{21} = 0.56$ and $lb_{22} = 0.82$ (Step 2), then job 2 is scheduled on machine 1 and $p_{21} = p_{21}^u = 0.31$ (Step 5). Next job to schedule is job 3. Since, $lb_{31} = 0.77$ and $lb_{32} = 1.12$, job 3 is scheduled on machine 1 and $p_{31} = 0.43$. Makespan on machine 1 is 0.77 and cost of the partial schedule is 1.33. Next job to consider is job 1, $lb_{11} = 1.94$ and $lb_{12} = 2.99$, but since $p_{11} = 1.15$, it is not feasible to schedule job 1 on machine 1 but it can be scheduled on machine 2 (Step 3). Since $p_{12}^u = 0.93 < 1.3$, $p_{12} = 0.93$ and the makespan on machine 2 is 0.93 and cost is

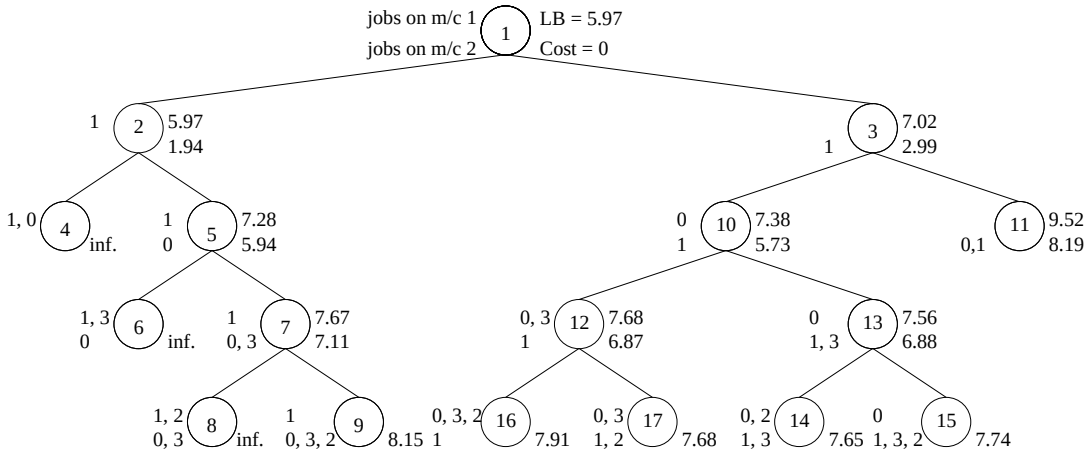

Figure 5.1: B&B tree for the numerical example

2.99. Finally, job 0 is to be scheduled. $lb_{01} = 2.70$ and $lb_{02} = 4.00$, so machine 1 is better and it is also feasible to schedule job 0 to machine 1. Solving the single machine subproblem P_m for machine 1, we get $p_{21} = 0.20$, $p_{31} = 0.232$ and $p_{01} = 0.868$ with corresponding manufacturing costs 0.63, 0.95 and 3.34, respectively. For the optimal solution of P_m, $\lambda = -2.75$. The solution achieved by IS has a total cost of 7.91 which is the initial upper bound UB^c to be used in B&B.

In Step 2 of B&B algorithm, we order the jobs in descending order of p^l_{jm}'s, so the job order is 1, 0, 3, 2. We form the root node with no jobs scheduled on it. Its cost is 0 and lower bound is $LB_R = 5.97$. We give the search tree traversed for this numerical example in Figure 5.1. Node 1 (root node) being the parent node, we first generate the child node 2. At node 2, we assign job 1 (the first job in the list) to machine 1 (Step 4.1). This is a feasible assignment (Step 4.2) and the processing time of job 1 is 1.21 (Step 4.3) and its cost is 1.94. The lower bound for node 1 is 5.97 (Step 4.5). Similarly, we form node 3 by assigning job 1 to machine 2. Cost of node 3 is 2.99 and lower bound is 7.02. Therefore, next parent node to be branched is node 2. From node 2, we form node 4 which corresponds to an infeasible schedule since jobs 1 and 0 cannot be assigned on machine 1 due to makespan constraint ($p^l_{11} + p^l_{01} = 1.81 > 1.3$). Then, we eliminate the subtree that would grow from node 4. We form node 5 by scheduling job 0 to machine 2. Node 5 is the next parent node and we first form nodes 6 from this node. Node

6 turns out to be infeasible.

We next form node 7 from node 5. We can illustrate the lower bound calculation on node 7. Cost of node 7 is 7.11, and $\lambda_1 = 0$ and $\lambda_2 = -0.56$. Then, $(\partial f_{21}/\partial p_{21})^{-1}(\lambda_1) = 0.31$, so $p_{21}^{ub} = 0.31$. $lb_{21} = f_{21}(0.31) = 0.56$. Similarly, $(\partial f_{22}/\partial p_{22})^{-1}(\lambda_2) = 0.20$ and $p_{22}^{ub} = 0.20$. Therefore, $lb_{22} = f_{22}(0.20) + 0.56 \times 0.20 = 0.94$. Then, $LB_R = \min\{0.56, 0.94\} = 0.56$ and cost lower bound for node 7 is its cost plus LB_R, which is $7.11 + 0.56 = 7.67$. If we used the lower bound LB_{IP}, we would get a higher lower bound since LB_{IP} takes the makespan constraint into account. Then, $LB_{IP} = 7.11 + 0.94 = 8.05$ in which case we would eliminate the subtree of node 7 due to lower bound ($8.05 > 7.91$) and we would not need to open the nodes 8 and 9. Similarly, if we used LB_{LP}, lower bound for node 7 would be $7.11 + 0.76 = 7.87$.

Next parent node to be branched on is node 7 and we achieve an infeasible child node (node 8) and a complete schedule child node (node 9). Cost of node 9 is 8.15, since $8.15 > 7.91$, we do not update UB^c (Step 4.4). Since, we processed the entire subtree grown from node 2, next parent node is node 3. One of the child nodes of node 3 is node 11. Node 11 has a lower bound of 9.52 which is higher than $UB^c = 7.91$, so we eliminate the subtree of node 11 due to optimality. Node 10 is the next parent node and we generate the subtree of node 10. Order of generation is same as the label order (12, 13,...,17). We achieve the optimal solution at node 14. In the optimal solution, jobs 0 and 2 are scheduled on machine 1 and jobs 1 and 3 on machine 2. Optimal processing times are as follows: $p_{01} = 1.09$, $p_{21} = 0.21$, $p_{12} = 0.93$ and $p_{32} = 0.31$. Makespan on machine 1 and 2 are 1.3 and 1.24 with manufacturing costs of 3.53 and 4.11, respectively.

The B&B algorithm defined above either ends up with an optimal solution or concludes that the problem is infeasible. The performance of this B&B algorithm is bounded by the computational requirements. In the next section we propose a beam search algorithm for the instances where applying B&B is inefficient.

5.6 Beam Search Algorithm (BS)

Up to now we have described an exact algorithm (B&B) for the problem. We have also proposed lower bounding methods to reduce the tree size for this algorithm. However, the problem is an $\mathcal{NP}$-hard problem and the size of the search tree for the B&B algorithm increases exponentially as N and M increase. For higher levels of N, M and K, we propose a beam search algorithm to find near optimal solutions. A well known beam search application on a scheduling problem is Ow and Morton [71]. They applied beam search on single machine early/tardy problem.

Beam search is a fast B&B method which keeps best b (beam width) nodes at a level of search tree and eliminates the rest. Therefore, its running time is polynomial in the problem size. In our BS algorithm, we will consider the same search tree structure as our B&B algorithm. Therefore, each node at a higher level than N will correspond to a partial schedule and at each level a new job from a list of jobs will be scheduled. As in B&B method, our beam search generates M nodes out of the root node at level 0. At level 1, beam search keeps the "most promising " b nodes and eliminates the others. Then, using the selected b nodes it generates child nodes at level 2 and again keeps b of them and eliminates the others. This is done until complete schedules are achieved at the leaves of the tree. The most critical aspect of a beam search algorithm is how it finds the most promising nodes. One approach is evaluating each node by a simple heuristic which finds an estimate of the total cost of the best solution available starting with the partial solution represented by that node. Hence, in our BS algorithm, in order to evaluate a node, we will use the lower bound LB_{LP}. At each level, b nodes with smallest LB_{LP} will be kept and the others will be eliminated. Each lower bounding method we defined in Section 5.3 can be used as an evaluation method which estimates the cost of complete schedule for a given partial schedule. A stepwise description of the beam search algorithm for the problem is below:

BS Algorithm

Step 1. Form a list of jobs $(j_1, \ldots, j_N)$ in descending order of $(\max_m p^l_{jm})$.

Step 2. Start with the root node as the parent node. set the level of the parent node $level_p$ to 0.

Step 3. For each selected parent node and for $i = 1, \ldots, m$, do Step 3.1 to 3.4:

Step 3.1. Generate child node i of the parent node by adding job j_{level_p+1} to machine i.

Step 3.2. Check the feasibility of child node i. If child node i is not feasible, eliminate it.

Step 3.3. Else, solve the subproblem P_m for machine i and calculate cost of the partial schedule F^p.

Step 3.4. If $level_p < (N-1)$, then find the cost change lower bound LB^p. If it turns out that no complete feasible schedule can be achieved from child node i, then eliminate it. Else, calculate the lower bound for node i by $LB^C = F^p + LB^p$.

Step 4. If all child nodes are eliminated due to feasibility, then no feasible solution could be found, stop.

Step 5. If $level_p < (N-1)$, then select best b child nodes with smallest LB^C values, set $level_p = level_p + 1$ and go to Step 2.

Step 6. If $level_p = (N-1)$, then select the best child node with minimum cost and stop.

If the problem is feasible, the BS algorithm either cannot find a feasible solution (Step 4) or finds an approximate optimal solution (Step 6). If the problem is infeasible, the BS algorithm cannot find a feasible solution (Step 4). Using our search tree structure that we proposed in Section 5.5 and our lower bounding methods as an evaluation function, the BS algorithm is a fast alternative for the cases that the B&B algorithm fails to undertake. The time complexity of the BS algorithm is $O(mnb)$. In the next section, we will propose improvement search steps for the problem which can be applied to any given schedule.

5.7 Improvement Search Heuristic (ISH)

We have given an exact algorithm (B&B) for the problem and then proposed a beam search method that runs in polynomial time. In this section, we extend our discussion to an improvement search algorithm. We will define an improvement search heuristic which starts with an initial schedule and improves the solution at each iteration to achieve a local optimal solution.

Our improvement search heuristic starts with an initial schedule which satisfies the optimality condition in Corollary 5.1 so that we assume the single machine subproblem is solved for each machine. We represent such a solution as a partition of the jobs to the machines. We define two moves to describe the neighborhood of a solution. The first one is 1-move, which is to move a job j from its current machine m_1 to another machine m_2. The other one is 2-swap, which is to exchange job j_1 on machine m_1 with another job j_2 on machine m_2. Given a solution, we proved cost change lower bound properties for the two moves we defined above. Lemma 5.8 gives the cost change lower bound for a 1-move:

Lemma 5.8. (Lower Bound for a 1-move) *Given a schedule which satisfies the condition in Corollary 5.1, assume that job j has a processing time p_{jm_1} on machine m_1 and machines m_1 and m_2 have the dual price values λ_{m_1} and λ_{m_2}, respectively. Then, a lower bound for the cost change that will result by moving job j from machine m_1 to m_2 is as below:*

$$LB(j : (m_1 \rightarrow m_2)) = \lambda_{m_1} p_{jm_1} - f_{jm_1}(p_{jm_1}) + f_{jm_2}(p^{ub}_{jm_2}) - \lambda_{m_2} p^{ub}_{jm_2}$$

where $p^{ub}_{jm_2} = \max((\partial f_{jm_2}/\partial p_{jm_2})^{-1}(\lambda_{m_2}), p^{l}_{jm_2})$.

Proof. Suppose that we first remove job j from machine m_1, the cost change lower bound for this action is the first two terms of the lower bound expression above. The first term is the lower bound for cost change to occur by expanding the processing times of the remaining jobs on machine m_1 and the second one is the cost of job j on machine m_1. Suppose that we next add job j to machine

m_2. The cost change lower bound for adding job j to m_2 can be calculated as discussed in Lemma 5.2. The cost change lower bound for this action is third and fourth terms of the lower bound expression. □

Lemma 5.8 can help us to decide to make a 1-move or not. Since $LB(\Delta_{1-move})$ is a lower bound, if it is a positive value for a particular 1-move, then it is sure that the move will make the cost objective worse, so we ignore the move since it is non-improving. Else, if it is negative, then it promises a reduction in cost but since it is a lower bound it does not guarantee a reduction. Hence, we need to try the move on the schedule and see if it improves. Next, we analyzed the cost change lower bound for 2-swap moves. Lemma 5.9 gives a lower bound for the resulting cost change for this move.

Lemma 5.9. (Lower Bound for a 2-swap) *Given a schedule which satisfies the condition in Corollary 5.1, assume that jobs j_1 and j_2 have processing times $p_{j_1m_1}$ and $p_{j_2m_2}$ and scheduled on machines m_1 and m_2, respectively. Machines m_1 and m_2 have the dual prices λ_{m_1} and λ_{m_2}, respectively. Then, a lower bound for the cost change that will result by moving job j_1 from machine m_1 to m_2 and job j_2 in opposite way is as below:*

$$\begin{aligned} LB(j_1 \leftrightarrow j_2) = \quad & \lambda_{m_1}(p_{j_1m_1} - p^{ub}_{j_2m_1}) - f_{j_1m_1}(p_{j_1m_1}) + f_{j_2m_1}(p^{ub}_{j_2m_1}) \\ & + \lambda_{m_2}(p_{j_2m_2} - p^{ub}_{j_1m_2}) - f_{j_2m_2}(p_{j_2m_2}) + f_{j_1m_2}(p^{ub}_{j_1m_2}) \end{aligned}$$

where $p^{ub}_{j_1m_2} = \max((\partial f_{j_1m_2}/\partial p_{j_1m_2})^{-1}(\lambda_{m_2}), p^{l}_{j_1m_2})$
and $p^{ub}_{j_2m_1} = \max((\partial f_{j_2m_1}/\partial p_{j_2m_1})^{-1}(\lambda_{m_1}), p^{l}_{j_2m_1})$.

Proof. The proof can be easily done by using Lemma 5.8. □

If the cost change lower bound for a move is nonnegative, then it is sure that the move cannot improve the cost. If it is negative, we call the move as a "promising" move. A promising move may improve the cost, but since we just have a

negative lower bound for the cost change, the real cost change after implementing the move may still be positive. Using this fact the proposed algorithm could only evaluate the promising moves which will make it computationally more efficient than a local search algorithm that tries all possible moves. Moreover, the lower bounds presented in Lemmas 5.8 and 5.9 will guide any search algorithm to try the most promising move first, like a steepest descent algorithm in some sense. Given an initial schedule, by calculating the cost change lower bounds for all possible 1-moves and 2-swaps, we can either conclude that the schedule is locally optimal, which is the case when all lower bounds are nonnegative, or we can try the moves which promise possible improvements since they have negative cost change lower bounds. By using these observations, we will propose an improvement search heuristic for the problem.

The improvement search heuristic starts with an initial schedule. First, the heuristic uses promising 1-moves to improve the initial schedule. To do this, it generates all possible 1-moves for this schedule and calculates the cost change lower bound for each possible 1-move. The heuristic applies the most promising move first and solves the single machine subproblems for the affected machines. If an improvement is achieved, new moves are generated for the new schedule. If no improvement is achieved by this move, the heuristic tries the next most promising move, until an improvement is achieved or no promising move is left. When no improvement is possible for the current schedule by using 1-moves, the heuristic considers 2-swap moves. It tries to improve the solution by 2-swap moves in the same way as we did by 1-moves and stops when no improvement is possible.

Improvement Search Heuristic (ISH):

Step 1. Take an initial schedule (S) and its cost is $F(S)$.

Step 2. Generate all 1-moves for S and calculate $LB(j : (m_1 \rightarrow m_2))$ for each 1-move.

Step 3. If no promising moves exist, go to Step 5. Else, find the most promising move.

Step 4. Apply the selected move on S and solve the P_m subproblem for the affected machines. The new solution is $F(S')$.

Step 4.1. If the solution is improved, replace S with S' and go to Step 2.

Step 4.2. Else, find the next most promising move and go to Step 4. If no promising move is found go to Step 5.

Step 5. Generate all 2-swap moves for S and calculate $LB(j_1 \leftrightarrow j_2)$ for each 2-swap move.

Step 6. If no promising moves exist, terminate. Else, find the most promising move and go to Step 7.

Step 7. Apply the selected move on S and solve the P_m subproblem for the affected machines. The new solution is $F(S')$.

Step 7.1. If the solution is improved replace S with S' and go to Step 5.

Step 7.2. Else, find the next most promising move and go to Step 7. If no promising 2-swap move is found, terminate.

By using 2-swap moves defined for ISH, in the next section we will extend the BS algorithm to a recovering beam search algorithm.

5.8 Recovering Beam Search (RBS)

Recovering beam search is first proposed by Della Croce et al. [29]. Recovering beam search algorithm combines beam search with local search techniques to improve the performance of classic beam search. The approach is to apply local search techniques to the partial solutions selected by the beam at each level in order to achieve better partial solutions. This local search step is called the recovering step. The idea is to prevent the elimination of good nodes (nodes that could lead to optimal or near optimal solutions) due to errors in the node evaluation step of beam search algorithms. A recovering beam search algorithm for

the single machine total completion time problem with release dates is provided by Della Croce and T'kindt [28].

In Step 3 in BS algorithm, we generate child nodes for a given level of the search tree. Assuming that K child nodes generated at level l, our recovering step is as follows:

Recovering Procedure:

Step 1. Sort the child nodes in non-decreasing order of their LB^C. Let n_k be the k-th best node.

Step 2. Define the set of selected nodes, empty set $S = \{\}$.

Step 3. Set $k = 1$. While $(|S| < b)$ and $(k < K)$ do

Step 3.1. Generate new partial schedules (nodes) by swapping last added job j_{l+1} with each job added to a different machine.

Step 3.2. If a node (n_k^*) with smaller LB^C is found, and if $n_k^* \notin S$ then $S = S \cup \{n_k^*\}$. Else $S = S \cup \{n_k\}$.

We put this recovering procedure into Step 5 of BS algorithm as below:

Step 5. If $level_p < (N-1)$, then apply Recovering Procedure to select b nodes, set $level_p = level_p + 1$ and go to Step 2.

The time complexity of the RBS algorithm is $O(mn^2b)$. In the next section, we give the results of our computational study.

5.9 Computational Results

In this chapter, we first developed an exact algorithm (B&B) for the problem with three different lower bounding methods. We next proposed a beam search

(BS) algorithm along with an improvement search heuristic (ISH). We further extended BS to a recovering beam search (RBS) algorithm. We coded these algorithms in C language and compiled with Gnu C compiler version 2.95.3. All codes were run on the operating system Solaris 2.7 on a workstation Sun HPC 4500 with 12×400 MHz UltraSPARC CPU and 3GB memory. The B&B, BS and RBS algorithms used the CPLEX 9.1 commercial solver to compute the lower bounds LB_{LP} and LB_{IP}. All reported computational times are in seconds.

We considered two experimental factors: number of jobs (N= 10, 15, 20) and number of machines (M= 2, 3, 4). For each experimental setting we took 5 replications. For each replication we generated cutting specifications (diameter, length, depth of cut and required surface roughness) of jobs randomly. For each job we randomly used one of the tool types out of ten types of cutting tools with different technical coefficients given in Chapter 3. We randomly generated the cost of each tool C_t from the uniform distribution U[5,10]. C_t determines the cost coefficient T_j with other job and tool specific parameters as discussed in Chapter 3. We used four types of machines with the following C_m, H_m couples: (0.3, 5), (0.5, 10), (0.7, 15) and (0.9, 20). The CNC machines with higher horsepower, H_m, capabilities can attain higher cutting speeds and feed rates (i.e. lower processing times), but their initial investment cost (and their operating cost) would be higher as well. This way we can evaluate the impact of different CNC machine technologies on the scheduling decisions. In our computational runs, when $M = 2$, we used first two machines described above and when $M = 3$, we used first three machines.

