A BICRITERIA RESCHEDULING PROBLEM ON UNRELATED PARALLEL MACHINES: NETWORK FLOW AND ENUMERATION BASED APPROACHES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

MELİH ÖZLEN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
INDUSTRIAL ENGINEERING

NOVEMBER 2006

Approval of the Graduate School of Natural and Applied Sciences

_____

Prof. Dr. Canan Özgen

Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Doctor of Philosophy.

_____

Prof. Dr. Çağlar Güven

Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Doctor of Philosophy.

_____

Prof. Dr. Meral Azizoğlu

Supervisor

Examining Committee Members

Prof. Dr. Murat Köksalan	(METU, IE)	_____
Prof. Dr. Meral Azizoğlu	(METU, IE)	_____
Prof. Dr. İhsan Sabuncuoğlu	(Bilkent Univ., IE)	_____
Assoc. Prof. Dr. Canan Sepil	(METU, IE)	_____
Asst. Prof. Dr. Ayten Türkcan	(METU, IE)	_____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : Melih Özlen

Signature :

ABSTRACT

A BICRITERIA RESCHEDULING PROBLEM ON UNRELATED PARALLEL MACHINES: NETWORK FLOW AND ENUMERATION BASED APPROACHES

Özlen, Melih
Ph.D., Department of Industrial Engineering
Supervisor : Prof. Meral Azizoğlu

November 2006, 99 pages

This study considers bicriteria approaches to the minimum cost network flow problem and a rescheduling problem where those approaches find their applications.

For the bicriteria integer minimum cost network flow problem, we generate all efficient solutions in two phases. The first phase generates the extreme supported efficient points that are the extreme points of the objective space of the continuous bicriteria network flow problem. In the second phase, we generate the nonextreme supported and unsupported efficient points by Integer Programming Based approaches.

Our rescheduling problem considers parallel unrelated machine environments. The criteria are the total flow time as an efficiency measure and the total reassignment cost as a stability measure. We show that the problems that address linear functions of the two criteria can be represented by bicriteria network flow models. To generate all efficient solutions, we use a Classical Approach that is based on the optimal solutions of the singly constrained network flow problem and provide a Branch and Bound approach that starts with extreme supported efficient set and uses powerful bounds. To find an

optimal solution to any nonlinear function of the two criteria, we provide a Branch and Bound approach and an Integer Programming Based approach that eliminates some portions of the efficient set that cannot provide improved solutions.

We contribute both to the network flow and scheduling literature by proposing algorithms to the bicriteria network flow models and applying them to a rescheduling problem that is bicriteria in nature.

The results of our extensive computations with up to 100 jobs and 12 machines have revealed that, the Branch and Bound algorithm finds the efficient set in less computational effort compared to the classical approach. In minimizing a nonlinear function of the two criteria both IP Based approach and Branch and Bound algorithm perform quite satisfactory.

Keywords: Bicriteria Network Flows, Rescheduling, Parallel Unrelated Machines, Total Flowtime, Total Reassignment Cost

ÖZ

İLGİSİZ PARALEL MAKİNALARDA İKİ KRİTERLİ YENİDEN ÇİZELGELEME PROBLEMİ: AĞ AKIŞ VE BİRERLEME TABANLI YAKLAŞIMLAR

Özlen, Melih
Doktora, Endüstri Mühendisliği Bölümü
Tez Yöneticisi : Prof. Meral Azizoğlu

Kasım 2006, 99 sayfa

Bu çalışmada enaz maliyetli ağ akış problemine iki kriterli yaklaşımlar, ve bu yaklaşımların uygulandığı bir yeniden çizelgeleme problemi ele alınmaktadır.

İki kriterli kesikli enaz maliyetli ağ akış probleminin tüm verimli noktaları iki aşamada bulunmuştur. İlk aşamada sürekli iki kriterli enaz maliyetli ağ akış probleminin amaç uzayında köşe noktalarda yer alan, köşe destekli verimli noktalar bulunmuştur. İkinci aşamada, destekli olmayan verimli noktalar, ve köşe olmayan destekli verimli noktalar, Tam Sayılı Programlamaya dayalı yaklaşımlarla bulunmuştur.

Yeniden çizelgeleme problemimiz ilgisiz parallel makinalar ortamlarında ele alınmıştır. Verimlilik ölçütü olarak toplam akış zamanı kriteri, ve tutarlılık ölçütü olarak toplam yeniden atama maliyeti kriteri kullanılmıştır. İki kriterin doğrusal fonksiyonunu ele alan problemlerin iki kriterli enaz maliyetli ağ akış modelleri kullanılarak ifade edilebileceği gösterilmiştir. Tüm verimli noktaların yaratılması için, tek kısıtlı ağ akışı probleminin en iyi çözümlerine dayalı Klasik Yöntem kullanılmıştır, ve köşe destekli verimli noktalarla başlayan Dal ve Sınır yöntemi önerilmiştir. İki

kriterin her hangi bir doğrusal olmayan fonksiyonun en iyi çözümünü bulmak için, verimli kümenin daha iyi çözümler sağlayamayacak kısımlarını eleyen, Tam Sayılı Programlamaya dayalı bir yöntem, ve Dal-Sınır yöntemi önerilmiştir.

Bu çalışmada iki kriterli ağ akış problemleri için çözüm yöntemleri önerilerek, ve bu önerilen yöntemler, doğası gereği iki kriterli olan bir yeniden çizelgeleme problemi üzerinde uygulanarak, hem ağ akışları ve hem de çizelgeleme alanlarına katkı yapılmıştır.

100 iş, ve 12 makinalı problemleri çözebilen geniş çaplı deneysel çalışmamızın sonuçları, Dal-Sınır yönteminin, Klasik yöntemle karşılaştırıldığında, tüm verimli noktaları daha az çözüm zamanı harcayarak bulduğunu göstermiştir. Doğrusal olmayan bir fonksiyonun en azlanmasında, Tam Sayılı Programlama tabanlı yöntem ve Dal-Sınır algoritmasının her ikisinin de oldukça başarılı oldukları görülmüştür.

Anahtar Kelimeler: İki Kriterli Ağ Akışları, Yeniden Çizelgeleme, Paralel İlgisiz Makinalar, Toplam Akış Zamanı, Toplam Yeniden Atama Maliyeti

To My Family

ACKNOWLEDGMENTS

I would like to thank my Supervisor Prof. Meral Azizoğlu for her guidance, contributions, and support throughout this study. She spent every piece of effort to improve our research, improve me, and make my life better. She will always be an ideal figure of a researcher, a teacher, and a supervisor for me all through my life.

I would like to thank Prof. Murat Köksalan, Prof. İhsan Sabuncuoğlu, Assoc. Prof. Canan Sepil and Asst. Prof. Ayten Türkcan for their valuable suggestions, and contributions to this study.

I would like to thank to my family, without their support I couldn't be where I am, and couldn't be who am I. Namely, I thank to my parents Serap Özlen, and Murat Özlen, and my brother Erkin Özlen.

In my years of assistantship and Ph. D. research, Tevhide Altekin, Sakine Batun, Pelin Bayındır, Baykal Hafızoğlu, Tülin İnkaya, Fatma Kılınç, Zeynep Kirkizoğlu, and Ayten Türkcan provide every mean of support to me, and their existence make my life easier, and make me feel that I am a member of a large family. I would also like to thank to my friends Julide Akşiyote, Barış Canlı, Makbule Çubuk, Onur Sarıoğlu, Selim Sandıkçıoğlu, and Tuba Pınar Yıldırım, for their support even in the hardest parts of this study.