Another very important factor for the problem is the limit (K) on makespan objective of the schedule. How to select a K value for a given problem setting is a critical decision since selecting a very small K value may cause all instances of a replication to be infeasible. In order to see the effect of K, we solved each replication of the problem for 5 different levels of K. To find proper K values, we first solved the makespan minimization problem for each replication for fixed processing times case where $p_{jm} = p^l_{jm}$ for each j and m. This is a makespan minimization problem on unrelated machines and known to be $\mathcal{NP}$-hard, so we used a polynomial-time algorithm by Davis and Jaffe [27], which was shown to have a

worst case bound of $(1+\sqrt{2})\sqrt{M}$. This algorithm provides us a feasible makespan level K which we denote as K_{DJ}. We calculated five different K levels by using the formula $K = k \times K_{DJ}$ where $k = 0.6, 0.8, 1, 1.2, 1.4$. In the computational study, we first solved randomly generated problems by using three different B&B algorithms. Each B&B algorithm uses a different lower bounding method that we proposed in Section 5.3. The B&B algorithm either finds out that a problem is infeasible or gives us an optimal solution. For the cases that B&B found an optimal solution we solved the problem by the BS and RBS algorithms which use LB_{LP} as a partial schedule evaluation tool. We next tested ISH algorithm using 3 different starting solutions provided by BS, RBS and IS algorithms.

A critical step in our B&B algorithm is deciding the job order (Step 2), i.e. determining which job to be added to a partial schedule next at a given level of the search tree. We ordered the jobs in a descending order of $\max_m p^l_{jm}$. This is intuitive because as we schedule the jobs with higher processing time lower bound at earlier stages of the B&B tree, we catch infeasible schedules earlier and this reduces the number of nodes to be opened and decreases the computation time. In computational study, we considered two other rules to order jobs in Step 2 of B&B and took trial runs to compare different methods. One method is to order the jobs in ascending order of $\min_m f_{jm}(p^u_{jm})$ as in the IS algorithm, which allows us to schedule lower cost jobs earlier in the B&B tree. We also considered $\max_m f_{jm}(p^u_{jm})$, which allows us to schedule highest minimum cost jobs earlier. We took trial runs for $N = 10$ and $M = 2$ and 3. We give the average results for CPU, number of opened nodes, number of eliminated nodes due to feasibility and number of eliminated nodes by lower bound (optimality) in Table 5.2. The results show that our selection of $\max_m p^l_{jm}$ order performs better than the other ordering rules both for the CPU requirement and for the node elimination capability due to feasibility and optimality.

We next discuss the performance of the B&B algorithm with different lower bounding methods. We consider the cases where a feasible solution is available for the problem. We give the size of the eliminated B&B tree and traversed nodes for different lower bounding methods in Table 5.3. For a given (N, M) instance, the maximal number of nodes to be traversed in worst case in our B&B tree can be

Job order	CPU	Opened Nodes	Eliminated due to Feasibility	Eliminated by Lower Bound
$\max_m p^l_{jm}$	0.20	1426	29199	10923
$\min_m f_{jm}(p^u_{jm})$	1.85	16056	21941	3550
$\max_m f_{jm}(p^u_{jm})$	1.45	12171	25333	4043

Table 5.2: Trial Results for Job Ordering Rules for Step 2 of B&B.

calculated by $(1-M^{N+1})/(1-M)$. For $M = 4$ and $N = 20$, total number of nodes to be traversed may reach 1,466,015,503,701. Similarly, when we decide to fathom a node at level L, we save from opening $(1 - M^{N-L+1})/(1 - M) - 1$ nodes which is the number of nodes that would grow from the fathomed node at the worst case. We measured the number of eliminated nodes due to bounds and feasibility in terms of their percentages to the maximal total number of nodes. The results in Table 5.3 show that our lower bounds can reduce the tree size by 29% on the average. There are instances where this reduction reaches to 87%. The feasibility effect reduced the tree size by 66% on the average. The last column of the table for traversed tree size shows that we could solve the problems by just opening 4.6% of the nodes on the average. There are cases solved by just traversing a negligible size of B&B tree. We have shown in Section 5.3 in Lemma 5.7 that for a given partial schedule (node) the following relationship holds: $LB_{IP} \geq LB_{LP} \geq LB_R$. We also observe this relationship between the sizes of the eliminated B&B trees by different lower bounding methods in Table 5.3.

	Eliminated by Lower Bound			Eliminated due to Feasibility			Traversed		
LB type	Mean	Min	Max	Mean	Min	Max	Mean	Min	Max
LB_R	26%	0 %	86.4%	68.4%	0.4%	100%	5.6%	0%	43%
LB_{LP}	28.8%	0%	87.2%	66.5%	0.4%	100%	4.7%	0%	38.4%
LB_{IP}	29.1%	0%	87.2%	66.3%	0.4%	100%	4.6%	0%	38.2%

Table 5.3: Eliminated and Traversed Tree Sizes

In Table 5.4, we present the CPU requirements of different lower bounding methods in B&B algorithm for different experimental settings. This table shows that increasing N or M strongly affects the running time of the B&B algorithm

as expected. For the considered N and M levels, LB_R has the minimum average running time. The second best alternative is the LB_{LP} in terms of the CPU time. An important observation on Table 5.4 is that as N and M increase the CPU time required by LB_R approaches to the CPU time required by LB_{LP}, so we may expect to see that LB_{LP} will have shorter CPU times for larger problem sizes. If we check the CPU time ratio LB_R/LB_{IP}, we observe that as N is increased, performance of the LB_R gets closer to the performance of LB_{IP}, but as M is increased, we observe the opposite. This is due to the fact that computing LB_{IP} is itself an $\mathcal{NP}$-hard problem and requires much more time when M is increased. Another observation on Table 5.4 is that for each lower bounding method we see big gaps between minimum and maximum CPU times. This is because we solve each problem for different K levels as discussed below.

		$M=2$			$M=3$			$M=4$		
N	LB type	Mean	Min	Max	Mean	Min	Max	Mean	Min	Max
10	LB_R	0.08	0.02	0.16	0.26	0.01	0.59	1.18	0.08	2.30
	LB_{LP}	0.26	0.10	0.46	1.11	0.05	2.33	5.03	0.44	10.51
	LB_{IP}	0.32	0.14	0.52	1.42	0.05	2.89	6.11	0.25	10.6
15	LB_R	2.06	0.20	5.81	21.3	0.26	80.6	241	0.99	1002
	LB_{LP}	4.88	0.69	12.7	65.4	1.21	268	662	3.67	3422
	LB_{IP}	5.99	1.74	13.5	90.7	0.28	314	927	2.90	4196
20	LB_R	56.2	3.84	186	2443	33.2	7380	69950	236	177030
	LB_{LP}	112	7.93	382	4853	114	18266	101293	618	362148
	LB_{IP}	129	22.2	386	7550	25.9	24468	169676	319	584398

Table 5.4: CPU Requirements (in seconds) for different lower bounding methods

Table 5.5 gives the average size of the eliminated and traversed nodes and required CPU time for $N = 20$ and $M = 4$ for different K levels, such that $K = k \times K_{DJ}$ where $k = 0.6, 0.8, 1, 1.2, 1.4$. For example, $k = 0.6$ corresponds to the case where K level is 1 and so on. We observe that as K is increased, the size of the traversed tree increases since fewer number of nodes are eliminated due

to the feasibility. Hence, the CPU time required to solve the problem increases, too. We see that the CPU time requirement when K level is 5 is twenty times higher than the CPU time requirement when K level is 3. This shows that CPU requirement of the B&B algorithm is strongly affected by K. Therefore, we can say that the B&B algorithm is more efficient for smaller K's in terms of running time.

K level	El. by LB	El. by Feas.	Traversed	CPU
3	1.4%	98.6%	0.0%	10402
4	7.5%	92.5%	0.004%	93921
5	18.7%	81.2%	0.014%	199556

Table 5.5: Eliminated and Traversed Nodes at different K levels for $N = 20$ and $M = 4$ by LB_{LP}

In Table 5.6, we give the solution quality results for IS, BS and RBS algorithms. We use a beam width $b = 3$ for BS and RBS. We define the relative solution quality of an algorithm A, R_A, as the ratio of the difference between cost achieved by A and the optimal cost achieved by B&B over the optimal cost expressed in %. It is the percentage deviation from the optimum. The average performance of IS algorithm varies between 7.2% and 22.9%. BS algorithm achieves an average performance between 1.8% and 7.9%. RBS algorithm performs best and gives an average performance between 0.1% and 1.4%. There are cases where BS and RBS achieves the optimum. The worst performance for RBS is 9.6% whereas it is 26.5% for BS. We observe that including a recovering step in BS algorithm significantly improved the solution quality. When we check the CPU time performance for each heuristic, we see that both three methods are very efficient. We give CPU time requirements for BS and RBS algorithms in Table 5.8. Even the worst case average performance is 0.71 CPU seconds. We also observe that IS has negligible CPU time requirement.

In Table 5.7, we give the solution quality results for ISH algorithm for three different starting solutions provided by IS, BS and RBS. We represent the deviation of ISH from the optimum as R_{A+ISH} where A stands for the algorithm output

		$R_{IS}(\%)$			$R_{BS}(\%)$			$R_{RBS}(\%)$		
N	M	Mean	Min	Max	Mean	Min	Max	Mean	Min	Max
	2	11	0.6	43.1	1.8	0	11.5	0.1	0	0.7
10	3	22.6	2.8	45.7	4.8	0	24.6	0.6	0	3.9
	4	22.9	6.1	40.6	2.1	0.1	11.9	1.4	0	9.6
	2	7.4	0.4	24.5	3.7	0	21.5	0.4	0	2.6
15	3	18.4	8.7	27.8	5.4	0.1	16.9	0.5	0	2.6
	4	15.4	10.0	22.3	5.0	0.4	23.5	0.8	0.1	3
	2	7.2	0.1	21.5	5.0	0	19.9	0.4	0	1.6
20	3	14.3	4.6	28.7	7.9	0.5	26.5	0.9	0	4.3
	4	17.2	8.5	28.9	5.3	0.6	17.4	1.1	0	5.6

Table 5.6: Deviations from the optimum for IS, BS and RBS heuristics

of which is used as starting solution by ISH algorithm. In comparison with the starting solutions, we observe that ISH achieves significant improvement. ISH achieves its maximum improvement when it started with the solution given by IS algorithm, which performs worse in comparison with BS and RBS. Even for RBS, which gives best results, ISH achieves 48.5% average improvement from R_{RBS} to $R_{RBS+ISH}$.

We next analyzed the performances of BS and ISH algorithms for different K levels as reported in Table 5.9. A very important observation is that solution quality of BS and ISH improve as K is increased. Hence, for the problem instances where our B&B algorithm is not computationally efficient, our BS and ISH algorithms can achieve solutions more closer to the optimum. This is due to the shape of the manufacturing cost function. When K is increased, we deal with higher processing time values where the manufacturing cost functions are flatter.

Finally, we tested IS, RBS and ISH algorithms for 50-100 jobs and 2,3,4 machines. We cannot solve these instances to optimum due to CPU time requirements. We compared the results achieved by RBS algorithm with the results of

		$R_{IS+ISH}(\%)$			$R_{BS+ISH}(\%)$			$R_{RBS+ISH}(\%)$		
N	M	Mean	Min	Max	Mean	Min	Max	Mean	Min	Max
	2	0.5	0	3.9	1.1	0	11.5	0.06	0	0.7
10	3	2.4	0	6.5	4.3	0	23.1	0.5	0	3.8
	4	1.9	0	7.1	1.7	0	11.9	0.1	0	9.3
	2	0.3	0	4.5	1.8	0	12.4	0.1	0	2.6
15	3	1.0	0	3.1	3.2	0	11.8	0.4	0	2.6
	4	1.5	0	3.2	4.4	0	23.2	0.5	0	2.8
	2	0.4	0	3.4	2.5	0	19.9	0.1	0	0.4
20	3	1.5	0	4.6	4.9	0.2	20.9	0.7	0	4.3
	4	1.6	0	3.2	4.4	0.4	16.5	0.9	0	5.4

Table 5.7: Deviations from the optimum for ISH algorithm

N	M	CPU_{BS}	CPU_{RBS}	CPU_{BS+ISH}	$CPU_{RBS+ISH}$	CPU_{IS+ISH}
	2	0.04	0.08	0.0	0.0	0.01
10	3	0.06	0.12	0.01	0.0	0.00
	4	0.08	0.15	0.01	0.01	0.01
	2	0.07	0.20	0.01	0.02	0.03
15	3	0.12	0.29	0.02	0.01	0.02
	4	0.16	0.35	0.02	0.01	0.03
	2	0.12	0.41	0.05	0.06	0.05
20	3	0.19	0.58	0.08	0.03	0.05
	4	0.28	0.71	0.06	0.04	0.05

Table 5.8: Average CPU time (sec.) requirements

IS algorithm. I_{RBS} is the percentage deviation of RBS from the solution achieved by IS. We observe that RBS together with ISH can achieve solutions 10% better on the average from IS. For large instances, our heuristics find a solution in reasonable CPU times.

	R_{BS}				R_{ISH}			
K level	Mean	Min	Max	Std. Dev.	Mean	Min	Max	Std. Dev.
2	9%	0%	21.5%	0.07	5.6%	0%	19.9%	0.06
3	6.2%	0%	26.5%	0.07	4.4%	0%	23.2%	0.06
4	3%	0%	18.2%	0.05	2.3%	0%	18.2%	0.04
5	2.1%	0%	12.4%	0.03	1.2%	0%	8.7%	0.02

Table 5.9: Performances of Beam Search and Improvement Search Heuristics at different K levels

		$I_{RBS}(\%)$			$I_{RBS+ISH}(\%)$			CPU_{RBS}			$CPU_{RBS+ISH}$		
N	M	Mean	Min	Max	Mean	Min	Max	Mean	Min	Max	Mean	Min	Max
	2	4.6	0.1	14.9	5	0.2	15.1	4.16	3.40	4.56	2.63	0.01	10.56
50	3	11.5	5.3	25.7	11.7	5.4	26.4	5.73	5.45	6.21	1.13	0.12	4.34
	4	9.0	7.0	11.7	9.4	7.4	12.2	7.39	6.84	8.48	2.65	1.19	4.13
	2	4.2	-0.1	13.5	4.6	0.2	13.6	27.79	23.25	29.80	50.85	0.13	194.21
100	3	9.8	4.7	16.1	10.1	4.9	17.0	42.30	39.60	44.91	22.49	1.10	64.95
	4	9.6	6.5	17.3	10.0	6.6	18.3	56.69	52.22	62.76	27.92	13.67	57.12

Table 5.10: Performances of IS and ISH

5.10 Conclusion

In this chapter, we considered the problem of minimizing total manufacturing cost in non-identical parallel CNC machine environment with an upper bound on the makespan of the schedule. We provided an exact algorithm (B&B) for the problem along with three alternative lower bounding methods. To the best of our knowledge, our algorithm is the first exact algorithm for this problem. We further proposed a recovering beam search algorithm which employs our lower bounding methods as an evaluation function for partial schedules. Finally, we gave an improvement search algorithm for the problem. For this algorithm, we showed two properties which provide improving search moves for a given schedule.

Our computational results show that the proposed exact algorithm can solve the problems by just traversing the 5% of the maximal possible B&B tree size and the proposed lower bounding methods can eliminate up to 80% of the search tree. For the cases where B&B is not computationally efficient, our beam search and improvement search algorithms achieved solutions within 1% of the optimum on the average in a very short computation time. The results of this chapter recently appeared as an article (Gürel and Aktürk [39]).

In the next chapter, we will focus on finding strengthened conic quadratic formulations for the machine-job assignment problem with controllable processing times. Differently, we will be considering machine capacity as constraint on each machine and we will be formulating manufacturing cost of a job in terms of the amount of compression on its processing time.

Chapter 6

A Strong Conic Quadratic Reformulation for Machine-Job Assignment Problems

In this chapter, we will take a different approach and study the polyhedral structure of a machine-job assignment problem with controllable processing times in order to achieve a stronger formulation for the problem. We will use the conic quadratic (second order cone) inequalities in reformulating the problem.

The machine-job assignment problem considered here is to maximize the profit for a given set of jobs on a set of machines with capacity constraints. We deal with a non-identical parallel machine environment where the decision to be made is the assignment of jobs to the machines and the compression levels on the processing times of the jobs. In this part, we assume that regular profit values for each job on each machine is given. Each job has a regular processing time level which can be compressed by incurring compression cost determined by a convex cost function. Modeling the cost change of a job as a function of compression on its processing time is a widely used approach in the scheduling with controllable processing times literature. Since the machines are non-identical, compression

cost functions are different for each job on different machines. If compression of processing times is not allowed, the machine-job assignment problem reduces to the classical generalized assignment problem, which is known to be $\mathcal{NP}$-hard.

The nonlinearity of the compression cost makes this assignment problem particularly difficult to solve in practice. Even for the quadratic case, commercially available software packages that employ fast quadratic programming (QP) algorithms within a branch-and-bound framework are far from solving large instances of the problem. Our approach is analogous to the polyhedral approach for linear integer programming with the goal of strengthening bounds from continuous relaxations of the problem.

This chapter is organized as follows. In Section 6.1 we give the definition of the machine-job assignment problem with controllable processing times and a nonlinear mixed 0-1 programming formulation for it. In Section 6.2 we describe conic reformulation of the problem so as to strengthen the bounds from the continuous relaxations. In Section 6.3 we present a computational analysis of the introduced formulations. Finally, we conclude with Section 6.4.

6.1 Problem Definition

Given n jobs and m non-identical parallel machines with finite capacity, the machine-job assignment problem is to choose a subset of the jobs and assign them to the machines so that the total profit from the assignment is maximized. Letting c_i denote the available machining time for machine $i = 1, \ldots, m$, and p_{ij} and h_{ij}, the regular processing time and profit corresponding to job j if it is assigned to machine i, the problem can be modeled as a linear 0-1 program. This problem is also referred to as the generalized assignment problem (Savelsbergh [75]).

In a flexible manufacturing system, where jobs are processed on computer numerically controlled (CNC) machines, processing times can be reduced by appropriately setting the machining parameters such as cutting speed and feed rate.

However, compressing processing time naturally leads to reduced tool life, and, consequently, increased machining cost. We model the change in the machining cost due to processing time compression $y \geq 0$ as

$$f(y) = ky^{a/b},$$

where a and b are integers satisfying $a \geq b > 0$ and $k > 0$, so that f is an increasing and convex function of the compression. The function f reflects the relationship between compression and cost in that as one decreases the processing time of a job, it becomes more expensive to compress it further. Technical specifications of a job such as its length, diameter, required surface quality, as well as machine and tool type determine the cost coefficients k, a, and b. Defining a binary assignment variable

$$x_{ij} = \begin{cases} 1, & \text{if job } j \text{ is assigned to machine } i, \\ 0, & \text{otherwise,} \end{cases}$$

and compression variable y_{ij} for each machine-job pair, the *machine-job assignment problem with controllable times* can be formulated as the following nonlinear mixed 0-1 program:

$$\begin{aligned}
\max \quad & \sum_{i=1}^{m}\sum_{j=1}^{n}(h_{ij}x_{ij} - f_{ij}(y_{ij})) \\
\text{s.t.} \quad & \sum_{j=1}^{n}(p_{ij}x_{ij} - y_{ij}) \leq c_i, && i = 1,\ldots,m, && (6.1) \\
\text{(MJ0)} \qquad & y_{ij} \leq x_{ij}u_{ij}, && i = 1,\ldots,m,\ j = 1,\ldots,n, && (6.2) \\
& \sum_{i=1}^{m} x_{ij} \leq 1, && j = 1,\ldots,n, && (6.3) \\
& x_{ij} \in \{0,1\},\ y_{ij} \in \mathbb{R}_+, && i = 1,\ldots,m,\ j = 1,\ldots,n. && (6.4)
\end{aligned}$$

Constraint (6.1) ensures that the jobs assigned to machine i take no more than the machine capacity c_i. Constraint (6.2) ensures that compression is allowed on the processing time of job j on machine i only if job j is assigned to machine i and that compression is no more than a specified maximum $u_{ij} < p_{ij}$. Finally, constraint (6.3) guarantees that each job is assigned to at most one machine.