Although their names are not mentioned individually here, I am grateful to all the professors, friends, colleagues, and students at the department for their valuable contributions to my life, and for their presence which I will always miss.

Finally, I would like to thank to a very special person, Berrak Dağ. I am very fortunate, to have her in my life, she is my energy and my inspiration for life. She constantly believed in and motivated me, so that I can finally complete this study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Scheduling, network flows, and multi-criteria optimization are well recognized areas in the Operations Research literature. These research areas are motivated by the practical problems that arise in a wide range of situations.

Network flows find many applications in scheduling area as many scheduling problems have network representations, where the jobs may be accepted as activities and the flows may be a representative of the sequence. Hence, advances in network flow theory trigger the development of efficient solution procedures for the scheduling problems having network flow representations.

Multi-criteria optimization is an important area of operations research, which finds its application on both network flow problems and scheduling problems. Many network flow problems like assignment, transportation, minimum cost network flow, might have several concerns like safety, reliability, resource usages in addition to the total cost criterion. The incorporation of those concerns necessitates the multi-criteria formulation of the network flow problems.

Many scheduling problems have both producer and consumer related concerns that may necessitate their simultaneous consideration in a multi-criteria context. Rescheduling is an important scheduling area where multi-criteria optimization finds its application. Rescheduling, has been a popular scheduling area since 1990's as evidenced by increasing evolving literature. The main motivation behind this popularity is the recognition of the manufacturing environments that are very often prone to disruptions. Rescheduling problems usually trade-off between the stability and efficiency measures. The efficiency measures are usually producer and/or consumer related. These measures aim to optimize classical performance measures of scheduling, like total flow time, total weighted flow time, total tardiness. The stability measures consider the deviation between the initial and new

schedules. Simultaneous treatment of the efficiency and stability measures, takes one into the area of multi-criteria optimization.

This thesis addresses a rescheduling problem on unrelated parallel machines that has network flow representation and multi-criteria nature. Our criteria are total flow time for efficiency and total reassignment cost for stability. The parallel unrelated machine total flow time and total reassignment cost problems are represented by minimum cost flow networks. The bicriteria problem defined for any function of the total flow time and total reassignment cost is a bicriteria minimum cost network flow problem. Hence any theory added to the bicriteria minimum cost network flow area helps the development of the solution approaches to our rescheduling problem.

In this thesis, we develop some theory for bicriteria minimum cost network flow problem. We apply the theory on our rescheduling problem. We also propose some implicit enumeration based approaches for our rescheduling problem.

The thesis is organized in two main parts: Chapter 2 and Chapter 3. Chapter 2 addresses a bicriteria minimum cost network flow problem. Chapter 3 considers the parallel unrelated machine problem with total flow time and total reassignment cost criteria. Each chapter has its own introduction where the importance of the addressed problem is discussed. In Chapter 4, we conclude and point out some future research directions.

CHAPTER 2

BICRITERIA INTEGER NETWORK FLOW PROBLEM

Network flow problems are well studied and applied models of operations research. The network flow problem with the single objective of minimum total cost is a well recognized problem in the operations research literature (Ahuja et al. (1993). The importance of the single objective problem not only stems from its applicability but also its appearance as a subproblem in many models exploiting network flow structure. Moreover, single objective minimum cost network flow models have very special structure explained by integrality of the extreme points of its feasible polyhedron. This structure, called total unimodularity property, allows use of special linear programming technique, namely network simplex algorithm.

Several well known operations research problems like transshipment, transportation, assignment, shortest path, maximum flow problems are special cases of the minimum cost network flow problem. The minimum cost objective associated with those problems might represent several concerns like minimizing the delivery time, maximizing the safety and reliability, minimizing the deterioration of goods, minimizing the shipping costs, minimizing the resource usages. In the basic model, these concerns are combined in a single total cost objective. But these concerns are usually in conflict. As mentioned in Hamacher (2007), applications with transportation planning faces conflicting criteria like minimization of cost for selected routes, minimization of arrival times at the destination points, minimization of deterioration of goods, maximization of safety, etc. This necessitates the multicriteria formulation of the network flow problem. The solution to the multi criteria problem is a set of efficient, i.e., non-dominated, solutions among which the decision maker is allowed to make a choice according to his/her preferences.

Note that the network flow applications require integer flows, which would be handled automatically, when there is a single objective. Incorporation of second objective, dispels the total unimodularity nature of the network flow models. Hence a need for integer programming based procedures arise.

A bicriteria network flow (BCNF) problem is a special case of multicriteria network flow problem with two criteria and has attracted the attention of many researchers. The majority of the BCNF studies assume continuous flow values. Pulat et al. (1992), Lee and Pulat (1991), Sedeno-Noda and Gonzales-Martin (2000) and Sedeno-Noda and Gonzales-Martin (2003) are the most noteworthy examples. The associated studies formulate the BCNF problem as a parametric programming model, which is solved by network simplex algorithm. The parameter of the models are updated iteratively based on the solution of the previous iteration.

The BCNF problem with integer flow values (BCINF) to find exact set of efficient solutions has been addressed in Lee and Pulat (1993), and Sedeno-Noda and Gonzales-Martin (2001). Sedeno-Noda and Gonzales-Martin (2001) argue that the algorithm by Lee and Pulat (1993) may miss some efficient points and introduce another network simplex based algorithm that implicitly assumes the connectivity of the adjacency graph. Przybylski et al. (2006) show that the adjacency graph is not connected for the BCINF problem, hence settle the incorrectness of Sedeno-Noda and Gonzales-Martin (2001)'s algorithm. Przybylski et al. (2006) also mention that it is not likely to find the exact efficient set for the BCINF problem by simple simplex pivots and interchange arguments. As stated in Hamacher et al. (2007), exact algorithms to find the efficient set is missing in the current literature. But there are few approximation based studies that find a representation of the efficient set, some noteworthy examples are due to Lee and Pulat (1991), Nikolova (1998) and Mustafa and Goh (1998). Lee and Pulat (1991) extend their algorithm for the continuous BCNF problem to find all integer points. Nikolova (1998) studies the problem of generating all supported efficient solutions. Mustafa and Goh (1998) consider bicriteria and tricriteria integer network flow problems and propose approximate solutions by adjusting the non-integer flows via an interactive approach.

For more details on the continuous and integer BCNF problems, the reader is referred to the survey paper of Hamacher et al. (2007) who give a thorough review of optimization and approximation algorithms.

In this study, we propose a two-phase approach to generate the exact efficient set for the BCINF problem. In the first phase, we generate a simplex based approach to generate the efficient solutions of the continuous flow problem. These solutions form the extreme supported efficient set of the integer flow problem. The remaining efficient solutions are found by integer programming based solution procedures that use valid inequalities to ensure the generation of non-extreme supported or unsupported efficient solutions.

The rest of the chapter is organized as follows: In section 2.1, we define our problem, in Section 2.2 we present our solution procedures.