MJ0 is $\mathcal{NP}$-hard as it reduces to the generalized assignment problem when all u_{ij} are zero. The nonlinearity introduced with the option of compression of processing times makes the problem much harder to solve, in practice, compared to the generalized assignment problem. Note that MJ0 is a maximization problem with a concave objective and the feasible set of its continuous relaxation

$$P = \left\{ (x, y) \in \mathbb{R}^{2mn}_{+} : \ (6.1), (6.2), (6.3) \right\}$$

is a polyhedron. Unlike for the case of generalized assignment problem, optimal solutions to its continuous relaxation are found typically in the interior of this polyhedron with almost all x_{ij} being fractional. Consequently, branch-and-bound algorithms based on such relaxations require an excessive branching to find feasible integer solutions. Even when f is quadratic, i.e., $a/b = 2$, it is a challenge to solve practical-size instances of MJ0 with quadratic MIP solvers of commercial software packages. We will elaborate on the computational difficulty of solving MJ0 in Section 6.3.

Rather than developing a special purpose algorithm for MJ0, our goal is to reformulate the problem so that its continuous relaxation is stronger and the formulation may be solved by readily available solvers of optimization software packages. In particular, we describe a conic quadratic relaxation whose optimal solutions avoid all non-extreme points of P. Our results on conic strengthening are general enough for them to be applicable to other mixed 0-1 minimization problems with convex separable objective with rational exponents.

6.2 Conic Reformulations

In this section we describe strong conic reformulations of MJ0. We first point out the source of difficulty in MJ0 due to the nonlinearity of the objective. Then we describe a strengthening and show how to express it using a polynomial number of conic quadratic constraints.

6.2.1 Working with epi(f)

For strengthening the formulation it is convenient to work with the epigraph of f. So, by introducing auxiliary variables $t_{ij} \in \mathbb{R}_+$ we bring the nonlinear objective into the constraints and linearize the objective of the formulation as

$$\begin{aligned} & \max \sum_{i=1}^{m}\sum_{j=1}^{n}(h_{ij}x_{ij} - k_{ij}t_{ij}) \\ \text{(MJ1)} \quad & \text{s.t.} \qquad y_{ij}^{a_{ij}/b_{ij}} \leq t_{ij}, \qquad i = 1,\dots,m,\ j = 1,\dots,n, \qquad (6.5) \\ & \qquad (6.1), (6.2), (6.3), (6.4). \end{aligned}$$

MJ1 is not necessarily easier to solve than MJ0. On the contrary, solvers can usually deal with nonlinearity in the objective easier than nonlinearity in the constraints. MJ1 is an intermediate formulation that will enable us to derive a strong conic formulation. Note that because MJ1 has a linear objective, its continuous relaxation has optimal solutions that are extreme points of its feasible region.

For our purpose it is sufficient to concentrate on the mixed 0-1 set

$$C = \left\{ x \in \{0,1\}, y, t \in \mathbb{R}_+ \ : \ y^{a/b} \leq t,\ y \leq ux \right\}.$$

Observe that constraints of C are of the form (6.1), (6.4), and (6.5). The results in this chapter are applicable to any optimization problem that contains C as a substructure. It is useful to consider the continuous relaxation of C

$$C_R = \left\{ x, y, t \in \mathbb{R}_+ \ : y^{a/b} \leq t,\ y \leq ux,\ x \leq 1 \right\}$$

to understand the source of difficulty with nonlinearity of the objective. The proposition below shows that C_R has infinitely many extreme points with fractional x.

Proposition 6.1. For $a > b$, each point on the curve defined as

$$L = \left\{ x, y, t \in \mathbb{R} : 0 < x < 1,\ y = ux,\ y^{a/b} = t \right\}$$

is an extreme point of C_R.

Proof. We will prove the claim by showing that each point on L is the unique face of C_R defined by some supporting hyperplane. To see this, consider the optimization problem with a parametric linear objective

$$\begin{aligned} \max\ & \lambda y - x - t \\ \text{s.t.}\quad & y^{a/b} - t \le 0, \ \ (\alpha) \\ & y - ux \le 0, \ \ (\beta) \\ & x \le 1, \ \ (\gamma) \\ & -x \le 0. \ \ (\delta) \end{aligned}$$

From complementary slackness conditions we have $\gamma = \delta = 0$ for $0 < x < 1$. Then writing the KKT dual feasibility conditions

$$\begin{aligned} \lambda &= \alpha \frac{a}{b} y^{\frac{a-b}{b}} + \beta, \ \ (y) \\ -1 &= -u\beta, \qquad\qquad (x) \\ -1 &= -\alpha, \qquad\qquad (t) \end{aligned}$$

we find that the remaining dual variables are $\alpha = 1$ and $\beta = 1/u$. Therefore, complementary slackness conditions imply that $y = ux,\ y^{a/b} = t$ for any KKT point with $0 < x < 1$. Also, since there is a unique solution to

$$\lambda - \frac{1}{u} = \frac{a}{b} y^{\frac{a-b}{b}}$$

between $0 \le y \le u$ for $1/u \le \lambda \le \frac{a}{b} u^{\frac{a-b}{b}}$, each point on L is a unique KKT point, hence optimal solution to the problem for such λ. □

The set of points L is illustrated with the dashed curve in Figure 6.1 (a). Our next goal is to reformulate C so that L is eliminated from its continuous relaxation.

6.2.2 Strengthening the continuous relaxation

First, observe that for C, as $y, t \geq 0$ and $b > 0$, inequality $y^{a/b} \leq t$ is equivalent to

$$y^a \leq t^b. \tag{6.6}$$

We propose to strengthen (6.6) as

$$y^a \leq t^b x^{a-b}. \tag{6.7}$$

Because $a \geq b$, for $0 \leq x \leq 1$ inequality (6.7) implies (6.6). It is also clear that (6.7) is valid for C as for $x \in \{0, 1\}$ it reduces to (6.6). Thus, we may replace (6.6) with (6.7).

Consider, then, the strengthened continuous relaxation of C:

$$C_S = \left\{ \; x, y, t \in \mathbb{R}_+ \; : y^a \leq t^b x^{a-b}, \; y \leq ux, \; x \leq 1 \; \right\}.$$

Although (6.7) is highly nonlinear, C_S is a convex set. Indeed, as we show in the next proposition, it is the smallest convex relaxation of C.

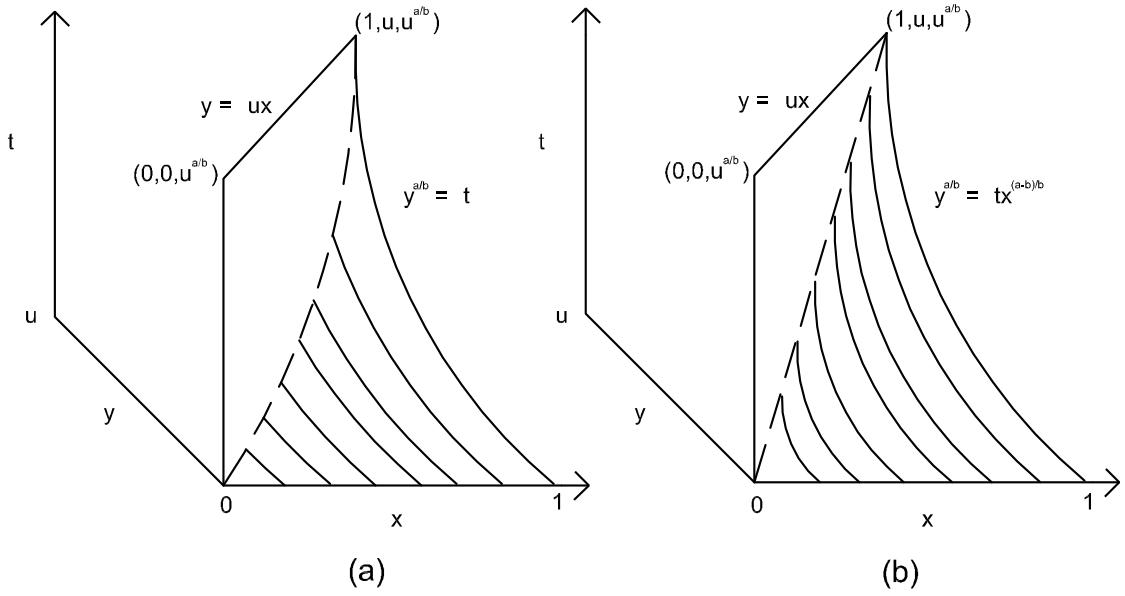

Figure 6.1: Surfaces defined by inequalities (6.6) and (6.7).

Proposition 6.2. The convex hull of C, $\text{conv}(C)$, equals C_S.

Proof. Consider the disjunction $C_0 \cup C_1$, where $C_0 := \{(x, y, t) \in C : x = 0\}$ and $C_1 := \{(x, y, t) \in C : x = 1\}$; thus, $C = C_0 \cup C_1$. We will first show that $\text{conv}(C) \subseteq C_S$. Consider an arbitrary point $p_0 = (0, 0, t_0) \in C_0$ and an arbitrary point $p_1 = (1, y_1, t_1) \in C_1$. Let p be a convex combination of p_0 and p_1; that is,

$$p = (x, y, t) = (1 - \lambda)(0, 0, t_0) + \lambda(1, y_1, t_1) = (\lambda, \lambda y_1, (1 - \lambda)t_0 + \lambda t_1)$$

for some $0 \leq \lambda \leq 1$. Clearly, $p \in \mathbb{R}^3_+$, $y = \lambda y_1 \leq ux = u\lambda$, and $x = \lambda \leq 1$. To see that (6.7) holds for p, observe that

$$(\lambda y_1)^a = (\lambda^b y_1^a)(\lambda^{a-b}) = [(1 - \lambda)0 + \lambda y_1^{a/b}]^b \lambda^{a-b} \leq [(1 - \lambda)t_0 + \lambda t_1]^b \lambda^{a-b},$$

where the last inequality holds from $0 \leq t_0$ and $y_1^{a/b} \leq t_1$. Thus, $\text{conv}(C) \subseteq C_S$.

For $C_S \subseteq \text{conv}(C)$, consider an arbitrary point $p = (x, y, t) \in C_S$. If $x = 0$ or $x = 1$, then $p \in C \subseteq \text{conv}(C)$ trivially. On the other hand, if $0 < x < 1$, then p is a convex combination of $(0, 0, 0) \in C_0$ and $(1, y/\lambda, t/\lambda)$ with $\lambda = x$. To see that the latter point is in C_1, observe that $y/\lambda \leq u$ and $(y/\lambda)^a \leq (t/\lambda)^b$, or equivalently, $y^a \leq t^b \lambda^{a-b}$ as $p \in C_S$. □

Inequality (6.7) is illustrated in Figure 6.1 (b). This figure shows that (6.7) defines the curved boundary of $\text{conv}(C)$ and that any point (x, y, t) on it with $0 \leq x \leq 1$ is a convex combination of C_0 and C_1.

The proof of Proposition 6.2 should convince the reader that inequality (6.7) indeed defines the epigraph of the perspective of f. Recently, Frangioni and Gentile [32] proposed an interesting cut generation method based on perspective functions. Their approach is to generate supporting hyperplanes of the perspective of a convex function as cuts to improve relaxations with convex objective on a polyhedral set. Although this is a more general approach as it applies to any separable convex function, as also stated by Frangioni and Gentile in their computational study, there are practical difficulties with approximating the perspective with a large number of linear inequalities and solving a relaxed problem as this leads to finding infeasible integer solutions that need to be avoided.

Here we use the nonlinear constraint (6.7) explicitly by reformulating it via conic quadratic constraints as discussed in Section 6.2.3. Before doing so, we address the important question: what happens when constraint (6.7) is used in conjunction with other constraints of the problem at hand? Specifically, consider the reformulation

$$\begin{aligned} & \max && \sum_{i=1}^{m}\sum_{j=1}^{n}(h_{ij}x_{ij} - k_{ij}t_{ij}) \\ \text{(MJ2)} \quad & \text{s.t.} && y_{ij}^{a_{ij}} \le t_{ij}^{b_{ij}} x_{ij}^{a_{ij}-b_{ij}}, \qquad i = 1,\dots,m,\ j = 1,\dots,n, \qquad (6.8) \\ & && (6.1), (6.2), (6.3), (6.4). \end{aligned}$$

It follows from the next proposition that any extreme point of the continuous relaxation of MJ2 projects to an extreme point of the continuous relaxation of MJ0. Thus, because MJ2 has a linear objective, by solving its relaxation, one avoids all non-extreme points of the continuous relaxation of MJ0. Therefore, a major source of difficulty due to the nonlinearity of the objective of MJ0 is eliminated.

Proposition 6.3. Let $P \subseteq \mathbb{R}^{2n}$ be a closed, convex set. For $i = 1, \dots, n$, let

$$C_i = \left\{ x, y, t \in \mathbb{R}_+ : y^{a_i} \le t^{b_i} x^{a_i - b_i}, y \le u_i x,\ x \le 1 \right\}$$

and

$$Q = \left\{ (x, y, t) \in \mathbb{R}_+^{3n} : (x, y) \in P,\ (x_i, y_i, t_i) \in C_i,\ i = 1, \dots, n \right\}.$$

If (x, y, t) is an extreme point of Q, then (x, y) is an extreme point of P.

Proof. Note that for any extreme point (x, y, t) of Q, t_i equals the smallest value defined by (x_i, y_i), i.e.,

$$t_i = \tau(x_i, y_i) := \begin{cases} x_i(y_i/x_i)^{a_i/b_i}, & \text{if job } x_i > 0, \\ 0, & \text{if } x_i = 0, \end{cases}$$

for all i. For contradiction, suppose $(x, y) = \lambda(x', y') + (1 - \lambda)(x'', y'')$ for some (x', y'), (x'', y'') in P and $0 < \lambda < 1$. Now consider the two points (x', y', t') and (x'', y'', t'') in Q with componentwise smallest t', t''. Then, for each i

$$\begin{aligned}\lambda t'_i + (1 - \lambda)t''_i &= \lambda\tau(x'_i, y'_i) + (1 - \lambda)\tau(x''_i, y'') \\ &= \tau(\lambda(x'_i, y'_i)) + \tau((1 - \lambda)(x''_i, y'')) \leq \tau(x_i, y_i) = t_i,\end{aligned}$$

where the second inequality follows from positive homogeneity and the inequality follows from superadditivity of a convex function $g : \mathbb{R}^n_+ \to \mathbb{R}$ with $g(0) = 0$. On the other hand, from convexity of τ, we have

$$\lambda\tau(x'_i, y'_i) + (1 - \lambda)\tau(x''_i, y''_i) \geq \tau(x_i, y_i)$$

as well, implying $\lambda t'_i + (1 - \lambda)t''_i = t_i$. Contradiction with the choice of (x, y, t) as an extreme point of Q. □

6.2.3 Conic quadratic representation

In this section we will give an efficient representation of the set C_S using a polynomial number of conic quadratic constraints. It is known that an inequality of the form

$$r^{2^l} \leq s_1 s_2 \cdots s_{2^l}, \tag{6.9}$$

for $r, s_1, \ldots, s_{2^l} \geq 0$ can be expressed equivalently using $O(2^l)$ variables and $O(2^l)$ hyperbolic inequalities of the form

$$u^2 \leq v_1 v_2, \;\; u, v_1, v_2 \geq 0 \tag{6.10}$$

[13]. Furthermore, each constraint (6.10) can be written as a conic quadratic (second-order cone) constraint

$$\|(2u, v_1 - v_2)\| \leq v_1 + v_2. \tag{6.11}$$

Proposition 6.4. Inequalities

$$y^a \le t^b x^{a-b},\ x, y, t \ge 0$$

can be expressed equivalently using $O(\log_2 a)$ variables and $O(\log_2 a)$ conic quadratic constraints of the form (6.11).

Proof. Let $l = \lceil \log_2(a) \rceil$ and, using $y \ge 0$, rewrite constraint (6.7) as

$$y^{2^l} \le t^b x^{a-b} y^{2^l - a}. \tag{6.12}$$

Now it is clear that (6.12) is a special case of (6.9) with $s_1 = \cdots = s_b = t,\ s_{b+1} = \cdots = s_a = x,\ s_{a+1} = \cdots = s_{2^l} = y$. Following the construction in Alizadeh and Goldfarb [7], inequalities (6.10) can be built using a binary tree with leaf nodes for $1, t, t^2, t^4, \ldots, t^{2^{\lfloor \log_2(b) \rfloor}}, x, x^2, x^4, \ldots, x^{2^{\lfloor \log_2(a-b) \rfloor}}$, and $y, y^2, y^4, \ldots, y^{2^{l-1}}$. Each non-leaf node of the binary tree represents a new hyperbolic inequality (6.10) and variable introduced. Because the binary tree has at most two times the number of its leaves, the number of inequalities and variables in the conic quadratic representation is at most $O(\log_2 a)$. □

Below we illustrate how to represent inequalities (6.7) with conic quadratic constraints with an example.

Example 1. Suppose the convex function is given as $f(y) = y^{7/5}$. We write the corresponding inequalities

$$y^7 \le t^5 x^2,\ y, t, x \ge 0$$

first as

$$y^8 \le t^5 x^2 y,\ y, t, x \ge 0.$$

Then using the binary representation of the exponents of t^5, x^2, and y on the right hand side to form the leaves of the construction tree shown in Figure 6.2, we express the inequalities equivalently with the following hyperbolic constraints:

$$v_1^2 \le yt, \quad y, t \ge 0,$$

$$v_2^2 \leq xv_1,\ x, v_1 \geq 0,$$
$$y^2 \leq tv_2,\ \ t, v_2 \geq 0.$$

Note that only those powers of 2 needed to express an integer exponent are used as the leaves of the binary tree. The hyperbolic constraints are then written in conic quadratic form as

$$\|(2v_1,\ y - t)\| \leq y + t,$$
$$\|(2v_2, x - v_1)\| \leq x + v_1,$$
$$\|(2y, t - v_2)\| \leq t + v_2.$$

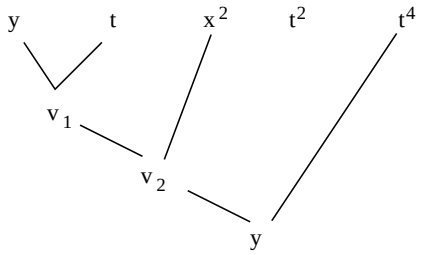

Figure 6.2: Binary construction tree for Example 1.

Observe that conic reformulations based on (6.6) can be obtained by simply fixing $x = 1$ in this derivation. We refer to the conic reformulation of MJ1 based on the weak constraint (6.6) as CMJ1 and to the conic reformulation MJ2 based on the strengthened constraint (6.7) as CMJ2. In the next section, we will compare these alternative conic reformulations computationally.

6.3 Computational Analysis

In order to test the computational effectiveness of our approach we have performed experiments with different formulations of the problem using three different objective functions. The first two are the quadratic and cubic cases, i.e., $f(y) = ky^2$ and $f(y) = ky^3$. The third one is $f(y) = ky^{a/b}$, where the rational exponent a/b is generated from Uniform [1.0,3.0] with a single decimal digit. All

experiments are performed using ILOG CPLEX Version 10.1 on a 3 GHz Linux workstation with 512 MB memory with a 1000 CPU seconds time limit.

We performed experiments on data sets with varying number of jobs ($n = 50, 100, 150, 200$), machines ($m = 1, 2, 3$), and capacity factors ($\kappa = 0.1, 0.2$). For each experimental configuration of n, m, κ, we generated five instances with h_{ij} from Uniform[2.0,6.0], k_{ij} from Uniform [1.0,3.0], p_{ij} from Uniform[1.0,3.0], and u_{ij} from $p_{ij}\times$ Uniform [0.2,0.8]. The capacity factor κ is used to set the machining time capacities c_i as

$$c_i = \kappa \times \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij}}{m} .$$

We compare three formulations for the quadratic case $f(y) = ky^2$. The first formulation is MJ0, which is a mixed 0-1 program with quadratic objective, solved by CPLEX MIQP solver. The second one is CMJ1, which is a quadratically constrained quadratic MIP (which is equal to MJ1 in the quadratic case) with constraints

$$y^2 \leq t$$

for each machine-job pair. Finally, the third one is CMJ2, the conic reformulation based on the strengthened inequality (6.7)

$$y^2 \leq tx,$$

which is already hyperbolic for the quadratic case.