2.1 Problem Formulation

Let $G = (N, A)$ be a network with node set N and arc set A. Let l_{ij} and u_{ij} be the integer non-negative lower and upper bounds on the flow values on each arc $(i, j) \in A$ and b_i be the integer demand (if negative) or supply (if positive) of each node $i \in N$. Let c^1_{ij} and c^2_{ij} be the non-negative integer cost coefficients for the unit flow on arc $(i, j) \in A$, in the objectives $f_1(x)$ and $f_2(x)$ respectively. The decision variable x_{ij} denotes the amount of flow on arc $(i, j) \in A$. The BCINF problem can be formulated as follows:

$$\text{Min} \quad f_1(x) = \sum_{(i,j)\in A} c^1_{ij} x_{ij} \tag{2.1}$$

$$\text{Min} \quad f_2(x) = \sum_{(i,j)\in A} c^2_{ij} x_{ij} \tag{2.2}$$

subject to

$$\sum_{j\in N} x_{ij} - \sum_{j\in N} x_{ji} = b_i \qquad \forall i \in N \tag{2.3}$$

$$l_{ij} \le x_{ij} \le u_{ij} \qquad \forall (i, j) \in A \tag{2.4}$$

$$x_{ij} \text{ is integer} \qquad \forall (i, j) \in A \tag{2.5}$$

Let X represents the set of feasible solutions to the BCINF problem. A feasible solution $x \in X$ is efficient if there does not exist any other feasible solution $x' \in X$ with either $f_1(x') < f_1(x)$ and $f_2(x') \leq f_2(x)$, or $f_1(x') \leq f_1(x)$ and $f_2(x') < f_2(x)$. An efficient solution $x \in X$ is supported if it optimizes any convex combination of $f_1(x)$ and $f_2(x)$ (See Ehrgott and Gandibleux (2000)). In other words, $x \in X$ is a supported efficient solution, if it is one of the optimal solutions to $w_1 f_1(x) + w_2 f_2(x)$ for any w_1, w_2. A supported efficient solution $x \in X$ is extreme supported if it can be found by parameterizing on $w_1 > 0$ and $w_2 > 0$. An extreme supported efficient solution is a boundary efficient solution if it lies at the corners of the $(f_1(x), f_2(x))$, i.e., objective space. A supported efficient solution $x \in X$ is non-extreme supported if lies on the convex combination of two adjacent extreme supported efficient solutions on the objective space. An efficient solution $x \in X$ is unsupported if it is not optimal for any convex combination of $f_1(x)$ and $f_2(x)$. Figure 2.1 illustrates the images of all solutions in the objective space.

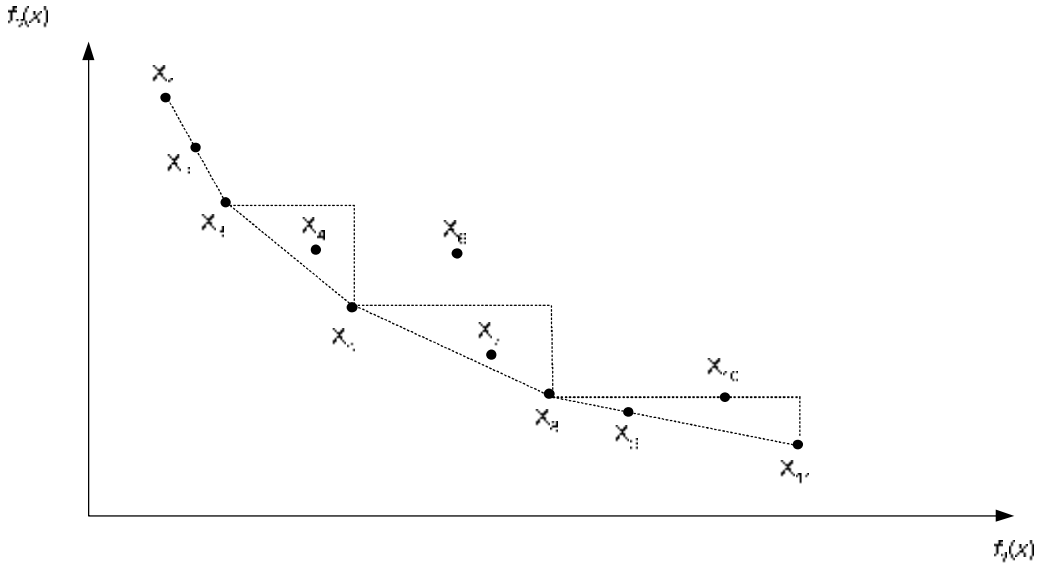

Figure 2.1 Efficient solutions

In Figure 2.1, X_1, X_3, X_5, X_8 and X_{11} are extreme supported efficient points. X_1 and X_{11} are boundary efficient points. X_2 and X_9 are non-extreme supported efficient points as they lie on the convex combination of two adjacent extreme supported points. X_4 and X_7 are unsupported efficient points. X_4 and X_7 cannot optimize any convex combination of $f_1(x)$ and $f_2(x)$, and therefore lie inside the triangle formed by two adjacent extreme supported efficient points. X_6 and X_{10} are inefficient points as they lie on or outside the triangle formed by adjacent extreme supported efficient points.

2.2 Solution Procedures

In this section, we describe two procedures to generate the efficient solution set (Set *E*). Both procedures use the boundary efficient solution set, they iterate starting from one boundary point and terminate when the other boundary point is reached. We describe the generation of the boundary efficient solution set (Set *BE*) in Section 2.2.1.

The first procedure generates all efficient solutions, using an optimal solution of a singly constrained minimum cost network flow problem. The second procedure first generates the set of extreme supported solutions (Set *ESE*), then having known Set *ESE*, it generates the set of non-extreme supported solutions (Set *NSE*) and the set of unsupported efficient solutions (Set *UE*). We present first and second procedures in Sections 2.2.2 and 2.2.3 respectively.

2.2.1 Generation of Boundary Efficient solutions, Set *BE*

Set *BE* can be generated through the solutions of the following hierarchical problem for p=1, s=2 and p=2, s=1, i.e., selecting one objective as primary, and the other as secondary.

(*P*) Min $f_S(x)$

s.t. (2.3), (2.4), and (2.5)

$$f_P(x) = f_P^*(x) \qquad (2.6)$$

where $f_P^*(x)$ is an optimal solution to the single objective, Min $f_P(x)$, network flow problem and can be found in polynomial time using network simplex algorithm.

Now consider the following single objective network flow problem.

(P') Min $f_P(x) + e_S f_S(x)$ where $e_S > 0$ and is sufficiently small.

s.t. (2.3), (2.4), and (2.5)

Corollary 2.1, below, defines a range for e_S that makes (P) and (P') equivalent.

Corollary 2.1. (P) and (P') are equivalent when $e_S < \dfrac{1}{\sum_{(i,j)\in A} (u_{ij} - l_{ij}) c^S{}_{ij}}$.

Proof. e_S should be set small enough so that objective p, should not increase even for the largest possible reduction in objective s. Minimum increase in $f_P(x)$, is 1 unit, since $c^P{}_{ij}$ can only take integer values. Maximum increase in $f_S(x)$, is the difference between $\sum_{(i,j)\in A} u_{ij} c^s{}_{ij}$, i.e., an upper bound on $f_S(x)$, and $\sum_{(i,j)\in A} l_{ij} c^s{}_{ij}$, i.e., a lower bound on $f_S(x)$. Hence e_S ($\sum_{(i,j)\in A} u_{ij} c^s{}_{ij}$ - $\sum_{(i,j)\in A} l_{ij} c^s{}_{ij}$)<1, i.e., $e_S < \dfrac{1}{\sum_{(i,j)\in A} (u_{ij} - l_{ij}) c^S{}_{ij}}$ should hold.

■

Let ($f_1(x^*)$, $f_2(x^*)$) be the optimal solutions to problem (P') with p=1, 2, when e_S is set to $\dfrac{1}{\sum_{(i,j)\in A} (u_{ij} - l_{ij}) c^S{}_{ij} + 1}$. These two solutions form set *BE*. Note that $f_P(x^*)$, with p=1, 2, are lower limits on the $f_1(x)$ and $f_2(x)$ values (f_1^{LB} and f_2^{LB}) of all efficient solutions, respectively. On the other hand, $f_S(x^*)$, with s = 1, 2, are upper limits on the $f_1(x)$ and $f_2(x)$ values (f_1^{UB} and f_2^{UB}) of all efficient solutions, respectively. These limits give an upper bound of Min$\{f_1^{UB} - f_1^{LB}, f_2^{UB} - f_2^{LB}\}+1$ on the number of all efficient solutions. The reader may refer to Steuer (1986), for generation of the boundary efficient solutions for the general bicriteria problem.

2.2.2 Generation of All Efficient Solutions: A Simultaneous Approach

Consider the following singly constrained minimum cost network flow problem

$$(P_k) \quad \text{Min } z = \sum_{(i,j)\in A} c^P{}_{ij} x_{ij} + \frac{1}{\sum_{(i,j)\in A} (u_{ij} - l_{ij}) c^S{}_{ij} + 1} \sum_{(i,j)\in A} c^S{}_{ij} x_{ij}$$

s.t. (2.3), (2.4), and (2.5)

$$\sum_{(i,j)\in A} c^S{}_{ij} x_{ij} \leq k \tag{2.7}$$

(P_k) is NP-Hard as its special case, singly constrained assignment problem, is NP-Hard (See Aggarwal (1985)).

An optimal solution to (P_k) is an efficient solution provided that k is no smaller than f_S^{LB}. (See, Haimes et al. (1971) for the general bicriteria problem)

Procedure 2.1 below generates Set E by varying the value of k between f_S^{UB} and f_S^{LB} and solving (P_k). The procedure can be implemented by taking either of the objectives as primary.

Procedure 2.1

Step 0. Let p=1 or 2.

Find f_S^{UB} and f_S^{LB} and let $k = f_S^{UB} - 1$.

Step 1. If $k \leq f_S^{LB}$ then stop.

Solve (P_k). Let the optimal solution be (f_1^*, f_2^*).

$E = E \cup (f_1^*, f_2^*)$

Step 2. $k = f_S^* - 1$, go to Step 1.

The procedure iterates pseudo-polynomial number of times as there exists pseudo-polynomial number of efficient solutions. Each iteration returns a new efficient solution by solving (P_k), i.e., an NP-Hard problem, in exponential time. Hence the procedure has an exponential-time complexity.

2.2.3 Generation of All Efficient Solutions: A Sequential Approach

We find the efficient set sequentially, by first generating the extreme supported efficient solutions, set *ESE* and then the non-extreme supported efficient solutions, set *NSE* and the unsupported efficient solutions, set *UE*. We next describe the generation of each set.

Generation of Extreme Supported Efficient solutions, Set ESE

We generate Set *ESE* through successive solutions of (P_k) by varying the value of k, in range [f_S^{LB}, f_S^{UB}]. Our procedure to generate Set *ESE*, Procedure 2.2, is similar to Procedure 2.1. It solves the Linear Programming (LP) relaxation of (P_k) and pivots in the slack variable of constraint $f_S(x) \leq k$, whereas Procedure 2.1 solves (P_k) exactly. Below is the stepwise description of the procedure used to generate set *ESE*.

Procedure 2.2

Step 0. Let p=1 or 2

k=f_S^{UB}-1

Step 1. If $k \leq f_S^{LB}$, then stop.

Step 2. Solve the LP relaxation of (P_k).

If the solution is non-integer, perform one simplex iteration by pivoting in the slack variable of constraint $f_S(x) \leq k$ and get an integer solution.

Let the current integer solution be (f_1, f_2).

$ESE = ESE \cup (f_1, f_2)$

$k = f_S(x) - 1$, go to Step 1

Each execution of Step 2 adds a new solution to Set *ESE* by solving the LP in polynomial time. Step 2 iterates pseudo-polynomial number of times as there exists pseudo-polynomial number of solutions in Set *ESE*. Hence the algorithm runs in pseudo-polynomial time.

Theorem 2.1 shows that Procedure 2.2 generates all extreme supported efficient solutions, i.e., Set *ESE*.

Theorem 2.1. Procedure 2.2 generates all extreme supported efficient solutions, i.e., Set *ESE*.

Proof. The correctness of Procedure 2 is based on finding the extreme supported efficient solutions by the LP relaxation of the singly constrained network flow problem. Each point in Set *ESE* corresponds to an extreme point of the corresponding unconstrained network flow problem (see Isermann (1974)). If the LP relaxation of the singly constrained network flow problem gives all integer variables then the resulting solution corresponds to one of the extreme points of the unconstrained network flow problem (see Glover et al. (1978)). Note that the optimal solution of our singly constrained network flow problem, (P_k), is an efficient point. If the LP relaxation of (P_k) provides all integer variables, the resulting solution is extreme supported efficient as it corresponds to one of the extreme points of the network flow problem.

If the LP relaxation gives a non-integral solution then the additional constraint is binding, hence the associated slack variable is zero, i.e., not in the basis (see Glover et al. (1978)). If the additional constraint is not binding, then the associated slack variable is positive, i.e., in the basis. In the latter case, the resulting solution is integral as the remaining constraint set (constraint set of the MCNF problem) is totally unimodular.

When the slack variable of a non-integral solution is pivoted, it takes the maximum value, s_{max}, that does not violate feasibility. The resulting solution is integral (Klingman and Russell (1978)) and solves $(p_{k-s_{\max}})$ problem where the slack variable is in the basis at level zero. The solution is degenerate and corresponds to the same extreme point of the LP relaxed solution of the (P_k) problem. This follows that there cannot exist any other extreme point, hence an extreme supported efficient point, having $f_S(x)$ value between k and k - s_{max}+1. The basis for the $(p_{k-s_{\max}})$ problem is no more feasible for the $(p_{k-s_{\max}-1})$ problem, so one can conclude that the extreme point representing the basis is different for each extreme supported point. Our algorithm catches those extreme points and therefore finds all extreme supported efficient points. ■

The previous approaches to generate set *ESE*, formulate the BCNF problem as a parametric programming model so as to minimize $f_1(x) + l\ f_2(x)$ where $l > 0$. For a specified l, they solve the parametric model by the network simplex method and get an extreme supported efficient solution after which l is updated by some adjustment procedure. The parametric model is resolved for each updated l using network simplex method.

Our algorithm solves the BCNF problem by the network simplex method only once for $k=f_S^{UB}-1$ and finds the remaining *ESE* solutions by the dual-simplex iterations based on the optimal basis of the most recently generated solution. As the associated problems are not solved from scratch, one can expect higher efficiency.