We summarize the results of this experiment in Table 6.1. For each formulation we report the averages for the CPU seconds required to solve the continuous relaxation at the root node (rcpu), the integrality gap at the root node as percentage (rgap), the number of branch-and-bound nodes explored (nodes), and the total cpu seconds (cpu). We also report the number of instances out of five that could be solved to optimality within the time limit (opt) and if there are instances that could not be solved, we report the average gap between the best known lower bound and upper bound at termination (egap).

Whereas most of the instances could not be solved to optimality with either MJ0 or CMJ1 within the time limit, they were all solved within only a few

			MJ0						MJ1/CMJ1						CMJ2			
κ	n	m	rcpu	rgap	egap	opt	nodes	cpu	rcpu	rgap	egap	opt	nodes	cpu	rcpu	rgap	nodes	cpu
0.1	50	1	0.00	8.62	-	5	272	0.1	0.01	8.62	-	5	6,942	84.8	0.01	0.08	4	0.1
		2	0.00	7.31	-	5	4,887	1.4	0.03	7.31	2.15	2	27,394	837.2	0.02	0.12	16	0.6
		3	0.00	7.17	-	5	88,180	43.3	0.05	7.17	3.9	0	20,787	1045.4	0.04	0.07	14	0.9
	100	1	0.00	6.66	-	5	4,047	0.9	0.02	6.66	1.95	2	27,334	752.8	0.02	0.04	6	0.3
		2	0.01	7.07	0.32	3	1,423,838	666.1	0.06	7.07	5.41	0	15,069	1034.9	0.07	0.02	8	0.8
		3	0.01	6.49	2.17	0	1,241,615	1019.9	0.12	6.49	5.49	0	8,608	1094.7	0.12	0.02	16	2.6
	150	1	0.00	6.79	0.01	5	178,033	57.5	0.04	6.79	4.22	0	22,310	1041.04	0.04	0.02	9	0.6
		2	0.00	6.47	2.2	0	1,364,224	1008.58	0.13	6.47	5.59	0	8,336	1081.4	0.12	0.02	15	2.3
		3	0.02	3.88	3.71	0	820,907	1018.8	0.24	3.88	5.83	0	4,794	1096.4	0.18	0.01	7	2.4
	200	1	0.01	6.65	0.27	3	1,541,513	657.6	0.07	6.65	5.2	0	13,830	1053.0	0.07	0.00	2	0.4
		2	0.02	6.61	3.57	0	994,446	1006.5	0.18	6.61	6.02	0	5,634	1083.0	0.17	0.00	6	1.7
		3	0.02	6.49	4.56	0	622,768	1017.7	0.32	6.49	6.16	0	3,462	1101.6	0.28	0.00	6	2.9
0.2	50	1	0.00	5.54	-	5	901	0.1	0.01	5.54	0.16	4	31,915	400.0	0.01	0.02	6	0.1
		2	0.00	4.36	-	5	11,697	3.4	0.03	4.36	1.74	0	34,718	1024.5	0.02	0.05	20	0.7
		3	0.00	2.20	-	5	28,574	16.3	0.05	2.20	0.7	0	19,632	1062.5	0.04	0.06	19	1.1
	100	1	0.00	4.62	-	5	57,813	13.3	0.02	4.62	2.89	0	36,666	1036.8	0.02	0.00	5	0.2
		2	0.01	4.14	0.81	1	1,890,229	986.7	0.07	4.14	3.18	0	13,506	1066.8	0.08	0.01	5	0.6
		3	0.01	1.95	0.59	0	1,122,271	1018.2	0.11	1.95	1.48	0	8,238	1095.4	0.11	0.01	8	1.2
	150	1	0.00	4.58	0.30	3	1,908,417	595.9	0.05	4.58	3.71	0	18,983	1037.4	0.05	0.00	3	0.3
		2	0.01	3.97	1.92	0	1,298,011	1021.6	0.11	3.97	3.53	0	9,024	1101.7	0.11	0.00	5	1.0
		3	0.02	1.99	1.19	0	842,076	1015.7	0.18	1.99	1.76	0	5,151	1078.2	0.16	0.01	7	1.8
	200	1	0.00	4.60	1.23	0	2,716,401	1011.4	0.05	4.60	3.98	0	15,183	1067.8	0.06	0.00	5	0.6
		2	0.01	3.90	2.47	0	970,277	1020.1	0.16	3.90	3.65	0	5,911	1097.0	0.17	0.00	6	1.6
		3	0.02	1.92	1.37	0	584,661	1004.7	0.3	1.92	1.79	0	3,274	1090.1	0.26	0.00	8	3.1
Optimal			45.8%						10.8%						100%			

Table 6.1: Computational results for the quadratic case: $f(y) = ky^2$.

seconds using the strong conic formulation CMJ2. As expected the integrality gap is the same for MJ0 and CMJ1 and it takes longer time to solve CMJ1 than MJ0. Because the continuous relaxation of MJ0 is a QP, it is solved much faster than the quadratically constrained QP relaxation of CMJ1. Thus, a conic reformulation is not helpful when its relaxation has the same bound as for the QP. On the other hand, with conic formulation CMJ2, the integrality gap at the root node is reduced to almost zero, which in turn leads to a very small number of branch-and-bound nodes. Even though continuous conic relaxation takes longer

	CMJ1	CMJ2′	CMJ2
Hyperbolic inequalities	$y^2 \leq v_1$ $v_1^2 \leq ty$	$y^2 \leq v_1 v_2$ $v_1^2 \leq ty$ $v_2^2 \leq x$	$y^2 \leq v_1 x$ $v_1^2 \leq ty$

Table 6.2: Alternative formulations for the cubic case: $f(y) = ky^3$.

to solve than QP, it certainly pays off when solving the integer problem due to the bound strengthening.

We furthermore observe that the tighter the machine capacity, the higher is the integrality gap at the root node for all problems sizes. Whereas only the smaller instances can be solved with MJ0, conic reformulation CMJ2 scales well and is suitable for all instances.

The next experiment is on the cubic case $f(y) = ky^3$. Inequalities (6.6) and (6.7), used in CMJ1 and CMJ2 for this case, are

$$y^3 \leq t$$

and

$$y^3 \leq tx^2.$$

In addition, in order to see whether only a partial strengthening would be effective, we also compared CMJ1 and CMJ2 with a conic formulation with a simpler inequality

$$y^3 \leq tx.$$

We refer to this partially strengthened formulation as CMJ2′. In Table 6.2 we present the corresponding hyperbolic constraints for the three formulations.

We summarize the results with these formulations in Table 6.3. The first observation is that the integrality gap is larger for the cubic case than for the quadratic case. Out of 120 instances only 5 could be solved to optimality with formulation CMJ1. Even though the partially strengthened formulation CMJ2′

resulted some improvement with smaller integrality gap, most of the instances still could not be solved with it. On the other hand, all of the instances were solved within a few seconds with the strong formulation CMJ2.

			CMJ1						CMJ2′						CMJ2			
κ	n	m	rcpu	rgap	egap	opt	nodes	cpu	rcpu	rgap	egap	opt	nodes	cpu	rcpu	rgap	nodes	cpu
0.1	50	1	0.04	12.79	1.20	3	11,070	496.6	0.04	4.83	0.01	5	2,210	89.2	0.02	0.06	5	0.2
		2	0.09	10.55	5.58	0	9,085	1066.2	0.09	4.02	1.11	2	7,445	732.8	0.05	0.16	29	1.8
		3	0.15	10.76	7.62	0	5,991	1077.1	0.13	4.03	2.15	0	7,252	1082.6	0.07	0.18	68	6.4
	100	1	0.08	9.91	5.41	0	9,546	1069.7	0.08	3.76	1.19	2	8,616	830.8	0.04	0.07	13	0.9
		2	0.21	10.29	8.62	0	4,065	1079.9	0.18	3.89	3.00	0	4,984	1089.7	0.11	0.03	13	2.2
		3	0.34	9.44	8.46	0	2,554	1085.9	0.32	3.45	2.85	0	3,177	1093.6	0.16	0.04	37	8.7
	150	1	0.14	10.05	7.60	0	5,996	1078.1	0.13	3.85	2.39	0	7,021	1083.0	0.06	0.02	7	0.9
		2	0.33	9.42	8.65	0	2,509	1085.4	0.31	3.52	3.06	0	3,102	1094.2	0.17	0.02	25	6.0
		3	0.54	9.32	8.90	0	1,591	1084.5	0.52	3.35	3.03	0	1,996	1094.9	0.27	0.01	13	5.5
	200	1	0.14	9.83	8.25	0	5,056	1069.4	0.15	3.87	3.03	0	6,155	1071.1	0.08	0.00	5	0.8
		2	0.46	9.58	9.04	0	1,794	1085.6	0.41	3.63	3.38	0	2,249	1095.9	0.25	0.01	12	4.3
		3	0.77	9.35	9.06	0	1,112	1082.6	0.73	3.36	3.19	0	1,429	1093.2	0.40	0.01	10	6.4
0.2	50	1	0.03	8.43	2.68	2	17,654	808.0	0.03	2.89	0.12	4	7,833	305.1	0.01	0.04	8	0.3
		2	0.08	6.62	4.03	0	10,142	1068.3	0.08	2.14	0.52	1	10,104	965.2	0.05	0.06	22	1.4
		3	0.13	2.66	1.53	0	6,077	1077.2	0.15	0.95	0.08	4	4,058	623.8	0.07	0.04	22	2.3
	100	1	0.07	6.95	5.19	0	10,226	1072.8	0.08	2.34	1.37	0	11,341	1071.6	0.04	0.01	7	0.6
		2	0.17	6.41	5.46	0	4,623	1094.1	0.18	2.06	1.55	0	5,039	1097.9	0.11	0.01	14	2.2
		3	0.31	2.40	2.07	0	2,623	1094.7	0.32	0.79	0.55	0	3,032	1102.1	0.16	0.02	19	4.6
	150	1	0.11	6.93	5.93	0	6,702	1087.5	0.12	2.29	1.80	0	7,401	1086.6	0.07	0.00	3	0.5
		2	0.23	6.08	5.57	0	3,467	1084.3	0.24	1.95	1.69	0	3,686	1086.1	0.15	0.00	8	1.9
		3	0.46	2.43	2.25	0	1,649	1086.2	0.57	0.81	0.69	0	1,944	1094.0	0.26	0.01	14	5.3
	200	1	0.16	6.90	6.24	0	4,785	1090.5	0.17	2.30	1.96	0	5,409	1088.9	0.11	0.00	4	0.9
		2	0.39	5.97	5.69	0	2,092	1096.5	0.44	1.90	1.77	0	2,287	1098.7	0.24	0.00	7	3.1
		3	0.68	2.36	2.28	0	1,152	1089.8	0.76	0.79	0.73	0	1,398	1095.6	0.40	0.01	14	7.9
Optimal			4.2%						15.0%						100%			

Table 6.3: Computational results for the cubic case: $f(y) = ky^3$.

The final experiment is with the rational exponent case $f(y) = ky^{a/b}$. As in the previous experiment, we compared three formulations. Formulations CMJ1 and

CMJ2 are based on constraints (6.6) and (6.7), whereas the partially strengthened formulation CMJ2′ is based on inequality

$$y^a \leq t^b x^{(a-b-1)},$$

which is only a slight weakening of (6.7). Recall that a/b is generated from Uniform [1.0,3.0] with a single decimal digit; thus, $10 \leq b \leq a \leq 30$. As the number of additional variables and constraints in the conic reformulations are $O(\log_2 a)$, the size of the formulations for the rational exponent case is larger than for the quadratic and cubic cases. In Table 6.4 we report average size of the conic formulations for instances with 200 jobs.

		CMJ1			CMJ2′			CMJ2		
n	m	vars	cons	quad cons	vars	cons	quad cons	vars	cons	quad cons
	1	2623.2	1930.6	882.6	3096.2	2327.8	958.4	3248.8	2556.2	882.6
200	2	5276.2	3671.8	1787.6	6233.8	4484.6	1932.4	6563.8	4959.4	1787.6
	3	7868.2	5383.4	2655.6	9245.8	6555.8	2860.8	9767.8	7283.0	2655.6

Table 6.4: Conic formulation size for the rational case: $f(y) = ky^{a/b}$.

We report the summary results for the rational exponent case in Table 6.5. As expected from the size of the formulations, the conic quadratic relaxations for the rational case took longer to solve compared to the quadratic and cubic cases. Yet they were all solved within a couple of seconds. The performance of the three formulations is consistent with the cubic case. Whereas most problems could not be solved with the weak conic formulation CMJ1, all of them were solved fairly quickly (within 2 minutes) using the strong conic formulation CMJ2. The comparison between CMJ2 and CMJ2′ clearly shows that it is crucial to formulate the problems with the strongest possible constraints. Even a small weakening of the constraint renders most of the instances unsolvable. This is a rather interesting observation because typically the integrality gap of the partially strengthened formulation is quite small (almost always within one percent). The difficulty of solving nonlinear MIPs even with very small integrality gaps highlights the importance of carefully constructed formulations, perhaps, even more so than for

linear MIPs.

			CMJ1						CMJ2′						CMJ2			
κ	n	m	rcpu	rgap	egap	opt	nodes	cpu	rcpu	rgap	egap	opt	nodes	cpu	rcpu	rgap	nodes	cpu
0.1	50	1	0.08	5.02	0.00	5	597	61.6	0.08	1.08	0.00	5	60	6.2	0.05	0.11	8	0.7
		2	0.15	3.61	0.66	3	3,006	607.6	0.17	1.02	0.01	5	201	42.5	0.12	0.02	11	2.1
		3	0.27	4.09	2.12	0	3,084	1102.4	0.33	0.68	0.01	5	673	248.8	0.19	0.12	45	12.9
	100	1	0.17	3.63	0.67	3	3,599	762.3	0.18	0.90	0.01	5	217	48.9	0.12	0.03	9	1.8
		2	0.36	3.14	2.25	0	2,284	1097.0	0.42	0.72	0.08	4	1,480	782.4	0.32	0.03	14	6.1
		3	0.65	3.49	2.97	0	1,380	1094.8	0.71	0.80	0.48	0	1,275	1092.9	0.49	0.02	14	10.1
	150	1	0.26	3.86	2.19	0	3,207	1106.3	0.30	0.53	0.01	5	931	345.8	0.22	0.02	8	2.9
		2	0.55	3.57	3.11	0	1,405	1102.0	0.70	0.76	0.50	0	1,286	1100.8	0.46	0.01	42	85.4
		3	0.85	3.37	3.11	0	961	1092.6	0.93	0.72	0.55	0	900	1101.6	0.64	0.01	11	11.9
	200	1	0.35	3.17	2.36	0	2,408	1090.7	0.41	0.75	0.21	2	1,864	926.5	0.28	0.01	7	3.4
		2	0.76	3.35	3.02	0	1,205	1081.9	0.77	0.60	0.44	0	1,114	1081.6	0.59	0.01	12	10.7
		3	1.53	3.23	3.07	0	647	1102.0	1.57	0.69	0.60	0	605	1101.1	1.16	0.00	9	15.3
0.2	50	1	0.05	3.23	0.01	5	2,850	240.2	0.07	0.63	0.01	5	141	12.4	0.05	0.03	6	0.5
		2	0.15	2.09	0.55	3	4,809	951.9	0.17	0.50	0.01	5	310	68.7	0.13	0.04	9	1.7
		3	0.27	1.16	0.31	2	2,737	876.6	0.29	0.21	0.01	5	152	51.3	0.20	0.03	18	4.4
	100	1	0.14	2.54	1.42	0	5,732	1095.5	0.16	0.62	0.01	5	618	128.5	0.13	0.01	6	1.4
		2	0.32	2.17	1.63	0	2,658	1093.9	0.36	0.47	0.17	1	2,092	948.8	0.26	0.01	22	7.8
		3	0.54	1.02	0.78	0	1,530	1112.4	0.62	0.21	0.06	3	880	674.3	0.44	0.01	9	5.9
	150	1	0.23	2.55	1.86	0	3,474	1110.9	0.28	0.40	0.03	4	1,553	527.7	0.24	0.00	4	1.8
		2	0.57	2.14	1.89	0	1,626	1110.6	0.61	0.39	0.21	0	1,428	1108.2	0.48	0.01	7	5.0
		3	0.97	1.05	0.94	0	950	1111.4	1.03	0.17	0.09	0	892	1116.2	0.69	0.00	6	7.2
	200	1	0.34	2.45	2.08	0	2,633	1113.2	0.41	0.55	0.24	0	2,312	1109.6	0.32	0.00	4	2.7
		2	0.77	2.08	1.93	0	1,134	1111.3	0.92	0.35	0.28	0	1,023	1108.8	0.70	0.00	5	6.0
		3	1.32	0.95	0.89	0	680	1108.4	1.49	0.19	0.16	0	657	1104.9	0.98	0.00	7	11.3
Optimal			17.5%						49.2%						100%			

Table 6.5: Computational results for the general case: $f(y) = ky^{a/b}$.

6.4 Conclusion

In this study we have given a strengthened conic quadratic reformulation for the machine-job assignment problem with controllable processing times. We have written a technical report [4] on the findings of this chapter. The conic strengthening is sufficiently general to be applicable to other mixed 0-1 programs with separable convex objective or constraints with rational exponents and variable upper bounds. Our computations demonstrate the viability of using conic quadratic constraints that exploit the problem structure as a means for strengthening nonlinear mixed 0-1 programs just like the strong polyhedral cuts based on problem structure for linear MIPs. Availability of efficient and stable SOCP solvers will likely stimulate further research on the development of conic quadratic cuts. In the next chapter, we will consider controllable processing times in rescheduling context, where we will employ the strengthened formulation presented in this chapter. Our focus will be on the trade-off between total manufacturing cost and match up times after a disruption occurs to a given schedule.

Chapter 7

Match up Scheduling with Manufacturing Cost Considerations

Given a breakdown on a schedule being executed, match up scheduling aims to find a new schedule which matches up with the preschedule at some point in the future, called the match up time. At match up time the new schedule is in the same state as the preschedule. Catching up the preschedule as soon as possible is critical because there are usually other production planning decisions like material flow, tooling and purchasing at different levels of decision making hierarchy which are dependent on the results of the initial schedule. Hence, the objective is to create a new schedule which is as consistent as possible with the preschedule.

In this chapter, we consider a given schedule being executed on non-identical parallel machine environment. We assume that this initial schedule is achieved by solving a cost minimization problem subject to capacity constraints on the machines. We will consider the case in which one of the machines will be down for a certain time due to an unexpected machine breakdown. The problem is

to reschedule the remaining jobs, such that the new schedule will catch up the preschedule as soon as possible considering its manufacturing cost implications. Hence, we consider alternative match up time related objectives. Catching up the previous schedule earlier implies incurring more compression cost, so we will also consider the total manufacturing cost as another objective. We have dealt with the trade-off between cost and scheduling performance measures in previous chapters. In this chapter we deal with the trade-off between manufacturing cost and match up time performance.

A predictive approach in coping with disruptions is to leave idle time periods in schedules so that any unexpected event, such as breakdown or new job arrival, can be handled without much disruption. However, when the processing times are controllable, inserting idle times into the schedules requires selecting smaller processing times which causes higher manufacturing cost for the schedule. If no disruption occurs, the machines will be under-utilized.

The notation used throughout the chapter is below:

Parameters:

p_{ij}^u: Processing time upper bound that gives minimum cost for job j on machine i.

c_{ij}: Cost of job j on machine i.

u_{ij}: Maximum possible compression for job j on machine i.

$f_{ij}(y_{ij})$: Compression cost function for job j on machine i.

D_i: Available machining capacity on machine i.

$\mathcal{S}$: Preschedule on which a breakdown occurs.

$y_{ij}^{\mathcal{S}}$: Compression on the processing time of job j on machine i in $\mathcal{S}$.

J: Set of jobs not yet started processing at the time of disruption.

J_i: Subset of J scheduled on machine i in $\mathcal{S}$.

J_i^m: Subset of J_i that can form match-up point on machine i.

s_j: Start time of job j in $\mathcal{S}$.

P_j: Set of jobs including job j and its predecessors which can form match up point on the same machine in $\mathcal{S}$.

E_i: End time of the last job on machine i in $\mathcal{S}$.

Decision Variables:

x_{ij}: 1, if job j is assigned on machine i. 0, otherwise.

z_j: 1, if start time of job j in $\mathcal{S}$ is selected as match up time
0, otherwise.

y_{ij}: Compression on the processing time of job j on machine i.