The definition of supported efficient solutions follows that they are optimal for the BCNF problem with the objective of $wf_1(x) + (1-w) f_2(x)$, for some range of w values. We hereafter let $(f_1^r(x), f_2^r(x))$ denote the r^{th} extreme supported efficient solution, S_r, such that $f_1^{r-1}(x) > f_1^r(x) > f_1^{r+1}(x)$ and $f_2^{r-1}(x) < f_2^r(x) < f_2^{r+1}(x)$ and let *#ESE* denote the number of solutions in Set *ESE*.

We let $[w_{r-1}, w_r]$ denote a range for w for which $(f_1^r(x), f_2^r(x))$ is optimal, where $w_0=0$. When $w = w_r$, S_r and S_{r+1} and the solutions that lie on their convex combination are alternate optimal. Hence w_r equates the objective function values of those supported solutions, i.e.,

$$w_r f_1^r(x) + (1-w_r) f_2^r(x) = w_r f_1^{r+1}(x) + (1-w_r) f_2^{r+1}(x)$$

This follows, $w_r = \dfrac{f_2^{r+1}(x) - f_2^r(x)}{f_1^r(x) - f_1^{r+1}(x) + f_2^{r+1}(x) - f_2^r(x)}$ $\quad r = 1,\ldots, \#ESE - 1$

In the next two subsections, we use w_r values to generate the non-extreme supported and unsupported efficient sets.

Generation of Non-extreme Supported Efficient solutions, Set NSE

Consider the following integer programming model.

(P_r^{NE}) Min $w_r f_1(x) + (1- w_r) f_2(x) + e_{NE} f_P(x)$

s.t. (2.3), (2.4), (2.5)

$f_S(x) \leq f_S^r(x) - 1$ (2.8)

(P_r^{NE}) selects the solution with minimum $w_r f_1(x) + (1- w_r) f_2(x)$ value and breaks the ties in favor of $f_P(x)$ value.

Note that when $e_{NE} = 0$, S_{r+1} solves (P_r^{NE}).

Any solution on the convex combination of S_r and S_{r+1}, is non-extreme supported and such a solution has a lower $f_P(x)$ value than that of S_{r+1}. An optimal solution to (P_r^{NE}) is a non-extreme supported efficient point between S_r and S_{r+1} having smallest $f_P(x)$ value, provided that e_{NE} is set according to Corollary 2.2. If such a non-extreme supported efficient solution does not exist, the optimal solution to (P_r^{NE}) is S_{r+1}.

Corollary 2.2. For $e_{NE} < \frac{1}{f_2^{r+1}(x) - f_2^r(x) + f_1^r(x) - f_1^{r+1}(x)} \cdot \frac{1}{f^{UB}_P - f^{LB}_P}$, (P_r^{NE}) minimizes $w_r f_1(x) + (1-w_r) f_2(x)$ and breaks the ties in favor of $f_P(x)$ value.

Proof. e_{NE} should be set small enough so that $z = w_r f_1(x) + (1-w_r) f_2(x)$ value should not be increased even for the largest possible reduction in $f_P(x)$. Let Δz be the difference between optimal value and the objective value of any solution. This difference can be defined mathematically as follows:

$\Delta z = w_r \Delta f_1 + (1-w_r)\Delta f_2$

$=$

$$\frac{f_2^{r+1}(x) - f_2^r(x)}{f_2^{r+1}(x) - f_2^r(x) + f_1^r(x) - f_1^{r+1}(x)} \Delta f_1 + \frac{f_1^r(x) - f_1^{r+1}(x)}{f_2^{r+1}(x) - f_2^r(x) + f_1^r(x) - f_1^{r+1}(x)} \Delta f_2$$

Since two objectives can only take integer values, $f_2^{r+1}(x) - f_2^r(x)$, $f_1^r(x) - f_1^{r+1}(x)$, Δf_1 and Δf_2 are all integers. $\frac{a}{c} - \frac{b}{c} \geq \frac{1}{c}$ holds for any three integers a, b and c such that a-b is positive. This follows $\Delta z > \frac{1}{f_2^{r+1}(x) - f_2^r(x) + f_1^r(x) - f_1^{r+1}(x)}$ such that $\Delta z > 0$.

The maximum increase in $f_P(x)$ is $f_P^{UB} - f_P^{LB}$.

Therefore, $(f_P^{UB} - f_P^{LB})\ e_{NE} < \frac{1}{f_2^{r+1}(x) - f_2^r(x) + f_1^r(x) - f_1^{r+1}(x)}$,

equivalently, $e_{NE} < \frac{1}{f_2^{r+1}(x) - f_2^r(x) + f_1^r(x) - f_1^{r+1}(x)} \cdot \frac{1}{f^{UB}_P - f^{LB}_P}$ should hold.

■

Before solving (P_r^{NE}) with the hope of finding a non-extreme efficient solution between S_r and S_{r+1}, one may check for the conditions for the non-existence of those solutions. One such condition is stated in Corollary 2.3.

Corollary 2.3. There is no non-extreme supported efficient solution that lies between two extreme supported efficient solutions S_r and S_{r+1} if there is no integer $(f_1(x), f_2(x))$ point on the line connecting $(f_1^r(x), f_2^r(x))$ and $(f_1^{r+1}(x), f_2^{r+1}(x))$.

Proof. The non-extreme supported efficient solutions lie on the line connecting two adjacent extreme support efficient solutions. The $f_1(x)$ and $f_2(x)$ values of all solutions are integers as our parameters are integers. If there are no integer values $(f_1(x), f_2(x))$ on the line connecting $(f_1^r(x), f_2^r(x))$ and $(f_1^{r+1}(x), f_2^{r+1}(x))$, there cannot exist any non-extreme supported efficient solutions between S_r and S_{r+1}.

■

Corollary 2.3 implies that one may skip the region in the objective space defined by S_r and S_{r+1}, if there is no integer point on their convex combination. Moreover the result of the corollary can be used whenever a non-extreme supported efficient solution, say $\hat{S}_r$, is reached and there is no integer point on the convex combination of $\hat{S}_r$ and $S_{r+!}$. In such a case one may again proceed to the region defined by S_{r+1} and S_{r+2}.

Note that, there does not exist an integer solution between S_r and S_{r+1}, if for each integer value of $f_1(x)$ between $f_1^r(x)$ and $f_1^{r+1}(x)$, the corresponding $f_2(x)$ value is continuous. The $f_2(x)$ value can be found using the following equation.

$$f_2(x) = f_2^r(x) + \frac{(f_1^r(x) - f_1(x)).(f_2^{r+1}(x) - f_2^r(x))}{f_1^r(x) - f_1^{r+1}(x)} \text{ where } f_1(x) \in [f_1^{r+1}(x)+1, f_1^r(x)-1].$$

This check can be made for each integer point $f_2(x) \in [f_2^r(x)+1, f_2^{r+1}(x)-1]$ and the corresponding $f_1(x)$ values.

We now provide the stepwise description of the algorithm that solves the (P_r^{NE}) problem for each (S_r and S_{r+1}) pair.

Procedure 2.3

Step 0. Let p = 1 or 2 and r = 1.

Step 1. $r = r+1$, if $r = \#\ ESE$ then stop.

$k = f_S^r(x) - 1$

Step 2. If there is no integer point on the line connecting ($f_1^r(x)$, $f_2^r(x)$) and ($f_1^{r+1}(x)$, $f_2^{r+1}(x)$)

then go to Step 1

Step 3. Solve (P_r^{NE}) with $f_S(x) \leq k$.

Step 4. Let $\hat{S}_r$ be the solution with ($\hat{f}_1^r(x)$, $\hat{f}_2^r(x)$)

If $\hat{S}_r = S_{r+1}$ then go to Step 1.

$NSE = NSE \cup \hat{S}_r$

If there is no integer point on the line connecting ($f_1^r(x)$, $f_2^r(x)$) and

($\hat{f}_1^r(x)$, $\hat{f}_2^r(x)$) then go to Step 1.

$k = \hat{f}_S^r(x) - 1$, go to Step 3

Generation of Unsupported Efficient solutions, Set UE

Consider the following inequality

$$w_r f_1(x) + (1\text{-}w_r) f_2(x) > z_r \qquad r = 1,..., \#ESE\text{ -}1 \qquad (2.9)$$

where $z_r = w_r f_1^r(x) + (1 - w_r) f_2^r(x)$

The unsupported efficient solutions satisfy constraint set (2.9), for all r as they do not optimize any convex combination of $f_1(x)$ and $f_2(x)$. Corollary 2.4 shows that (2.9) is not satisfied by any supported efficient solution, thereby providing a valid cut for any model that aims to find an unsupported efficient solution.

Corollary 2.4. The constraint $w_r f_1(x) + (1\text{-}w_r) f_2(x) > z_r$ eliminates all supported efficient solutions.

Proof. Two adjacent extreme supported efficient solutions, S_r and S_{r+1}, and any solution on their convex combination do not satisfy (2.9), as they minimize $w_r f_1(x) + (1 - w_r) f_2(x)$ with an objective function value of z_r, i.e., $z_r = w_r f_1(x) + (1 - w_r) f_2(x)$ for S_r and S_{r+1},. This follows, $w_r f_1(x) + (1\text{-}w_r) f_2(x) > z_r$ eliminates S_r and S_{r+1}, and any non-extreme efficient solution on their convex combination. Therefore, constraint set (2.9) defined over all adjacent extreme supported solution pairs, eliminates all supported efficient solutions.

■

From the standpoint of using a mathematical programming software, we convert (2.9) into 'greater than or equal to' type constraint. Recall from the proof of Corollary 2.2 that the minimum increase in z value is $e_r = \dfrac{1}{f_1^{r+1}(x) - f_1^r(x) + f_2^r(x) - f_2^{r+1}(x)}$ and hence constraint set (2.9) is equivalent to the following constraint set.

$$w_r f_1(x) + (1\text{-}w_r) f_2(x) \geq z_r + \frac{1}{f_2^{r+1}(x) - f_2^r(x) + f_1^r(x) - f_1^{r+1}(x)} \qquad (2.10)$$

$$r = 1,..., \#ESE\text{ -}1$$

Constraint set (2.10) eliminates the supported efficient solutions, but not the inefficient solutions. To eliminate the inefficient solutions, we use the efficiency definition and add either $f_1(x) \leq f_1^r(x) - 1$ or $f_2(x) \leq f_2^r(x) - 1$, constraints for each r:

One can linearize this either/or type relation via a binary variable, Y_r, as follows:

$$f_1(x) \leq f_1^r(x) - 1 + (f_1^{UB}(x) - f_1^{LB}(x))(1 - Y_r)$$

$$f_2(x) \leq f_2^r(x) - 1 + (f_2^{UB}(x) - f_2^{LB}(x))Y_r$$

The model to find an unsupported efficient solution can then be written as,

(P_k^{UE}) Min $f_P(x) + e_S f_S(x)$

s.t. (2.3), (2.4), (2.5)

$$f_S(x) \leq k$$

$$w_r f_1(x) + (1-w_r) f_2(x) \geq z_r + e_r \qquad \forall r \quad \text{s.t. } f_S^r(x) \leq k$$

$$f_1(x) \leq f_1^r(x) - 1 + (f_1^{UB}(x) - f_1^{LB}(x))(1 - Y_r) \qquad \forall r \quad \text{s.t. } f_S^r(x) \leq k$$

$$f_2(x) \leq f_2^r(x) - 1 + (f_2^{UB}(x) - f_2^{LB}(x))Y_r \qquad \forall r \quad \text{s.t. } f_S^r(x) \leq k$$

$$Y_r = 0 \text{ or } 1$$

For a given k in range $(f_S^{LB}(x), f_S^{UB}(x))$, (P_k^{UE}) either returns an unsupported efficient solution or concludes that no unsupported efficient solution having $f_S(x)$ value no bigger than k exists. Below is the formal description of the algorithm that uses (P_k^{UE}) to generate all unsupported efficient solutions.

Procedure 2.4

Step 0. Let $p = 1$ or 2, $k = f_S^{UB} - 1$

Step 1. If $k = f_S^{LB}$, then stop.

Solve (P_k^{UE}).

If the solution is infeasible, all unsupported solutions are generated, stop.

Step 2. Let $(f_1^u(x), f_2^u(x))$ be the solution.

UE = UE $\cup$ $(f_1^u(x), f_2^u(x))$

$k = f_S(x) - 1$, go to Step 1

Procedure 2.4 iterates pseudo-polynomial number of times, as each execution of Step 2 returns a new unsupported efficient solution and there is pseudo-polynomial number of unsupported efficient solutions. Step 2 solves an integer program (P_k^{UE}), in exponential time. So, the procedure runs in exponential time.

In Chapter 3, we deal with a rescheduling problem that is bicriteria in nature and has a network flow representation. We, thus, apply the theory derived in this section to the bicriteria minimum cost network flow problem to deal with our rescheduling problem that trade-offs between total flow time and total reassignment cost objectives.

CHAPTER 3

RESCHEDULING PROBLEM

Majority of the scheduling literature considers a manufacturing environment with no disruptions. However in manufacturing practice, the environment is very often subject to disruptions that makes the initial scheduling plan inefficient or even infeasible and necessitates rescheduling. The common disruptions are machine breakdowns, hence subsequent repairs, new order arrivals, order cancellations, changes in order specifications like priorities, release times, and due dates, and shortages of resources like materials, labor, tools and equipments.

We consider a parallel machine environment where the machines are subject to disruptions and where the jobs are initially scheduled so as to minimize total flow time, i.e., total time the jobs spent in the system. Flow time gives a direct indication of the work-in-process inventory levels, hence its minimization is an important concern of many manufacturers.

We assume the customer promises are given and the resource allocations are made according to the initial minimum flow time schedule. During the execution of the initial plan, a disruption blocks the machines for a specified length of time. Thereafter, the manufacturer still aims to minimize the total flow time of the jobs that have not yet started, considering the disruption effect. However, the new minimum flow time schedule may deviate from the initial schedule, in terms of machine allocations. A deviation may cause disturbances, in particular, when the machine setups and resource allocations are made according to the initial allocations.

We aim to consider the trade-off between the efficiency of the new schedule, measured by the total flow time and the stability measured by the difference between the initial and new machine allocations. As a stability measure, we use the total reassignment cost. The jobs receive costs, i.e., penalties, according to the machines they are assigned in the new schedule. The reassignment cost of job *i* on machine *j* is zero, if job *i* is assigned to machine *j* in the initial schedule. We consider the unrelated parallel machine environment where the processing time of a job is dependent on the machine it is assigned on.