In Section 7.1, we present different rescheduling approaches on a numerical example. In Section 7.2, we give the description of the rescheduling environment and formulations for the considered rescheduling problems. In Section 7.3, we discuss alternative formulations for the problems by using conic quadratic inequalities. In Section 7.4, we propose a heuristic algorithm to generate approximate efficient solutions for the problems. We give the results of our computational study in Section 7.5 and finalize with concluding remarks in Section 7.6.

7.1 Rescheduling with Controllable Processing Times: A Numerical Example

In order to clarify the distinctions between match-up rescheduling with fixed and controllable times, we demonstrate alternative rescheduling approaches on a numerical example in Figure 7.1. This is a parallel machine rescheduling example arising due to a breakdown on one of the machines. In this study, the problem is to reschedule a set of jobs on non-identical machines where each job has different cost and processing time data on different machines. However, in the numerical example, for the simplicity we assume that the jobs are of the same type and the machines are identical. Hence, we assume $p_{ij}^u = 2.0$ and $u_{ij} = 1$ for all i, j. The processing time of the jobs can be compressed by incurring additional manufacturing cost determined by the compression cost function which is $f_{ij} = 5y_{ij}^2$ for all i, j. The example starts with a preschedule with the minimum total compression cost. There are 15 jobs scheduled on three parallel CNC machines. When there is a machine breakdown on one of the machines, we illustrate how alternative rescheduling approaches can be used to remedy this unavailability

period using a different Gantt Chart representation for each one of them.

Gantt Chart 1 in Figure 7.1 belongs to the preschedule. In order to obtain this preschedule we first solve a machine-job assignment problem with cost minimization objective. Thus, we find the optimum machine-job assignments (5 jobs on each machine) and compression levels (0.2) on the processing times subject to given capacities on the machines. Selected machine-job allocation and processing times utilize the available machining time capacity, which is taken as (9.0) for this numerical example.

Gantt Chart 2 shows the breakdown on machine 1. In practice, we do not know when a machine breakdown occurs, but we can determine when it ends right after its occurrence. Therefore, we assume that the breakdown time is not known a priori, but immediately after the event occurs the down duration can be determined. In this numerical example, a breakdown occurs during the operation of job 2. At this point, the jobs 1,6 and 11 have been completed, and some are in process on other machines. These jobs, which are highlighted on the chart, will not be considered in rescheduling decisions since they are already completed. Gantt Chart 2 also gives the resulting schedule achieved by right-shifting the jobs not yet finished at time of breakdown on machine 1. The right-shift approach assumes fixed processing times and it is the simplest rescheduling strategy. However, the resulting schedule violates the capacity constraint by the duration of down time, which is 5.40. The total manufacturing cost of the jobs considered in rescheduling stays the same (2.40) as their cost in the preschedule since the processing times did not change. Since there is no inserted idle time in the preschedule, we will not be able to find a feasible solution by looking for an alternative machine-job allocation with fixed processing times. The only way to find a feasible schedule without violating the capacity constraints for this particular example is to consider the controllable processing times and machine reallocation decisions simultaneously.

Gantt Charts 3 to 6 in Figure 7.1 next show alternative rescheduling approaches with controllable processing times. The first approach finds a schedule with minimum sum of match-up times on the machines. In other words, the

objective is to minimize the length of the rescheduled portion of the preschedule. Minimum sum of match-up times for the schedule shown in Gantt Chart 3 is 16.20. This schedule is achieved by moving job 2 to machine 2 and re-planning the processing times of only six jobs represented by the dotted boxes. This is a minimum cost schedule for the sum of match-up time level of 16.20. As a result, the match-up times on machine 1, machine 2 and machine 3 are the end time of job 5, the starting time of job 9, and the starting time of job 4 in the preschedule, respectively. As it can be observed, the schedule on each machine is exactly the same as the preschedule beyond the match-up points. In contrast to the fixed processing time approach given in the previous charts, this approach neither violates the capacity constraint nor leaves unnecessary idle time. This schedule fully utilizes the available capacity on the machines and it has a manufacturing cost of 20.40.

Gantt Chart 4 in Figure 7.1 presents the schedule achieved by minimization the manufacturing cost for a given upper bound on the sum of match-up times. Sum of match-up times for the schedule is 19.80 and the total cost is 17.70. Hence, compared to the schedule in chart 3, by giving up from the sum of match-up times performance, which implies distributing the compression on more jobs, we improve the cost criterion. We re-assigned jobs 2 and 3 to different machines compared to the preschedule in this case. Gantt Chart 5 gives the schedule which minimizes the maximum of match-up times on the machines. Minimum of maximum match-up times is 5.40 for the given example. Minimizing the total cost subject to a maximum match-up time level of 7.20, we find the schedule shown in Gantt Chart 6 in Figure 7.1.

Alternative rescheduling approaches applied in this example show that using processing time controllability has definite advantages. The first one is that we can catch up the preschedule soon after the breakdown occurs and satisfy the due dates and production plans in the system. Processing time and machine-job assignment changes affect only a small part of the preschedule which helps to decrease the stress in the system. Under the fixed processing times assumption, it may not be possible to catch up the preschedule as shown in the numerical example. Secondly, we have the flexibility to generate different alternative schedules

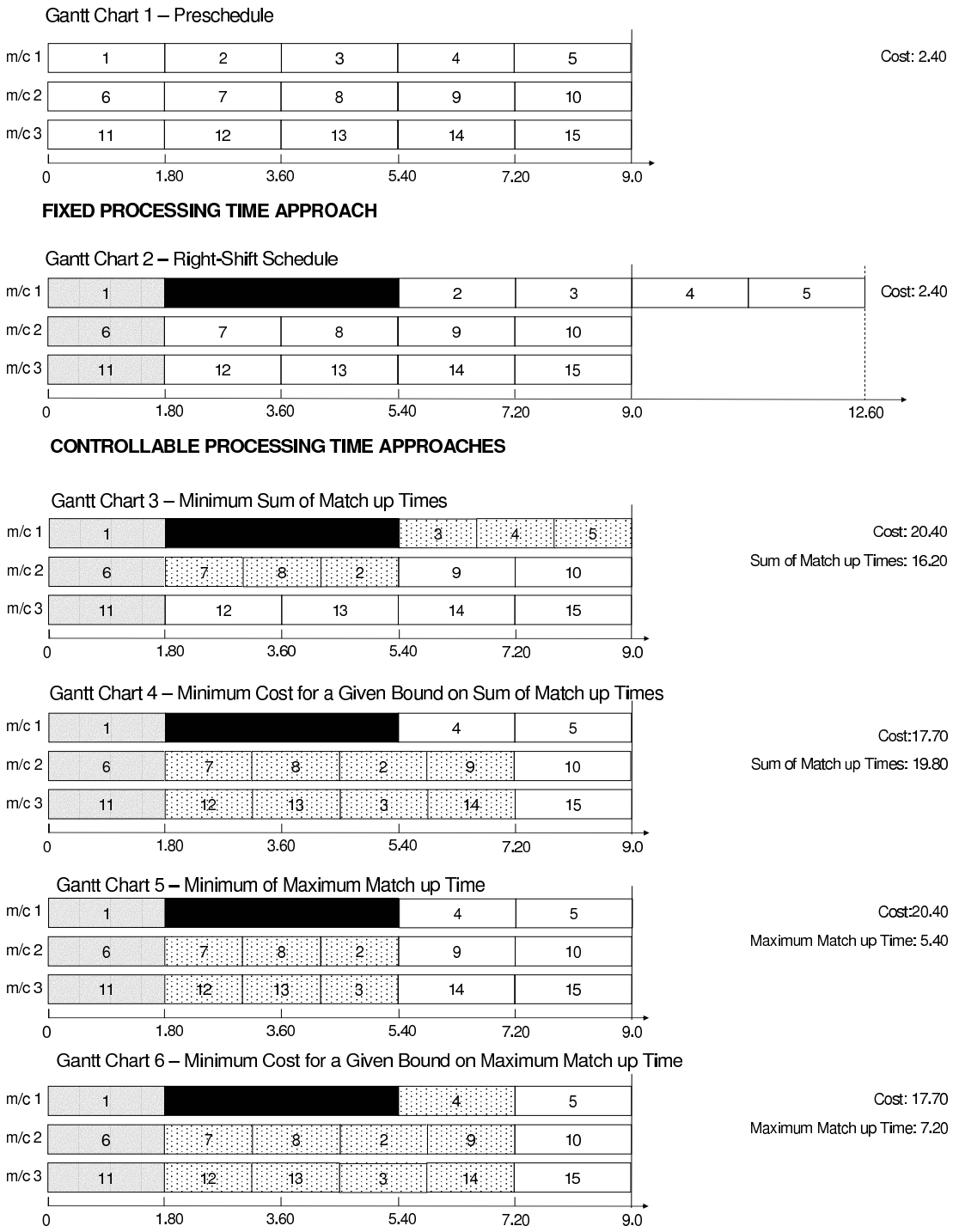

Figure 7.1: Alternative Reactive Scheduling Approaches

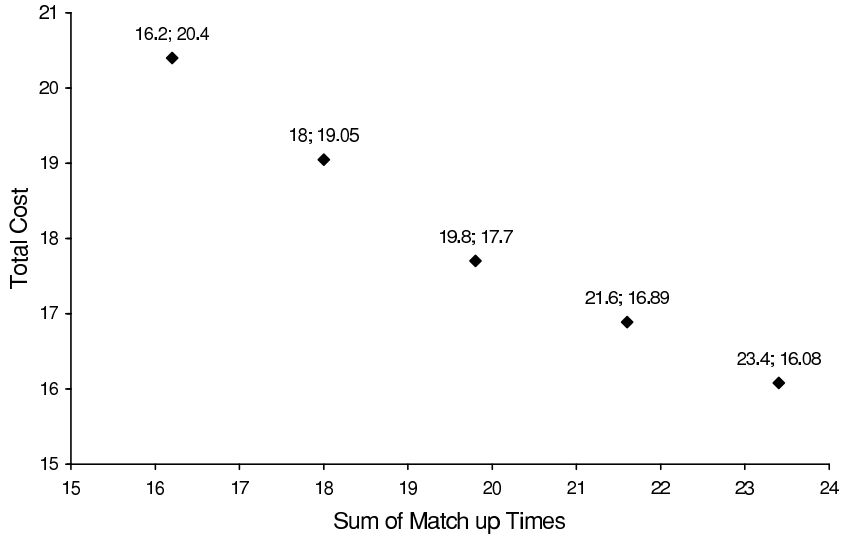

Figure 7.2: Efficient Solution Set for Total Cost and Sum of Match-up Times Objectives

with varying total cost and match-up time objectives. We can provide a set of alternative schedules to the decision maker. Thirdly, process time controllability provides a more robust system. With fixed processing times, a breakdown may lead to a rescheduling problem which is infeasible even for a preschedule with idle times.

For the considered numerical example, we present a set of efficient solutions illustrating the trade-off between the sum of match-up times and total cost objectives in Figure 7.2. Each solution on the chart is labeled by its sum of match-up times value and total cost value, respectively. The first solution corresponds to the schedule in Chart 3 with the minimum sum of match-up times but a high total cost. The third solution corresponds to the schedule in Chart 4. Similarly, we present a set of efficient solutions for the minimum of maximum match-up times and the manufacturing cost objectives. The first two solutions correspond to the schedules given in Charts 5 and 6, respectively. The third efficient solution shown in the figure has the minimum manufacturing cost for the problem. In the next section, we will describe the scheduling environment in detail and give problem definitions for different rescheduling objectives.

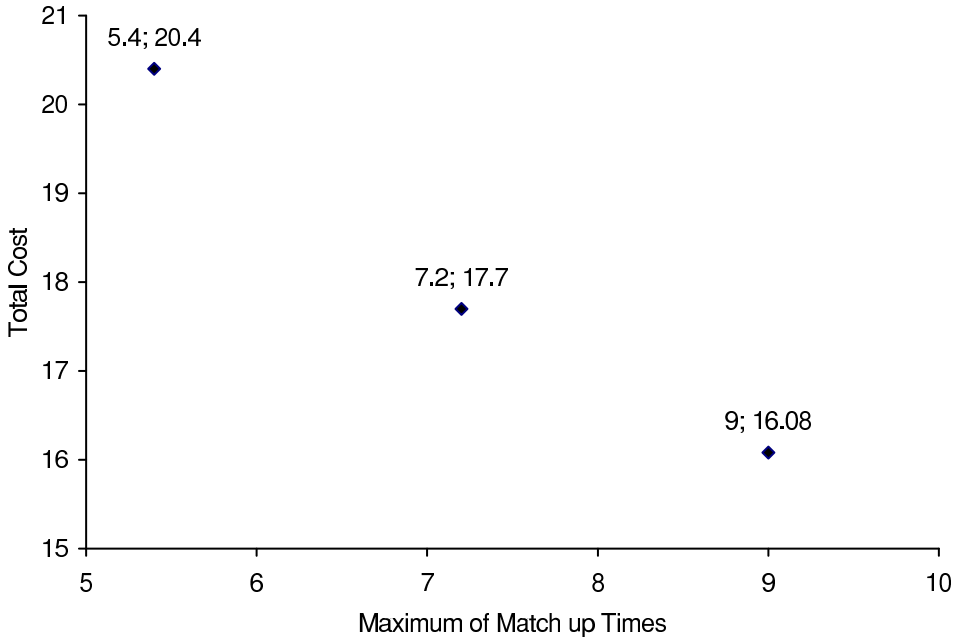

Figure 7.3: Efficient Solution Set for Total Cost and Minimum of Maximum Match- up Time Objectives

7.2 Scheduling Environment and Problem Definitions

We consider n jobs to be processed in non-identical parallel CNC machine environment. The term "non-identical" means that each job may have different processing time, upper bound on compression and manufacturing cost values and different compression cost function on different machines. There are m machines, each of which has a certain available capacity. There is a given preschedule $\mathcal{S}$ being executed in this environment. As we have mentioned in Section 7.1, $\mathcal{S}$ could be formed by solving the machine-job assignment problem to minimize total manufacturing cost objective.

We assume that during the execution of $\mathcal{S}$, there occurs a breakdown on one of the machines. This means the machine will be down for a certain time to be maintained or repaired. Given such a disruption, $\mathcal{S}$ is no longer executable. We assume that if the machine fails in the middle of processing a job then this job has to be reprocessed in its entirety, called the preempt-repeat case in the literature.

The interrupted job and all the other jobs which are not yet started processing on their machines have to be rescheduled. Rescheduling involves making new machine-job assignment and processing time compression decisions. These decisions can be made in order to catch up the preschedule at some point on each machine. Since we assume a non-preemptive machining environment, we select match-up times out of the start times previously determined in $\mathcal{S}$. The schedule, namely the sequence of the jobs and their start-end times, that follows a match-up time on a machine has to be the same as the preschedule.

The match-up time idea is used successfully to solve the rescheduling problems in the literature, but the critical issue is how to determine the match-up time on each machine. We could either minimize the sum of match-up times on each machine or minimize the maximum one regardless of their cost implications as will be discussed in Sections 3.1 and 3.2. Another alternative could be to provide alternative match-up schedules with varying time/cost trade-offs to the decision maker as will be discussed in Sections 3.3 and 3.4. In the previous numerical example summarized in Figure 1, by allowing machine reallocation and controllable processing times we could absorb the machine breakdown duration as soon as possible as in Gantt Charts 3 and 5 with two different match-up strategies albeit at a high manufacturing cost. With the maximum match-up time strategy, we expect to distribute the required compression amount more evenly among the selected jobs. This will eventually provide a better solution in terms of the manufacturing cost, but a higher number of jobs will be affected which decreases the stability of the system. We also suggest alternative nondominated solutions for this bi-criteria problem as given in Gantt Charts 4 and 6 for each strategy respectively. In the following sections, we formulate alternative match-up scheduling strategies to deal with the stated time/cost trade-off.

7.2.1 Minimize Sum of Match up Times

We first consider the problem of minimizing sum of match up times. This problem arises when the length of the rescheduled part of schedule $\mathcal{S}$ is critical and has to be minimized. We can formulate this problem as follows:

$$\min : \sum_i \sum_{j \in J_i^m} s_j z_j + \sum_i E_i (1 - \sum_{j \in J_i^m} z_j)$$

$$\text{(SM)} \quad \text{s.t.} \quad \sum_{j \in J} (p_{ij}^u x_{ij} - y_{ij}) \leq D_i \qquad i = 1, \ldots, m \quad (7.1)$$

$$y_{ij} \leq x_{ij} u_{ij} \qquad i = 1, \ldots, m \text{ and } j \in J \quad (7.2)$$

$$\sum_{i=1}^{m} x_{ij} = 1 \qquad \forall j \in J \quad (7.3)$$

$$\sum_{j \in J_i^m} z_j \leq 1 \qquad i = 1, \ldots, m \quad (7.4)$$

$$\sum_{j_2 \in P_{j_1}} z_{j_2} \leq x_{ij_1} \qquad i = 1, \ldots, m \text{ and } \forall j_1 \in J_i^m \quad (7.5)$$

$$(u_{ij_1} - y_{ij_1}^{\mathcal{S}}) \sum_{j_2 \in P_{j_1}} z_{j_2} \leq u_{ij_1} - y_{ij_1} \quad i = 1, \ldots, m \text{ and } \forall j_1 \in J_i^m \quad (7.6)$$

$$y_{ij_1}^{\mathcal{S}} \sum_{j_2 \in P_{j_1}} z_{j_2} \leq y_{ij_1} \qquad i = 1, \ldots, m \text{ and } \forall j_1 \in J_i^m \quad (7.7)$$

$$x_{ij} \in \{0, 1\}, \; y_{ij} \in \mathbb{R}_+ \qquad i = 1, \ldots, m, \text{ and } j \in J \quad (7.8)$$

$$z_j \in \{0, 1\} \qquad j \in \cup_i J_i^m \quad (7.9)$$

The objective to minimize is the sum of match-up times. Possible match-up times are the start times of jobs which can still be started at the same time and on the same machine as in $\mathcal{S}$. For instance, in the numerical example, in Figure 7.1, job 3 cannot form a match-up time since its machine is not available at that time, while job 10 can. End of horizon or ending time of the last job on a machine can also be the match-up time. Constraint set (7.1) guarantees that the new schedule does not exceed the available capacity on each machine. Constraint set (7.2) is the variable upper bounding constraints on the amount of compression, guaranteeing that processing time of a job on a machine can be compressed only if the job is assigned on that machine and also the compression cannot be greater than the upper bound u_{ij}. Constraint set (7.3) assigns each job to a machine. Start time of at most one of the jobs can be selected as a match-up point on each machine which is forced by constraint set (7.4). Constraint set (7.5) guarantees that the jobs following a selected match-up time on a machine has to

stay on the same machine in the new schedule. Constraints (7.6) and (7.7) fix the compression on a job which follows a match-up point at its compression in $\mathcal{S}$.

In Figure 7.1, for the considered example, solution to the above problem is given in Gantt Chart 3. Selected match-up points on the machines are the end time of job 5 on machine 1 and the start times of jobs 9 and 12 on machines 2 and 3, respectively, sum of which gives the minimum sum of match-up times. As can be seen from the example, the new schedule following the match-up points is exactly the same as the preschedule. The length of the rescheduled part is at its minimum. Another critical performance criterion is the maximum of match-up times which is considered in the next section.

7.2.2 Minimize Maximum of Match up Times

It may also be critical for the decision maker to catch the preschedule on all machines as soon as possible. Then, his objective will be to minimize the latest match-up time on the machines. We can formulate this problem as follows:

$$\begin{aligned} & \min: \quad W \\ \text{(MM)} \quad & \text{s.t.} \sum_{j \in J_i^m} s_j z_j + E_i(1 - \sum_{j \in J_i^m} z_j) \leq W \quad i = 1, \ldots, m \qquad (7.10) \\ & \text{and } (7.1) - (7.9). \end{aligned}$$

Constraints (7.10) bound the match-up time on each machine from above by the variable W, which is minimized. The other constraints are the same as the sum of match-up times problem. Solving the rescheduling problem for either of the two match-up time related objectives that we have discussed so far results extremely compressed processing times and costly machine-job assignments. Hence, the manufacturing cost for the resulting schedule could be too high. In the following sections we will consider the objective of total manufacturing cost, while bounding the match-up time criterion of the schedule.