The rest of the Chapter is organized as follows. In Section 3.1, we review the rescheduling literature. In Section 3.2, we introduce the basic definitions, notation, and define our problems. In Section 3.3, we present the optimization algorithms for each of our problems. The results of our experiments are presented in Section 3.4.

3.1 Literature Review

The rescheduling studies are of relatively recent origin. Almost all related work are published in 1990's and 2000's. Vieira et al.(2003) classify rescheduling strategies as dynamic with no schedule generation or predictive-reactive with schedule generation and update. Dynamic strategies can be dispatching rules or control-theoretic approaches. Rescheduling can be done periodic, event-driven or hybrid in predictive-reactive strategies. Schedule generation and schedule repair are the two methods used for rescheduling. Schedules can be robustly generated by taking disruptions into account. As a repair methodology right-shift scheduling, partial rescheduling or complete regeneration can be used. Raheja and Subramaniam (2002) review rescheduling in a job shop environment and identify the methods used in rescheduling. Right shift scheduling is the simplest strategy that recovers disruption by shifting all the jobs towards the right in the time horizon without changing the initial sequence. Affected operations rescheduling is a partial scheduling strategy where only the jobs that are affected from the disruptions are rescheduled.

Aytug et al. (2005) review the literature on executing production schedules in the presence of disruptions. A four dimensional taxonomy is introduced. The taxonomy is; Cause – object, state; Context – free or sensitive; Impact – time, material, quality, dependency, context; Inclusion; predictive and/or reactive. A number of directions for future work are suggested on; problem formulation, estimation of reconfiguration costs, using available information on the nature of disruptions and integrating with structural control. Henning and Cerda (2000) present a knowledge-based framework, based on the object oriented technology, for building scheduling systems aimed at solving real-world problems. The paper points out the most relevant aspects of the proposed framework architecture that support both predictive and reactive scheduling.

Hall and Potts (2005) consider inserting new jobs in a schedule without excessive disruption of the old jobs. New jobs must be inserted into the current schedule while preserving the original assignments as much as possible. They consider maximum lateness and total flow time as efficiency measures and the total sequence deviation and total completion time deviation between the initial and new schedules as stability measures. They utilize two different models, in the first model they minimize scheduling cost under a limit on the disruption amount and in the second model they simultaneously consider the two criteria in the objective function. They provide either efficient algorithms or show that such algorithms are unlikely to exist. Unal et al. (1997) consider the problem of rescheduling a facility modeled as a single machine in the face of newly arrived jobs with part-type dependent setup times. Their aim is to insert the new jobs in the schedule so as to minimize the total weighted completion time or the maximum completion time of the new jobs. They provide a polynomial-time algorithm for the maximum completion time problem. Daniels and Kouvelis (1995) formalize the robust scheduling concept for scheduling situations with uncertain or variable processing times. They consider a single-machine environment and minimize the total flow time. O'Donovan et al. (1999) applies predictable scheduling approach to minimize total tardiness on a single machine with stochastic machine failures. Their procedure considers the case where the processing times are affected by machine breakdowns, and provides specialized rescheduling heuristics.

Bean et al. (1991) consider the rescheduling operations with release dates and multiple resources. They specify some optimality conditions and a solution approach. Their approach, called as Match-up Scheduling, follows the initial schedule until a disruption occurs. After a disruption, part of the schedule is reconstructed to match up with the initial schedule at some future time. Leung and Pinedo (2004) consider parallel machine scheduling, where the machines are identical and subject to repair and breakdown. Three objectives namely, the total completion time, the makespan, and the maximum lateness are considered. They analyze the case where the jobs have deadlines and are subject to precedence constraints.

Mason et al. (2004) work on rescheduling strategies for minimizing total weighted tardiness in complex job shops of semiconductor manufacturing environment. Three rescheduling strategies, namely right shift scheduling, fixed sequence rescheduling and complete rescheduling are examined, to investigate the efficiency of each strategy on the on-time delivery performance measured by the total weighted tardiness. Aktürk and Görgülü (1999) propose a rescheduling strategy and match-up point determination procedure to increase both the schedule quality and stability on modified flow shops (MFS) in which the machines are physically arranged in cellular form. Abumaizar and Svestka (1997) present an affected operations rescheduling algorithm in a job shop and compare it with complete rescheduling and Right-shift Scheduling strategies. Their results demonstrate the superiority of the Affected operations algorithm over other rescheduling methods. O'Kane (2000) describes research on the development of an intelligent simulation environment. The environment is used to analyze reactive scheduling scenarios in a specific flexible manufacturing systems (FMS) configuration. Various intelligent systems and concepts are developed and implemented to provide decision making and control across a FMS schedule lifetime. Sabuncuoğlu and Bayız (2000) study a reactive scheduling problem in classical job shop environments, and use mean tardiness and makespan as performance measures. Kutanoğlu and Sabuncuoğlu (2001) study reactive scheduling in dynamic job shops, where the machines are prone to unexpected failures. Their strategy is to reroute the jobs if one the machines on the original route fails.

Rangsaritratsamee et al.(2004) propose a rescheduling methodology whose objective contains both efficiency and stability measures. Schedules are generated at each rescheduling point using a genetic local search algorithm that allows efficiency and stability to be balanced in a way that is appropriate for each situation. Mehta and Uzsoy (1998) present a predictable scheduling approach which can absorb disruptions without affecting planned external activities, while maintaining high shop performance. The procedure inserts additional idle time into the schedule to absorb the impacts of breakdowns. The amount and location of the additional idle time are determined from the breakdown and repair distributions as well as the structure of the predictive schedule. The effects of the disruptions on the planned support activities are measured by the deviations of the job completion times between the realized and predictive schedule. This approach is applied to maximum lateness (L_{max}) problem in a job shop environment with random machine breakdowns.

Wu, Storer and Chang (1993) develop rescheduling heuristics for single machine environments. They utilize makespan as the efficiency measure and start time and sequence deviation of the initial and new schedules as their stability measure. Li and Shaw (1996) consider dynamic scheduling on job shop environment. They utilized a simulation model to evaluate their proposed heuristic against classical heuristics.

Alagöz and Azizoğlu (2003) and Azizoğlu and Alagöz (2005) address the trade-off between the total flow time and the number of reassigned jobs. Azizoğlu and Alagöz (2005) develop a polynomial time algorithm to generate all non-dominated solutions, whereas Alagöz and Azizoğlu (2003) consider eligibility constraints and propose approximation and optimization algorithms. Curry and Peters (2005) consider total reassignment penalty as a stability measure and total tardiness as an efficiency measure. They propose a simulation study to test the efficiencies of some heuristic procedures and rescheduling strategies.

Church and Uzsoy (1992) consider single machine and parallel identical machine environments to minimize the maximum lateness and the number of times rescheduling is done. They provide a simulation study to test the efficiencies of some rescheduling strategies like periodic, event-driven and continuous rescheduling.

In our study, we consider a rescheduling problem on unrelated parallel machine environments that address the trade-off between the total flow time and the total reassignment cost. Our model is a generalization of the identical parallel machine models in Alagöz and Azizoğlu (2003), and Azizoğlu and Alagöz (2005) that consider the trade-off between the total flow time and the number of reassigned jobs. Our aim is to contribute to the rescheduling literature by proposing a solution methodology for a bicriteria problem on unrelated parallel machine environments.