7.2.3 Minimize Total Manufacturing Cost Subject to a Bound on Sum of Match up Times

Compressing the processing time of a job requires using additional resource for the job. In a flexible manufacturing system, reducing the processing time of an operation on a CNC machine by increasing the cutting speed and feed rate naturally leads to reduced tool life, and, consequently, increased machining cost. We can model the change in the machining cost due to processing time compression $y \geq 0$ as

$$f(y) = ky^{a/b},$$

where a and b are integers satisfying $a \geq b > 0$ and $k > 0$, so that f is an increasing and convex function of the compression. The function f reflects the relationship between compression and cost in that as one decreases the processing time of a job, it becomes more expensive to compress it further. Technical specifications of a job such as its length, diameter, required surface quality, as well as machine and tool type determine the cost coefficients k, a, and b as discussed in Kayan and Aktürk [54].

Minimizing the manufacturing cost necessitates making appropriate machine-job assignment and compression decisions. In match-up rescheduling, manufacturing cost due to process time compression and match-up time objectives are in conflict. Increasing the match-up time on a machine allows to distribute the required compression on more jobs which improves the cost performance. It may also permit a machine-job assignment with a lower cost. Having two conflicting objectives, one way to find efficient solutions is to minimize one of the objectives subject to the constraint that the solution value of the second objective cannot be worse than the given upper bound, and solve the overall problem as a single objective problem. This method known as the ϵ-constraint approach, as discussed in T'kindt and Billaut [83], has been widely used in the literature, because the decision maker can interactively specify and modify the bounds and analyze the influence of these modifications on the final solution. Below, we formulate the problem of minimizing the manufacturing cost subject to an upper bound T on

the sum of match-up times in the new schedule.

$$\begin{aligned} &\min : \sum_i \sum_{j \in J} (c_{ij} x_{ij} + f_{ij}(y_{ij})) \\ \text{(CSM)} \quad &\text{s.t.} \sum_i \sum_{j \in J_i^m} s_j z_j + \sum_i E_i (1 - \sum_{j \in J_i^m} z_j) \leq T \qquad (7.11) \\ &\text{and } (7.1) - (7.9). \end{aligned}$$

The objective function above is the sum of fixed costs and compression costs. The formulation above includes convex functions in the objective. Constraint (7.11) sets the upper bound on the sum of match-up times for the schedule and, thus, limits the size of the rescheduled portion of the preschedule.

7.2.4 Minimize Total Manufacturing Cost Subject to a Bound on Maximum Match up Time

Given an upper bound T on the maximum match-up time, one can set the match-up time on machine i to be $T_i = \max_{j \in J_i} \{s_j : s_j \leq T\}$. This is due to the fact that increasing the match-up time on a machine does not increase the total manufacturing cost of a schedule as discussed above. Defining the set of jobs that precede selected match-up times on their machines as J^T, we can formulate the problem of minimizing manufacturing cost subject to a given maximum match-up time as:

$$\begin{aligned} &\min : \quad \sum_i \sum_{j \in J^T} (c_{ij} x_{ij} + f_{ij}(y_{ij})) \\ \text{(CMM)} \quad &\text{s.t.} \sum_{j \in J^T} (p_{ij}^u x_{ij} - y_{ij}) \leq T_i \qquad i = 1, \ldots, m \qquad (7.12) \\ &y_{ij} \leq x_{ij} u_{ij} \quad i = 1, \ldots, m \text{ and } j \in J^T \qquad (7.13) \\ &\sum_{i=1}^{m} x_{ij} = 1 \qquad j \in J^T \qquad (7.14) \\ &x_{ij} \in \{0, 1\},\ y_{ij} \in \mathbb{R}_+ \qquad i = 1, \ldots, m \text{ and } j \in J^T \qquad (7.15) \end{aligned}$$

In this section, we have described the rescheduling environment and provided four different rescheduling problems to solve. In the computational analysis section, we will demonstrate that minimizing sum of match-up times and minimizing maximum match up time problems can be solved easily by commercial mixed-integer programming (MIP) solvers. However, it is usually difficult to solve cost minimization problems since they have convex cost terms in their objective functions. In the next section, we will discuss how the cost minimization problems can be reformulated in a stronger way by using conic quadratic inequalities.

7.3 Strong Conic Quadratic Formulations for Cost Minimization Problems

When solving the continuous relaxation of CSM and CMM formulations, convex cost terms in the objectives cause highly fractional optimal solutions which lie in the interior of their convex relaxation. Hence, branch-and-bound algorithms based on such relaxation usually require excessive branching to find integer feasible solutions. However, in reactive rescheduling, solution times for the problems is quite critical. Recent work by Aktürk et al. [4] on problems with separable convex objective and variable upper bounding constraints shows that conic quadratic inequalities can be employed for strengthening the formulations of such problems. Their approach is based on second-order cone programming (SOCP), which is also called conic quadratic programming. Alizadeh and Goldfarb [7] give an extensive review on the theory of SOCP and report the recent advances in this area. Existence of efficient SOCP algorithms implemented in branch-and-bound solvers allows us to reformulate CSM and CMM models as proposed by Aktürk et al. [4] and solve them efficiently. In this section, we discuss the implementation of this approach in the rescheduling problems with cost minimization objectives.

The first step of the reformulation is to put CSM into conic optimization problem form with linear objective and conic constraints. Thus, we first replace each convex cost function $f_{ij}(y_{ij})$ in the objective with an auxiliary variable t_{ij}

and move $f_{ij}(y_{ij})$'s into the constraint set as below:

$$\begin{aligned} &\min \sum_i \sum_{j\in J} (c_{ij}x_{ij} + k_{ij}t_{ij}) \\ \text{(CSM1)} \quad &\text{s.t. } y_{ij}^{(a_{ij}/b_{ij})} \le t_{ij}, \quad i = 1,\dots,m,\ j \in J \\ &\text{and (7.1)} - \text{(7.9) and (7.11).} \end{aligned} \tag{7.16}$$

As $c_{ij}, t_{ij} \ge 0$ and $b_{ij} > 0$ for all i, j, inequality (7.16) is equivalent to

$$y_{ij}^{a_{ij}} \le t_{ij}^{b_{ij}}. \tag{7.17}$$

which can be strengthened as

$$y_{ij}^{a_{ij}} \le t_{ij}^{b_{ij}} x_{ij}^{a_{ij}-b_{ij}}. \tag{7.18}$$

Because $a_{ij} \ge b_{ij}$, for $0 \le x_{ij} \le 1$, inequality (7.18) implies (7.17). Thus, we can substitute the inequalities with the stronger ones and obtain:

$$\begin{aligned} &\min \sum_i \sum_{j\in J} (c_{ij}x_{ij} + k_{ij}t_{ij}) \\ \text{(CSM2)} \quad &\text{s.t. } y_{ij}^{a_{ij}} \le t_{ij}^{b_{ij}} x_{ij}^{a_{ij}-b_{ij}}, \quad i = 1,\dots,m,\ j \in J \\ &\text{and (7.1)} - \text{(7.9) and (7.11).} \end{aligned} \tag{7.19}$$

Aktürk et al. [4] have proven that any extreme point of the continuous relaxation of CSM2 projects to an extreme point of the continuous relaxation of CSM. Since, CSM2 has a linear objective, by solving its continuous relaxation, one avoids all non-extreme points of the continuous relaxation of CSM. Thus, the source of difficulty due to convexity of the objective of CSM is eliminated.

Inequalities (7.17) and (7.18) can be represented by using conic quadratic constraints. This can be shown first by using the fact that an inequality of the form

$$r^{2^l} \le s_1 s_2 \cdots s_{2^l}, \tag{7.20}$$

for $r, s_1, \dots, s_{2^l} \ge 0$ can be expressed equivalently using $O(2^l)$ variables and $O(2^l)$ hyperbolic inequalities of the form

$$u^2 \le v_1 v_2, \ u, v_1, v_2 \ge 0 \tag{7.21}$$

[13] and then using the fact that each constraint (7.21) can be written as a second-order conic constraint

$$\|(2u, v_1 - v_2)\| \leq v_1 + v_2. \tag{7.22}$$

Aktürk et al. [4] have shown that inequalities (7.17) and (7.18) can be represented equivalently by using $O(\log_2 a_{ij})$ variables and $O(\log_2 a_{ij})$ conic quadratic constraints of the form (7.22). An example for the conic quadratic representation of a given inequality (7.18) is shown below.

Example 2. Consider the convex function $f(y) = y^{5/4}$. We first write inequality (7.18) as

$$y^5 \leq t^4 x,\ y, t, x \geq 0$$

then put it in the form of (7.20) as

$$y^8 \leq t^4 x y^3,\ y, t, x \geq 0.$$

which we can express equivalently by using the following hyperbolic constraints:

$$\begin{aligned} v_1^2 &\leq xy,\ \ x, y \geq 0, \\ v_2^2 &\leq y v_1,\ y, v_1 \geq 0, \\ y^2 &\leq t v_2,\ t, v_2 \geq 0. \end{aligned}$$

The hyperbolic constraints are then written in conic quadratic form as

$$\begin{aligned} \|(2v_1,\ y - x)\| &\leq y + x, \\ \|(2v_2, y - v_1)\| &\leq y + v_1, \\ \|(2y, t - v_2)\| &\leq t + v_2. \end{aligned}$$

We have discussed the strengthening and reformulation on CSM model. CMM can also be reformulated in the same way. Using strengthened conic quadratic formulations allows one to solve CSM and CMM very quickly as discussed in Section 7.5. In the next section, we describe a heuristic approach to generate approximately efficient solutions for the cost and the match-up objectives.

7.4 Generating A Set of Approximately Efficient Solutions: Heuristic Approach

In the rescheduling literature, a common approach to solve scheduling problems is by heuristics which can find a solution quickly. In the previous section, we have discussed a way of getting strong formulations for the considered cost minimization problems, CSM and CMM. In this section, we propose a heuristic search algorithm which generates a set of approximately efficient solutions that can be presented to the decision maker quickly. We consider two bi-criteria rescheduling problems. In the first problem, the conflicting objectives are the manufacturing cost and the sum of match-up times. In the second one, the conflicting objectives are total manufacturing cost and maximum match-up time.

We first give a step by step definition of the proposed heuristic search algorithm below. The algorithm generates approximately efficient solutions for the bi-criteria match-up scheduling problems under consideration.

Heuristic Search Algorithm
Step 1. Given a preschedule $\mathcal{S}$ and a disruption on one of the machines.
Step 2. Solve SM to find initial schedule S, hence job pool P, and match up times T_i.
Step 3. Repeat Steps 3.1-3.4 until $no\_possible\_update = TRUE$.
Step 3.1. Apply 1-move Algorithm.
Step 3.2. Apply 2-swap Algorithm.
Step 3.3. Report solution achieved.
Step 3.4. Apply Update Job Pool.

Heuristic Search Algorithm starts with an initial schedule $\mathcal{S}$ and a given disruption. We first solve the problem of minimizing sum of match-up times (or minimizing maximum of match-up times). The solution for this problem gives the initial match-up times and a job pool, i.e. the set of incomplete jobs that precede the match-up times. With the given match-up times, the algorithm applies the `1-move` and `2-swap` improvement algorithms on the current job pool

and records the solution. The next step is to expand the job pool by adding an appropriate job to the pool and the improvement algorithms are applied on the new job pool. Algorithm terminates when the job pool cannot be expanded, i.e. no jobs exist to be added to the pool. Heuristic Search Algorithm generates a set of solutions where each solution is an approximately efficient solution. In the following, we will describe a subproblem which motivates the improvement search steps of the algorithm and then we will describe the steps of Heuristic Search Algorithm in detail.

7.4.1 A Subproblem

We first define a subproblem that is solved at each iteration of Heuristic Search Algorithm. The solution to the subproblem is used by the `1-move` and `2-swap` improvements and for augmenting the job pool. The subproblem for machine i is described as the following: Given a match-up time T_i and a set of jobs J_c to be scheduled before T_i on machine i, find optimal compression levels for the jobs so that the total compression cost is minimized. The problem is formulated as

$$\begin{aligned} & \min \quad \sum_{j \in J_c} f_{ij}(y_{ij}) \\ \text{(COMPi)} \quad & \text{s.t.} \sum_{j \in J_c} (p_{ij}^u - y_{ij}) \leq T_i & (7.23) \\ & 0 \leq y_{ij} \leq u_{ij}, \;\; j \in J_c. & (7.24) \end{aligned}$$

COMPi is a nonlinear continuous resource allocation problem. Optimality properties and a solution method for the problem were given by Bretthauer and Shetty [15]. We can write the Karush-Kuhn-Tucker conditions for COMPi as below:

$$\frac{\partial f_{ij}}{\partial y_{ij}} - \lambda - \nu_j + \eta_j = 0, \;\; j \in J_c \tag{7.25}$$

$$\lambda \left(\sum_{j \in J_c} (p_{ij}^u - y_{ij}) - T_i \right) = 0 \tag{7.26}$$

$$\nu_j y_{ij} = 0, \;\; j \in J_c \tag{7.27}$$

$$\eta_j(y_{ij} - u_{ij}) = 0, \;\; j \in J_c \tag{7.28}$$

$$\nu_j \geq 0, \; \eta_j \geq 0, \;\; j \in J_c \tag{7.29}$$

$$\lambda \geq 0 \tag{7.30}$$

and inequalities (7.23)–(7.24). These conditions imply the following lemma which will be very useful in designing improvement steps and augmenting the job pool in our heuristic.

Lemma 7.1. *Let y_i^* and $(\lambda^*, \eta^*, \nu^*)$ be an optimal pair of solutions to COMPi. For each job j, we have the following:*

$$(\partial f_{ij}/\partial y_{ij})(y_{ij}^*) \begin{cases} \geq \lambda^*, & \text{if } y_{ij}^* = 0; \\ = \lambda^*, & \text{if } 0 < y_{ij}^* < u_{ij}; \\ \leq \lambda^*, & \text{if } y_{ij}^* = u_{ij}. \end{cases}$$

Because $(\partial f_{ij}/\partial y_{ij})(0) = 0$, the first part holds only if $\lambda^* = 0$, in which case the other parts imply that $y_{ij}^* = 0$ for j. Thus Lemma 7.1 states that whenever $\lambda^* > 0$, the partial derivative of the cost function for each job must be equal unless its compression is at its upper bound u_{ij}. Here, λ^* is the rate of change of the optimal cost as T_i changes. But also from the lemma, the rate of change of the optimal cost for each job with $0 < y_{ij}^* < u_{ij}$ is also equal to λ^*. This interpretation will be used in designing search steps in the following sections.

7.4.2 Job Pool

In the heuristic algorithm, at each iteration we work on a job pool which is the set of jobs to be rescheduled at that iteration. A job pool includes the interrupted job on the breakdown machine plus the set of jobs which are not started processing yet at the time of breakdown and precede the given match-up times in preschedule $\mathcal{S}$. For the bi-criteria problem with cost and sum of match-up times objectives, the initial job pool is determined by the match-up points

found by solving the SM problem in Section 7.2.1. Solving SM gives us the minimum attainable sum of match-up times. As shown in Gantt Charts 3 and 5 in Figure 7.1, minimizing sum of match-up times or maximum of match-up times requires extensively compressing a small set of jobs. The solution with minimum sum of match-up times has the highest manufacturing cost. Thus, the first approximately efficient solution we generate has the minimum possible sum of match-up times but its manufacturing cost is the worst of all efficient solutions.

For the problem with the objectives of minimizing total cost and maximum match up time we get the initial job pool by solving the MM problem and setting match up time on each machine to be the highest match-up time less than the maximum match-up time found by solving MM.

We augment the job pool at each iteration by adding a new job to the pool. The question is which job to add to the current job pool. Considering the jobs which are immediate successors of match-up points, there are at most m candidates. Adding a new job to the job pool increases the sum of match up times, but it may give us a schedule with a lower manufacturing cost after reallocating the jobs and solving the subproblems on each machine. We use the following rule to select the machine for increasing its match-up time. For each machine i, we have λ_i^*, the rate of change of the optimal cost by the change of match-up time, for the jobs scheduled before match-up point. Suppose that the job that immediately succeeds match-up point on machine i is j and that the compression on job j is y_{ij}^*, then we select the machine with the smallest

$$\Delta_i := \frac{k_{ij}\hat{y}_{ij}^{a_{ij}/b_{ij}} - k_{ij}{y_{ij}^*}^{a_{ij}/b_{ij}} - \lambda_i^*(\hat{y}_{ij} - y_{ij}^*)}{p_{ij}^u - y_{ij}^*},$$

where $\hat{y}_{ij} = \min((\partial f_{ij}/\partial y_{ij})^{-1}(\lambda_i^*), u_{ij})$. Δ_i is an estimate for the ratio of the cost change to the match-up time change that will be obtained by moving the match-up point to the start time of the next job. Since we are interested in the efficient frontier of manufacturing cost and sum of match-up time objectives, we select the machine that gives the maximum cost improvement estimate per unit match-up time change as outlined below:

Update Job Pool

Step 1. Given match up times T_i for each machine.
Step 2. If $T_i = E_i$ for all i then return FALSE. **Step 3.**Else do Steps 3.1 to 3.4.
Step 3.1. Calculate Δ_i for the machines with $T_i \neq E_i$.
Step 3.2. Select i^* with minimum Δ_i.
Step 3.3. $T_{i^*} = s_j$ where s_j is the start time of next job on i^*.
Step 4. Return TRUE.

For the maximum match-up time criterion, the match-up time increasing rule we use is to select the candidate job which has the smallest completion time. We have discussed the compression subproblem, formation of initial job pool and job pool augmentation rules used in our heuristic. We next describe the improvement search methods.

7.4.3 1-move Improvement Search

`1-move` method assumes that we have a schedule at hand with given match-up times and that the compression subproblem COMPi is solved for each machine. `1-move` method looks for cost improving move of a job in the job pool from its current machine to another machine. A `1-move` results compression cost improvement in its original machine since the compression for the remaining jobs can be decreased due to the additional capacity that becomes available when the job leaves. On the other hand, it increases the compression cost on the new machine as the jobs on this machine need to be compressed further to make up space for the new job to get a feasible schedule. Below we give a lower bound on the cost change for a given `1-move`.

Lemma 7.2. *(Lower Bound for a `1-move`) For a given schedule let λ_{i_1} and λ_{i_2} be optimal dual prices for COMPi$_1$ and COMPi$_2$, respectively, and $y_{i_1 j}$ be the compression of job j on machine i_1. Then, a lower bound for the cost change that will result by moving job j from machine i_1 to i_2 is as stated below:*

$$LB(j : (i_1 \rightarrow i_2)) = -\lambda_{i_1}(p_{i_1 j} - y_{i_1 j}) - c_{i_1 j} - f_{i_1 j}(y_{i_1 j}) + c_{i_2 j} + f_{i_2 j}(\hat{y}_{i_2 j}) + \lambda_{i_2}(p_{i_2 j} - \hat{y}_{i_2 j}),$$

where $\hat{y}_{i_2 j} = \min((\partial f_{i_2 j}/\partial y_{i_2 j})^{-1}(\lambda_{i_2}), u_{i_2 j})$.

Proof. The first three terms in $LB(j : (i_1 \rightarrow i_2))$ give a lower bound on the cost reduction by removing job j from machine i_1; whereas the last three terms give a lower bound on the cost increase by inserting job j into machine i_2. □

$LB(j : (i_1 \rightarrow i_2))$ gives us a lower bound on the change of manufacturing cost change for moving job j from i_1 to i_2. So if the lower bound is positive, then the corresponding `1-move` does not improve the cost of the schedule. On the other hand, if it is negative, then it may be possible to improve the cost of the schedule by reallocating this job to machine i_2. Hence, we employ a procedure to implement one moves on a given schedule as given below:

1-move Search Algorithm

Step 1. A given schedule S and a job pool P, initialize $improved \leftarrow TRUE$.

Step 2. While *improved* do the following Steps 2.1 to 2.7.

Step 2.1. Generate all feasible 1-moves for each j in P in S.

Step 2.2. Calculate LB for all moves.

Step 2.3. If $LB \geq 0$ for all feasible moves then stop. Else, go to Step 6.

Step 2.4. Make a list of moves with $LB < 0$ in ascending order of LB's.

Step 2.5. Initialize $found\_improving\_move \leftarrow FALSE$ and $end\_of\_list \leftarrow FALSE$.

Step 2.6. While $found\_improving\_move = FALSE$ and $end\_of\_list = FALSE$, do Steps 2.6.1 to 2.6.4.

Step 2.6.1. Do the next move in the list.

Step 2.6.2. If it is the last move in the list then $end\_of\_list \leftarrow TRUE$.

Step 2.6.3. Solve COMPi for the affected machines.

Step 2.6.4 New schedule is S'.

If $COST(S') < COST(S)$ then

$S \leftarrow S'$,

$found\_improving\_move \leftarrow TRUE$, $improved \leftarrow TRUE$.

Step 2.7. If $found\_improving\_move = FALSE$ then $improved \leftarrow FALSE$.

1-move Search Algorithm starts with an initial schedule and applies promising `1-moves` iteratively to obtain schedules with improved total manufacturing cost. The algorithm terminates when no improvement is possible for the current

schedule. In the next section we give a larger neighborhood search by a `2-swap` move.

7.4.4 2-swap Improvement Search

`2-swap` move is the exchange of two jobs, j_1 and j_2, between two machines i_1 and i_2, i.e. moving job j_1 from machine i_1 to i_2, and job j_2 in the opposite direction. We can consider a `2-swap` move as a combination of two `1-move`'s and calculate a cost change lower bound for a given 2-move as below:

Lemma 7.3. *(Lower Bound for a `2-swap`) For a given schedule let λ_{i_1} and λ_{i_2} be optimal dual prices for $COMPi_1$ and $COMPi_2$, respectively, and $y_{i_1j_1}$ and $y_{i_2j_2}$ be the compression of the jobs j_1 and j_2 on machine i_1 and i_2, respectively. Then, a lower bound for the cost change that will result by swapping jobs j_1 and j_2 between machines i_1 and i_2 is calculated as below:*

$$\begin{aligned} LB(j_1 \leftrightarrow j_2) = \lambda_{i_1}(p_{i_1j_1} - y_{i_1j_1} - p_{i_1j_2} + \hat{y}_{i_1j_2}) - c_{i_1j_1} - f_{i_1j_1}(y_{i_1j_1}) + c_{i_1j_2} + f_{i_1j_2}(\hat{y}_{i_1j_2}) \\ + \lambda_{i_2}(p_{i_2j_2} - y_{i_2j_2} - p_{i_2j_1} + \hat{y}_{i_2j_1}) - c_{i_2j_2} - f_{i_2j_2}(y_{i_2j_2}) + c_{i_2j_1} + f_{i_2j_1}(\hat{y}_{i_2j_1}), \end{aligned}$$

where $\hat{y}_{i_2j_1} = \min((\frac{\partial f_{i_2j_1}}{\partial p_{i_2j_1}})^{-1}(\lambda_{i_2}), u_{i_2j_1})$ *and* $\hat{y}_{i_1j_2} = \min((\frac{\partial f_{i_1j_2}}{\partial y_{i_1j_2}})^{-1}(\lambda_{i_1}), u_{i_1j_2})$.

Proof. Similar to the proof of Lemma 7.2. □

As in the `1-move` case, if the lower bound for a given `2-swap` is positive, then the `2-swap` does not reduce the cost of the current schedule. However, a `2-swap` with a negative lower bound has the potential for improvement. So starting from the most promising `2-swap`, we try possible two swaps for a given schedule and improve it iteratively. Hence, the algorithm for `2-swap` improvement search is same as 1-move Search Algorithm except that `2-swap`s are considered instead of `1-move`s.

In this section we have described a heuristic algorithm which generates a set of approximately efficient solutions for each bi-criteria problem considered. Each

iteration of the algorithm gives a new solution with increased match-up times, and for each new solution the manufacturing cost is either decreased or stays the same. At the end, Heuristic Search Algorithm produces a set of solutions which approximate the efficient frontier of match-up time versus manufacturing cost trade-off. In the next section, we will describe our computational results on this heuristic method and the exact solution approaches.

7.5 Computational Study

In the computational study, we tested the performance of alternative conic quadratic formulations introduced in Sections 7.2 and 6.2.3 for generating exact efficient solutions and of the heuristic algorithm described in Section 7.4 for generating approximate efficient solutions. We compared the computation time and solution quality of these alternative approaches on a set of randomly generated test problems. The test problems have varying number of jobs ($n = 50, 100$), machines ($m = 2, 3$), capacity factor ($\kappa = 0.25, 0.30$), and length of disruption ($ld = 2.0, 5.0$). The fixed cost (c_{ij}) for each job-machine pair is generated from Uniform[2.0,6.0]. The coefficients of the compression cost function ($f_{ij}(y_{ij}) = k_{ij} y_{ij}^{a_{ij}/b_{ij}}$) k_{ij} are generated from Uniform[1.0,3.0] and a_{ij}/b_{ij} from Uniform [1.1,3.1]. Processing time p_{ij}^u is generated from Uniform [1.0,3.0], whereas the compression bound u_{ij} from $p_{ij}^u \times$ Uniform [0.5, 0.9]. We set the machining capacity of each machine equal to

$$D_i = \kappa \times \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij}^u}{m} \ .$$

In order to construct preschedules we first solved the machine-job assignment problem that minimizes total manufacturing cost subject to given capacity for each machine. We sequenced the allocated jobs on each machine by using the shortest processing time (SPT) first rule, which gives the minimum total completion time for a given set of jobs on a machine.

Having formed a preschedule, we generated a breakdown on the schedule by randomly selecting a machine and a job on the selected machine so that the

breakdown occurs during the execution of the selected job and lasts for a duration generated from Uniform $[ld - 1.0, ld + 1.0]$. Rescheduling problem includes the interrupted job on the breakdown machine and all jobs planned to be started processing at the time of breakdown or later in the preschedule.

For each problem instance, we first ran Heuristic Search Algorithm which generates a set of approximately efficient solutions. In order to test the solution quality of the points generated by Heuristic Search Algorithm against efficient solutions to be generated by CSM and CMM problems, we first selected three solutions out of the solution set generated by the algorithm. $\bar{T}_{min}$ and $\bar{T}_{max}$ being the minimum and maximum values of sum of match-up times objective, for $k = 1, 2, 3$, the k^{th} solution selected is the one with the closest sum of match-up times value $\bar{T}_k$ to $\bar{T}_{min} + k \times \frac{\bar{T}_{max} - \bar{T}_{min}}{4}$. Then, for each k we solved CSM problem by setting $\bar{T} = \bar{T}_k$ and measured the relative gap between the cost of the heuristic solution and the cost of the exact efficient solution. We solved CSM by using the conic quadratic formulations, CSM1 and CSM2, given in Section 6.2.3. We followed the same approach for the maximum match-up time problem and solved formulations CMM1 and CMM2. All experiments were performed using ILOG CPLEX Version 10.1 on a 3 GHz Linux workstation with 512 MB memory with a 1000 CPU seconds time limit.

In the experimental runs, solving the machine-job allocation problem to form the preschedule took 16.14 CPU seconds on the average. After generating the disruptions, the average number of jobs to be rescheduled was 40.7 for the 50-job problems and 84.9 for the 100-job problems. Solving the SM model required 0.39 CPU seconds on the average, which was 0.16 for the MM model. Thus, we can solve the sum of match-up times or maximum match-up time problems in very short CPU times.

In Table 7.1 we give the computational results for the conic quadratic representations of CSM1 and CSM2 . We report the number of problems (out of 15) solved to optimality within the time limit (opt). We also report the average relative gap between the best known upper bound and the lower bound at time of termination (egap), the average number of branch-and-bound nodes explored

Table 7.1: Sum of Match-up Times

				CSM1				CSM2			
ld	κ	n	m	opt	egap (%)	node	cpu	opt	egap (%)	node	cpu
2.0	0.25	50	2	9	0.4	2177.9	613.5	15	0	35.7	9.8
			3	11	0.09	1095.9	527.5	15	0	188.7	71.1
		100	2	0	2.29	1498.3	1087	15	0.01	106	55
			3	0	0.52	742.3	1100	14	0.01	311.3	340.2
	0.30	50	2	12	0.2	1464.3	484.2	15	0.01	41.3	12.6
			3	12	0.17	625.3	343.6	13	0.07	280.9	176.1
		100	2	0	0.94	1340.8	1071.2	15	0.01	82.5	58.1
			3	5	0.08	540	856.3	13	0.01	463.4	502.9
5.0	0.25	50	2	6	1.02	3054.5	761.3	15	0	54.2	13.9
			3	6	0.35	1552.1	784.5	14	0.03	306.9	161.8
		100	2	0	2.68	1250.4	1073.2	15	0.01	120.1	80
			3	1	0.7	789.4	1049.3	11	0.05	624.4	567.6
	0.30	50	2	9	0.39	2381.2	664	15	0	35.9	9.6
			3	6	0.33	1197.4	729.8	14	0.01	496.1	213.7
		100	2	0	1.57	1374.1	1080.7	14	0.02	213.8	157.1
			3	1	0.22	621.5	1015.7	8	0.05	710.2	792.7
Average				4.9	0.75	1356.6	827.6	13.8	0.02	254.5	201.4

(node) and the average CPU time in seconds to solve the problems (cpu). Comparing the number of problems solved to optimality, we see that CSM1 could not solve most of the problems. However, CSM2 was able to solve all problems within given time limit in most of the cases. Whereas only 32% of all the problems could be solved to optimality with CSM1, 92% of the problems were solved to optimality with CSM2. The average optimality gap for CSM2 is quite low compared to CSM1. In worst case it is 0.07% for CSM2, while it is up to 2.68% for CSM1. CSM2 explored significantly fewer number of nodes to solve the problems compared to CSM1. Hence, formulation CSM2 is much more effective in solving the problems than CSM1.

Table 7.2 summarizes the results for the cost minimization problem subject to a given maximum match-up time bound. We solved conic quadratic representations of CMM1 and CMM2. The results show that CMM2 could solve all the

Table 7.2: Maximum Match-up Time Results

				CMM1				CMM2			
ld	κ	n	m	opt	egap(%)	node	cpu	opt	egap(%)	node	cpu
2.0	0.25	50	2	13	0.11	1734.5	309.6	15	0	2.9	1.3
			3	15	0.01	204.1	65.7	15	0	14.6	4.1
		100	2	5	1.38	1820.1	795.8	15	0	0	0.6
			3	8	0.18	712.9	625.6	15	0	6.9	6.4
	0.30	50	2	15	0.01	954.7	179.2	15	0	2.6	1
			3	15	0.01	40.7	15.9	15	0	15.9	4.5
		100	2	6	0.53	1879.1	707.9	15	0	2.4	2.1
			3	15	0.01	63.6	64.5	15	0	14	10.6
5.0	0.25	50	2	12	0.33	2287.8	423.9	15	0	1.1	0.7
			3	15	0.01	735.3	218.7	15	0	10.2	3.5
		100	2	2	1.74	2690.7	1046.8	15	0	2.2	2
			3	6	0.32	1036.6	782.9	15	0.01	11.7	7.9
	0.30	50	2	13	0.08	1835.9	320.4	15	0	2.8	1
			3	15	0.01	146.2	52	15	0	25.1	7.7
		100	2	5	0.73	2000.5	817.2	15	0	0.9	1.3
			3	15	0.01	157	152.7	15	0.01	17.6	14.2
Average				10.9	0.34	1143.7	411.2	15	0.001	8.2	4.3

problems to optimality within short CPU times by exploring a small number of branch-and-bound nodes. On the other hand, CMM1 could solve only 72% of the problems to optimality within the time limit of 1000 CPU seconds. The average CPU time spent to solve CMM1 is quite high compared to CMM2. The results show that CMM2 is a powerful tool to solve this match-up scheduling problem. We can even use the CMM2 to generate all the solutions in the efficient frontier. Our computational results indicate that advances in conic mixed-integer programming provide us strong formulations which can solve rescheduling problems with controllable processing times within reasonable CPU times.

In Table 7.3 we present the computational results for the heuristic algorithm, which generates approximate efficient solutions with varying cost and match-up time. In this table we report the average number of solutions generated by the algorithm (# sol), and the average CPU time in seconds (cpu). Since we solved

the CSM (CMM) problems for the sum of match-up times (maximum of match-up times) of the selected solutions generated by the heuristic, we can compare the manufacturing cost of the selected heuristic solutions with the corresponding optimal cost achieved by solving CSM (CMM). The relative gap is measured by $100 \times (z_H - z_{CSM})/z_{CSM}$, where z_H and z_{CSM} are the cost value of the heuristic solution and the optimal cost achieved by solving CSM, respectively. In the table, we report the mean, minimum and maximum of the relative gap (gap).

The results show that Heuristic Search Algorithm runs within few seconds and generates a large number of alternative solutions with varying time/cost trade-off to the decision maker. Such a solution set can be used to visualize an approximate efficient frontier. When we check the solution quality for cost versus sum of match-up times problem, we see that the average gap between the heuristic solution and the exact solution is less than 1% in most of the cases and is 1.58% at most. The worst gap performance is 6.23%, but in most of the cases it is close to 0.0%, implying an almost equivalent solution quality with exact approaches. For the maximum match-up time problem, the average gap for the heuristic is 0.73%. While the minimum gap can be as low as 0.0%, and we see the worst gap is 24.04%. For the cases where the decision maker may want to see the behavior of the efficient frontier quickly, the heuristic algorithm may be very valuable.

Finally, we have checked the effect of solving the sum of match-up time problem on the maximum match-up time objective and vice versa. If we solve the sum of match-up time problem, and check the maximum match-up time of the achieved schedules, we see that the resulting maximum match-up times deviate by 14.7% from optimum. Similarly, if we solve the maximum match-up time problem and check the sum of match-up times, we see a deviation of 25.5% from the optimum. We have also compared the manufacturing cost of the initial schedules achieved by the heuristic algorithm. The cost of the schedule achieved by solving maximum match-up time is 0.4% higher on the average than the schedule achieved by solving sum of match-up times objectives. However, in terms of manufacturing cost we do not see a statistically significant relationship between two approaches.

Table 7.3: Heuristic Algorithm Performance

				Sum of Match-up Times					Maximum Match-up Time				
				# sol	cpu	gap (%)			# sol	cpu	gap(%)		
ld	κ	n	m	mean	mean	mean	min	max	mean	mean	mean	min	max
2.0	0.25	50	2	19.8	1.26	0.04	0.00	0.21	27.6	2.05	0.28	0.00	1.06
			3	13.8	0.37	1.21	0.00	6.23	31.8	1.38	0.35	0.00	1.43
		100	2	70.2	8.58	0.21	0.00	0.66	67.6	12.03	0.17	0.08	0.24
			3	49	3.77	0.09	0.00	0.18	78.8	12.51	0.11	0.00	0.37
	0.30	50	2	18.6	0.64	0.39	0.00	1.92	31.4	2.30	0.21	0.00	0.62
			3	14.4	1.15	0.82	0.00	3.04	34.4	4.63	0.73	0.10	2.08
		100	2	50.4	3.77	0.09	0.00	0.55	72.6	10.22	1.17	0.00	15.75
			3	27.6	1.35	0.03	0.00	0.09	80.6	27.16	0.08	0.00	0.23
5.0	0.25	50	2	19.4	1.94	0.14	0.00	0.48	20.6	2.00	0.48	0.00	1.60
			3	21.2	1.16	0.49	0.00	1.41	25.0	1.72	1.94	0.00	18.00
		100	2	53.6	11.74	0.30	0.00	0.93	57.0	9.97	0.41	0.16	0.86
			3	57	10.33	0.26	0.00	0.92	70.2	15.60	0.38	0.07	1.04
	0.30	50	2	66.2	2.10	0.21	0.00	0.61	24.6	2.28	1.89	0.00	17.94
			3	19.4	1.23	1.58	0.17	4.11	31.2	2.44	0.36	0.00	1.25
		100	2	69.4	12.89	0.17	0.01	0.50	64.8	9.05	0.46	0.25	0.73
			3	48	6.26	0.97	0.00	2.68	74.0	24.80	2.64	0.09	24.04
Average				38.6	4.28	0.44	0.01	1.53	49.5	8.76	0.73	0.05	5.45

In this section, we have showed that we can efficiently solve match-up rescheduling problems with controllable processing times exactly by using recently developed reformulation techniques and commercial solvers. We have also observed that our heuristic is able to generate a very good approximation of the efficient frontier of match-up time and manufacturing cost quickly. In the next section, we conclude the chapter with some final remarks.

7.6 Conclusions

The results in this chapter clearly indicate that controllable processing times introduce an important flexibility to deal with machine breakdowns. As a result of this solution flexibility, we can either generate schedules which can catch up the

preschedule very quickly after the disruption albeit at a high manufacturing cost, or we can distribute the effects of disruption to the entire schedule by creating alternative time/cost trade-offs to the decision maker. We have introduced new match-up time related objectives, sum of match-up times and maximum match-up time, each which has its own advantages. It may be critical for the scheduler to balance the match-up time and manufacturing cost objectives, hence we provide effective exact mathematical programming formulations and heuristic algorithms to provide alternative solutions. Processing time controllability and convex cost functions complicate the scheduling problems significantly. Here, we have also seen that reformulation approach given in Chapter 6 can play an important role in mitigating this difficulty. In the next chapter, we will review the results of this thesis and give concluding remarks. We will also give possible future research directions.

Chapter 8

Conclusion

In this chapter, we first summarize the work we did in each chapter of this thesis. Then, we state the further possible research directions.

8.1 Concluding Remarks

In this thesis, we have studied a group of scheduling problems which involve making simultaneous processing time and scheduling decisions. As the selected processing time determines the manufacturing cost of a job, total manufacturing cost was a common objective for the problems we have considered in this thesis. We devoted Chapters 3, 4 and 5 to explore the trade-off between total manufacturing cost objective and various scheduling performance measures in different machine environments.

In Chapter 3, we considered total completion time and total weighted completion time objectives in single machine environment. In Chapter 4, we considered total completion time objective on identical parallel machines. We considered ϵ-approach for finding efficient solutions for those problems and hence solved $P1$ variant of the problem where we minimize total manufacturing cost subject to a given bound on the scheduling performance criterion. For those three problems in

Chapters 3 and 4, we proposed efficient formulations such that the continuous relaxation of which can be solved to integer local optimal solutions. Each problem had a nonlinear objective and a non-convex continuous relaxation which complicate solving the problems by continuous relaxation based branch-and-bound algorithms. For each of those problems, we have derived optimality properties which state the relationship between processing times of the jobs at optimality. Those properties led us to design heuristic algorithms which generate approximate efficient solutions for the problems. We have tested the algorithms against best known NLP solver MINOS, which showed that the solution quality of the algorithm is almost equivalent to the solution quality of MINOS. On the other hand, the heuristic algorithms were able to generate very large sets of alternative approximate efficient solutions in very short CPU times compared to MINOS, which is very important for bi-criteria decision making problems.

The next scheduling objective considered in Chapter 5 was the makespan in non-identical parallel machine environment. We considered minimizing total manufacturing cost subject to a given bound on the makespan of the schedule. Using the optimality property for single machine case, we derived lower bounding approaches which we employed in a branch-and-bound algorithm and heuristic search algorithms. For the cases where the exact approach is not efficient, we have proposed recovering beam search and improvement search heuristics for the problem. The results show that the search algorithms can achieve a solution within 1% of the optimum in very short CPU times.

Chapters 3- 5 were focusing on finding efficient solutions for the total manufacturing cost objective and various well known scheduling performance measures. For each performance measure, we have provided alternative methods to find efficient solutions. Next, in Chapter 6 we took a different point of view and considered a machine-job assignment problem under capacity constraints on the machines. This time, cost of a job was given by a convex function of the amount of compression on its processing time. Modeling the cost as a function of compression is frequently used in the literature. We studied the problem structure of the problem and proposed a strengthened conic quadratic formulation in Chapter 6. Proposed formulation has greatly reduced the root node gap, required CPU

time and required number of nodes opened to solve the problem. Furthermore, our approach was applicable to many problems having separable convex objective functions and variable upper bounding constraints from different areas.

Finally, in Chapter 7, we have considered controllable processing times in rescheduling problems arising in non-identical parallel machine environment. We showed that controllable processing times brings new solution alternatives to rescheduling problems due to the flexibility of making new processing time decisions after a disruption occurs. We also showed that some objectives such as total match-up time, maximum match up time and total manufacturing cost can be considered in this environment. Another approach is considering the trade-off between cost and match-up time related objectives. We employed conic quadratic formulations to solve cost minimization problems and proposed a heuristic search algorithm for the problems.

8.2 Future Research Directions

In this thesis, we have studied time/cost trade-off problems dealing with different scheduling performance measures in different machine environments. What distinguishes our work from the current literature is the nonlinear convex manufacturing cost function of processing time that we have considered. We observe that current studies mostly assume linear cost functions or more simpler nonlinear relationship between processing times and cost. As we have seen in this study, nonlinearity of the cost functions causes difficulty in finding exact optimal solutions to the problems even for the simplest scheduling performance measures such as total completion time and makespan. We can consider other scheduling performance measures such as the due date related measures with the total manufacturing cost objective. Furthermore, we can work on more complex machine environment settings to explore time/cost trade-off problems.

We can extend the work in this thesis to consider different scheduling environments and different objectives as discussed above. On the other hand, we can also

apply the methodology used here to analyze different problems. In this thesis, we have focused on finding efficient solutions for a set of problems by considering the ϵ-approach. Indeed, multi-objective optimization literature provides different point of views to analyze bi-criteria problems. We may consider characterizing supported solutions, finding where non-supported solutions lie and describing the behavior of the efficient frontier for the bi-criteria scheduling problems.

In this thesis, we have also shown that the advances in the field of nonlinear optimization namely the conic quadratic programming can be used to get efficient mathematical formulations for the scheduling problems with controllable processing times. In addition to the possible extensions in terms of selected performance measures and in terms of deeper multi-objective analysis, we can look for alternative, possibly stronger formulations of different time/cost trade-off problems.

Conic mixed integer programming is a new research area in the field of mixed-integer optimization. In context of a scheduling problem, we have shown that conic programming allows us stronger formulations for the problems with convex separable cost terms in the objective and variable upper bounding constraints in the constraint set. A direct extension of our work is exploring possible stronger formulations for the non-separable convex objectives with variable upper bounding constraints. One potential application area of our reformulation approach is the inventory/location problems with inventory holding costs.

In this thesis, we have also shown that controllable processing times can be considered in rescheduling problems to generate alternative reactive schedules against disruptions. In rescheduling we consider stability measures such as match-up time, number of reassigned jobs, etc. besides the scheduling performance measures. To the best of our knowledge, there is no published work done in rescheduling with controllable processing times. We considered working with total manufacturing cost, makespan and match-up time related measures in this thesis. Different stability measures such as completion time difference between preschedule and reactive schedule and number of re-planned jobs (new processing times assigned and/or reassigned to a different machine) can be considered in a new research problem.

One possible extension to our rescheduling work is including reassigning costs in the cost objective for the jobs reassigned to different machines than their machine in the preschedule. This cost may be occurred due the necessity of transporting a job from its original machine to another machine.

A direct extension of our rescheduling work in this thesis is considering alternative sequencing rules when forming the preschedule for the given problem in Chapter 7. We used the SPT rule to sequence the jobs on each machine in the current work. However, we think that more robust ordering rules can be designed which take into account the flexibility of the jobs which can be defined in terms of amount of possible compression and its cost, and so on. We can also consider the mean time to fail and mean time to repair data to find sequencing approaches for the machines.

In airline scheduling, it is likely to see a similar trade-off between match-up time and flight costs in case of disruptions to flight schedules. So, potentially, taking a similar approach to the one used in this thesis, airline rescheduling problems can be another topic of further research.

Bibliography

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

[2] M.S. Aktürk and E. Görgülü. Match-up scheduling under a machine breakdown. *European Journal of Operational Research*, 112:81–97, 1999.

[3] M.S. Aktürk and S. Gürel. Machining conditions based preventive maintenance. *International Journal of Production Research*, 45:1725–1743, 2007.

[4] M.S. Aktürk, A. Atamtürk, and S. Gürel. A strong conic quadratic reformulation for machine-job assignment with controllable processing times. Research Report BCOL 07.01, University of California-Berkeley, April 2007.

[5] O. Alagöz and M. Azizoğlu. Rescheduling of identical parallel machines under machine eligibility constraint. *European Journal of Operational Research*, 149:523–532, 2003.

[6] B. Alidaee and A. Ahmadian. Two parallel machine sequencing problems involving controllable job processing times. *European Journal of Operational Research*, 70:335–341, 1993.

[7] F. Alizadeh and D. Goldfarb. Second-order cone programming. *Mathematical Programming*, 95:3–51, 2003.

[8] A. Atamtürk and V. Narayanan. Conic mixed-integer rounding cuts. Research Report BCOL 06.03, University of California-Berkeley, December 2006. Shorter version forthcoming in the *Proceedings of IPCO 2007*.

[9] H. Aytug, M.A. Lawley, K. McKay, S. Mohan, and R. Uzsoy. Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research*, 161:86–110, 2005.

[10] M. Azizoğlu and O. Alagöz. Parallel-machine rescheduling with machine disruptions. *IIE Transactions*, 37:1113–1118, 2005.

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

[12] J.C. Bean, J.R. Birge, J. Mittenthal, and C.E. Noon. Match-up scheduling with multiple resources, release dates and disruptions. *Operations Research*, 39(3):470–483, 1991.

[13] A. Ben-Tal and A. Nemirovski. *Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications*. SIAM, 2001.

[14] 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.

[15] K.M. Bretthauer and B. Shetty. The nonlinear resource allocation problem. *Operations Research*, 43(4):670–683, July-August 1995.

[16] A. Broke, D. Kendrick, A. Meeraus, and R. Raman. *GAMS: A User's Guide*. GAMS Development Corporation, 2004.

[17] D. Cao, M. Chen, and G. Wan. Parallel machine selection and job scheduling to minimize machine cost and job tardiness. *Computers and Operations Research*, 32:1995–2012, 2005.

[18] M.T. Çezik and G. Iyengar. Cuts for mixed 0-1 conic programming. *Mathematical Programming*, 104:179–202, 2005.

[19] Z.L. Chen. Simultaneous job scheduling and resource allocation on parallel machines. *Annals of Operations Research*, 129:135–153, 2004.

[20] Z.L. Chen, Q. Lu, and G. Tang. Single machine scheduling with discretely controllable processing times. *Operations Research Letters*, 21:69–76, 1997.

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

[22] J.C. Choi and D.L. Bricker. Effectiveness of a geometric programming algorithm for optimization of machining economics models. *Computers and Operations Research*, 23:957–961, 1996.

[23] R.W. Conway, W.L. Maxwell, and L.W. Miller. *Theory of Scheduling*. Addison-Wesley, Massachusetts, 1967.

[24] J. Curry and B. Peters. Rescheduling parallel machines with stepwise increasing tardiness and machine assignment stability objectives. *International Journal of Production Research*, 43(15):3231–3246, 2005.

[25] R.L. Daniels and R.K. Sarin. Single machine scheduling with controllable processing times and number of jobs tardy. *Operations Research*, 37(6):981–984, November-December 1989.

[26] 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.

[27] E. Davis and J.M. Jaffe. Algorithms for scheduling tasks on unrelated processors. *Journal of the Association for Computing Machinery*, 28(4):721–736, 1981.

[28] F. Della Croce and V. T'kindt. A recovering beam search algorithm for the one-machine dynamic total completion time scheduling problem. *Journal of the Operational Research Society*, 53:1275–1280, 2002.

[29] F. Della Croce, M. Ghirardi, and R. Tadei. Recovering beam search: Enhancing the beam search approach for combinatorial optimization problems. *Journal of Heuristics*, 10:89–104, 2004.

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

[31] C.A. Floudas. *Nonlinear and Mixed-Integer Optimization*. Oxford University Press, New York, 1995.

[32] A. Frangioni and F. Gentile. Perspective cuts for a class of convex 0-1 mixed integer programs. *Mathematical Programming*, 106:225–236, 2006.

[33] 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.

[34] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic machine scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[35] O. Günlük, J. Lee, and R. Weismantel. MINLP strengthening for separable convex quadratic transportation-cost UFL. IBM Research Report RC24213, IBM, TJ Watson Research Center, Yorktown, NY, 2007.

[36] J.N.D. Gupta and J.C. Ho. Minimizing makespan subject to minimum flow time on two identical parallel machines. *Computers and Operations Research*, 28:705–717, 2001.

[37] J.N.D. Gupta and A.J. Ruiz-Torres. Generating efficient schedules for identical parallel machines involving flow-time and tardy jobs. *European Journal of Operational Research*, 167:679–695, 2005.

[38] S. Gürel and M.S. Aktürk. Scheduling parallel CNC machines with time/cost trade-off considerations. *Computers and Operations Research*, 34:2774–2789, 2007.

[39] S. Gürel and M.S. Aktürk. Optimal allocation and processing time decisions on non-identical parallel CNC machines: ϵ-constraint approach. *European Journal of Operational Research*, 183:591–607, 2007.

[40] S. Gürel and M.S. Aktürk. Scheduling preventive maintenance on a single CNC machine. *International Journal of Production Research*, 2007, to appear.

[41] S. Gürel and M.S. Aktürk. Considering manufacturing cost and scheduling performance on a CNC turning machine. *European Journal of Operational Research*, 177:325–343, 2007.

[42] N.G. Hall and C.N. Potts. Rescheduling for new orders. *Operations Research*, 52(3):440–453, May-June 2004.

[43] M.P. Hansen and A. Jaszkiewicz. Evaluating the quality of approximations to the non-dominated set. IMM technical report, Technical University of Denmark, 1998.

[44] W. Herroelen and R. Leus. Robust and reactive project scheduling: a review and classification of procedures. *International Journal of Production Research*, 42(8):1599–1620, 2004.

[45] W. Herroelen and R. Leus. Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research*, 165: 289–306, 2005.

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

[47] H. Hoogeveen and G.J. Woeginger. Some comments on sequencing with controllable processing times. *Computing*, 68:181–192, 2002.

[48] H. Hoogeven. Multicriteria scheduling. *European Journal of Operational Research*, 167:592–623, 2005.

[49] A. Janiak and M.Y. Kovalyov. Single machine scheduling subject to deadlines and resource dependent processing times. *European Journal of Operational Research*, 94:284–291, 1996.

[50] A. Janiak, M.Y. Kovalyov, W. Kubiak, and F. Werner. Positive half-products and scheduling with controllable processing times. *European Journal of Operational Research*, 165:416–422, 2005.

[51] K. Jansen and M. Mastrolilli. Approximation schemes for parallel machine scheduling problems with controllable processing times. *Computers and Operations Research*, 31:1565–1581, 2004.

[52] M.T. Jensen. Improving robustness and flexibility of tardiness and total flow-time job shops using robustness measures. *Applied Soft Computing*, 1: 35–52, 2001.

[53] M. Kaspi and D. Shabtay. Convex resource allocation for minimizing the makespan in a single machine with job release dates. *Computers and Operations Research*, 31:1481–1489, 2004.

[54] R.K. Kayan and M.S. Aktürk. A new bounding mechanism for the CNC machine scheduling problems with controllable processing times. *European Journal of Operational Research*, 167:624–643, 2005.

[55] P. Kesavan, R.J. Allgor, E.P. Gatzke, and P.I. Barton. Outer approximation algorithms for separable non-convex mixed-integer nonlinear programs. *Mathematical Programming*, 100(3):517–535, 2004.

[56] E. Kutanoğlu and İ. Sabuncuoğlu. Routing-based reactive scheduling policies for machine failures in job shops. *International Journal of Production Research*, 39:3141–3158, 2001.

[57] B.F. Lamond and M.S. Sodhi. Using tool life models to minimize processing time on a flexible machine. *IIE Transactions*, 29:611–621, 1997.

[58] C.Y. Lee, J.Y-T. Leung, and Y Gang. Two machine scheduling under disruptions with transportation considerations. *Journal of Scheduling*, 9:35–48, 2006.

[59] V.J. Leon, S.D. Wu, and R.H. Storer. Robustness measures and robust scheduling for job shops. *IIE Transactions*, 26(5):32–43, 1994.

[60] R. Leus and W. Herroelen. Scheduling for stability in single-machine production systems. *Journal of Scheduling*, 2008, to appear.

[61] B. Malakooti and J. Deviprasad. An interactive multiple criteria approach for parameter selection in metal cutting. *Operations Research*, 37(5):805–818, September-October 1989.

[62] M. Mastrolilli. Notes on max flow time minimization with controllable processing times. *Computing*, 71:375–386, 2003.

[63] S.V. Mehta and R.M. Uzsoy. Predictable scheduling of a job shop subject to breakdowns. *IEEE Trans. Robot. Autom.*, 14:365–378, 1998.

[64] K.G. Murty and S.N. Kabadi. Some NP-complete problems in quadratic and nonlinear programming. *Mathematical Programming*, 39:117–129, 1987.

[65] Y. Nesterov and A. Nemirovski. *Interior-point polynomial algorithms for convex programming*. SIAM, Philadelphia, 1993.

[66] C.T.D. Ng, T.C.E. Cheng, and M.Y. Kovalyov. Batch scheduling with controllable setup and processing times to minimize total completion time. *Journal of the Operational Research Society*, 54:499–506, 2003.

[67] S.Y. Nof and F.H. Grant. Adaptive/predictive scheduling: review and a general framework. *Production Planning and Control*, 2(4):298–312, 1991.

[68] E. Nowicki and S. Zdrzalka. A bicriterion approach to preemptive scheduling of parallel machines with controllable job processing times. *Discrete Applied Mathematics*, 63:237–256, 1995.

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

[70] R. O'Donovan, R.M. Uzsoy, and K.N. McKay. Predictable scheduling of a single machine with breakdowns and sensitive jobs. *International Journal of Production Research*, 37(18):4217–4233, 1999.

[71] P.S. Ow and T.E. Morton. Filtered beam search in scheduling. *International Journal of Production Research*, 26(1):35–62, 1988.

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

[73] M. Pinedo. *Scheduling: theory, algorithms and systems*. Prentice Hall, New Jersey, second edition, 2002.

[74] F.M. Ruiz Diaz and S. French. A note on SPT scheduling of a single machine with controllable processing times. Technical Report Note 154, Department of Decision Theory, University of Manchaster, 1984.

[75] M. Savelsbergh. A branch-and-price algorithm for the generalized assignment problem. *Operations Research*, 45:831–841, 1997.

[76] D. Shabtay. Single and two-resource allocation algorithms for minimizing the maximal lateness in a single machine. *Computers and Operations Research*, 31:1303–1315, 2004.

[77] D. Shabtay and M. Kaspi. Parallel machine scheduling with a convex resource consumption function. *European Journal of Operational Research*, 173(1): 92–107, 2006.

[78] D. Shabtay and M. Kaspi. Minimizing the total weighted flow time in a single machine with controllable processing times. *Computers and Operations Research*, 31:2279–2289, 2004.

[79] D. Shabtay and G. Steiner. A survey of scheduling with controllable processing times. *Discrete Applied Mathematics*, 155(13):1643–1666, 2007.

[80] D. Shmoys and E. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62:461–474, 1993.

[81] M.S. Sodhi, B.F. Lamond, A. Gautier, and M. Noël. Heuristics for determining economic processing rates in a flexible manufacturing system. *European Journal of Operational Research*, 129:105–115, 2001.

[82] F.W. Taylor. On the art of cutting metals. *Transaction ASME*, 1906.

[83] V. T'kindt and J.-C. Billaut. *Multicriteria Scheduling: Theory, Models and Algorithms*. Springer, Berlin, second edition, 2006.

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

[85] A. Türkcan. *Essays on scheduling with controllable processing times in flexible manufacturing systems*. PhD dissertation, Bilkent University, Department of Industrial Engineering, April 2003.

[86] D. Tuyttens, J. Teghem, P.H. Fortemps, and K. Van Nieuwenhuyze. Performance of the MOSA method for the bicriteria assignment problem. *Journal of Heuristics*, 6:295–310, 2000.

[87] L.N. Van Wassenhove and K.R. Baker. A bicriterion approach to time/cost tradeoffs in sequencing. *European Journal of Operational Research*, 11:48–54, 1982.

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

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

[90] G.E. Vieira, J.W. Herrmann, and E. Lin. Rescheduling manufacturing systems: A framework of strategies, policies and methods. *Journal of Scheduling*, 6:39–62, 2003.

[91] G. Wan, B.P.C. Yen, and C.L. Li. Single machine scheduling to minimize total compression plus weighted flow cost is NP-hard. *Information Processing Letters*, 79:273–280, 2001.

[92] J. Wu and S. Azarm. Metrics for quality assessment of a multi-objective design optimization solution set. *Journal of Mechanical Design*, 123(1):18–25, 2001.

[93] B. Yang. Single machine rescheduling with new jobs arrivals and processing time compression. *International Journal of Advanced Manufacturing Technology*, 34(3-4):378–384, 2007.

[94] L. Yedidsion, D. Shabtay, and M. Kaspi. A bicriteria approach to minimize maximal lateness and resource consumption for scheduling a single machine. *Journal of Scheduling*, 10:341–352, 2007.

[95] F. Zhang, G. Tang, and Z.L. Chen. A 3/2-approximation algorithm for parallel machine scheduling with controllable processing times. *Operations Research Letters*, 29:41–47, 2001.

[96] E. Zitzler. *Evolutionary algorithms for multi-objective optimization: Methods and applications*. Shaker Verlag, Aachen, Germany, 1999. Ph.D. dissertation.

[97] E. Zitzler and L. Thiele. Multiobjective optimization using evolutionary algorithms-a comparative case study. In A.E. Eiben, Th. Bck, M. Schoenauer, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature (PPSN V)*, number 1498 in Lecture Notes in Computer Science, pages 292–301, Berlin, Germany, 1998. Springer.

[98] E. Zitzler, L. Thiele, M. Laumans, C. Fonseca, and V.G. Fonseca. Performance assessment of multi-objective optimizers: an analysis and review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, 2003.

Appendix A

Single Machining Operation Problem (SMOP)

SMOP is the manufacturing cost minimization problem for the turning operation. Decision variables for the problem are the cutting speed (v_i) and the feed rate (f_i). The job to be machined has certain specifications like job diameter, length, depth of cut and maximum allowable surface roughness and a selected cutting tool. A cutting tool has certain technical coefficients.

There are three constraints in the problem. The first one is the tool life constraint which comes from the limitation that each job can use at most one tool. The second constraint is the machine power constraint that comes from the maximum applicable cutting power by the machine. The last one, the surface roughness constraint, satisfies the surface quality requirement for the operation. The geometric programming model for the problem to minimize manufacturing cost (i.e. the sum of the operating and the tooling costs) for job i is as follows:

$$\begin{aligned} \min \text{Cost} &= C_o \cdot p_i + C_t \cdot U_i \\ &= C_1 v_i^{-1} f_i^{-1} + C_2 v_i^{(\alpha-1)} f_i^{(\beta-1)} \\ \text{s.t.} \quad & C_t' v_i^{\alpha-1} f_i^{\beta-1} \leq 1 \qquad \text{(Tool life constraint)} \end{aligned}$$

$$C_m' v_i^b f_i^c \leq 1 \quad \text{(Machine power constraint)}$$
$$C_s' v_i^g f_i^h \leq 1 \quad \text{(Surface roughness constraint)}$$
$$v_i, f_i > 0$$

where $C_1 = \frac{\pi D_i L_i C_o}{12}$, $C_2 = \frac{\pi D_i L_i d_i^{\gamma} C_{t_i}}{12 C_i^{TL}}$, $C_t' = \frac{\pi D_i L_i d_i^{\gamma}}{12 C_i^{TL}}$, $C_m' = \frac{C_m d_i^e}{H}$ and $C_s' = \frac{C_s d_i^l}{S_i}$.

Theorem A.1. *(Aktürk and Avcı [1]) At least one of the surface roughness and machine power constraints is binding at optimality for SMOP.*

Theorem A.2. *(Kayan and Aktürk [54]) The surface roughness constraint must be tight at the optimal solution.*

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 worldwide 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 Prof. M. Selim Aktürk on his graduate study. He got his M.S. degree in 2003. He had been on the grant 2211 awarded by TÜBİTAK during most of his Ph.D. study. While he was on another TÜBİTAK grant (2214) in Fall 2006, he visited the IEOR Department of University of California, Berkeley, where he worked with Assoc. Prof. Alper Atamtürk.