3.2 Problem Definition

We consider a manufacturing environment with m unrelated parallel machines. We assume all jobs are available at time zero, and each should be assigned to one of the machines, and processed without interruption. Each job i is characterized by an integer processing time p_{ij} time units on machine j.

We assume the initial schedule is known. There is a disruption of D time units on one of the machines, say machine DM, after executing the initial schedule for DT time units. The job that is being processed on DM, and the jobs that start on or after DT on other machines are to be rescheduled at time DT. We assume there are n such jobs. Once we take the reference starting point from time zero to DT, our rescheduling problem reduces to scheduling n jobs, available at time zero, on m unrelated parallel machines where machine j becomes available at time a_j. Accordingly, $a_{DM} = D$ and a_j is the completion time of the job processed at time DT on non-disrupted machine j. Note that, multiple simultaneous disruptions can also be handled by letting $a_j = D_j$ where D_j is the time at which the disruption on machine j, is recovered. We assume D, DT, and a_j are all integers. Figure 3.1 illustrates a rescheduling environment where $DM=M1$.

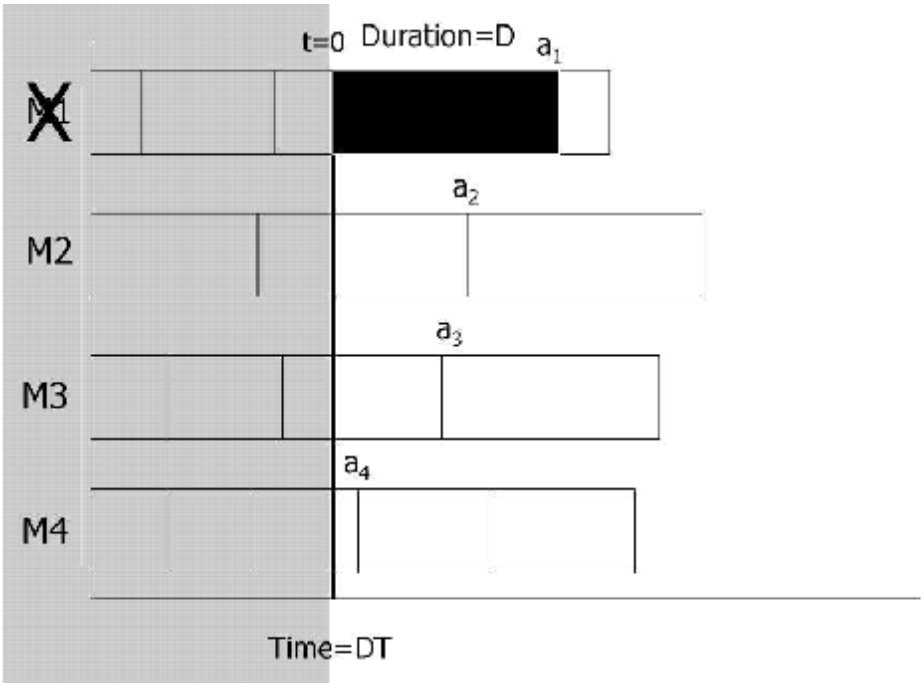

Figure 3.1 Example rescheduling environment

The scheduling cost, that defines our efficiency measure, is the total flow time, *F*. Total flow time is the total time the jobs spent in the system and therefore is the direct indication of total work-in-process inventory levels. As we assume all zero ready times, the total flow time and total completion time are equivalent measures. If we let C_i denote the completion time of job i in the new schedule, total flow time, $F = \sum_{i=1}^{n} C_i$. The disruption cost that defines our stability measure is the total reassignment cost of jobs that are reassigned to different machines between initial and new schedules, shortly reassigned jobs. We let

$$R_i = \begin{cases} 1 & \text{if job } i \text{ is reassigned} \\ 0 & \text{otherwise} \end{cases}$$

The total number of reassigned jobs, a special case of total reassignment cost with unique cost terms, is $\sum_{i=1}^{n} R_i$. Total reassignment cost is defined by letting

wr_{ij} = integer cost (penalty) of assigning job i to machine j

We can interpret wr_{ij} as the additional cost incurred due to the reassignment of job i to machine j. Such a cost might be incurred due to the additional set-up, adjustment, tooling, material/labor shifting done.

The total reassignment cost, WRJ, is $\sum_i \sum_j wr_{ij} R_i$.

A schedule S is said to be efficient with respect to F and WRJ if there exists no schedule S' with $F(S') \leq F(S)$ and $WRJ(S') \leq WRJ(S)$ with at least one strict inequality. An efficient schedule $s \in S$ is supported if it optimizes any weighted sum of WRJ and F. In other words, $s \in S$ is a supported efficient solution, if it is one of the optimal solutions to $w_1\ WRJ + w_2\ F$ for any non-negative w_1, w_2. A supported efficient schedule $s \in S$ is extreme supported efficient if it can be found by parameterizing on w_1 and w_2. A supported efficient schedule $s \in S$ is nonextreme supported efficient if lies at the convex combination of two adjacent extreme supported efficient schedules on the (WRJ, F). An efficient schedule $s \in S$ is unsupported if it is not optimal for any weighted sum of WRJ and F.

The standard classification schemes for scheduling problems use three-field representation $a \mid b \mid g$ where α is the machine environment, β is the constraints or special characteristics of the problem and γ is the objective function (see Lawler et al. (1989)). We consider unrelated parallel machines and hence set α = R, when the parallel machines are identical, i.e., $p_{i\,j} = p_i$ for all i and j, we set α = P. We have initial machine available times denoted by a_j in β field. Moreover, we use the following constraints

β = $F = F^*$: total flow time should be kept at its minimum value

β = $WRJ = WRJ^*$: total reassignment cost should be kept at its minimum value

β = $F \leq k$: total flow time can be at most k

β = $WRJ \leq k$: total reassignment cost can be at most k

β = a_j : the machines have initial available times

We consider *F, WRJ* as efficiency and stability measures, hence we have,

γ = *F, WRJ* : generating set of efficient schedules with respect to *F* and *WRJ*

γ = *f* (*F, WRJ*) : finding an optimal schedule for a specified function of *F* and *WRJ*

3.3 Solution Procedures

In this section, we provide solution procedures to our problems that are described in detail in the following sections.

3.3.1 The $R|a_j|F$ problem

Kaspi and Montreuil (1988) show that the $P|a_j|F$ problem can be solved in polynomial time by assigning the shortest available job to the earliest available machine. Lee and Liman (1992) and Mosheiov (1994) study the more general case of the $P|a_j|F$ problem where the machines are unavailable at arbitrary, but not necessarily initial, times.

A special case of the $R|a_j|F$ problem where p_{ij}= p_i or ∞ for all *i* and *j*, is formulated as a network flow problem in Alagöz and Azizoğlu (2003). We now extend this network formulation to the arbitrary p_{ij} case.

Our decision variable is defined as

$$X_{ikj}: \begin{cases} 1 & \text{if job } i \text{ is scheduled } k^{th} \text{ position from last on machine } j. \\ 0 & \text{otherwise} \end{cases}$$

The objective function requires the minimization of the total flow time values, i.e.,

$$\text{Min} \sum_{i=1}^{n}\sum_{k=1}^{n}\sum_{j=1}^{m}(kp_{ij}+a_j)X_{ikj} \qquad (3.1)$$

kp_{ij} is the contribution of the processing time of job *i* to the total flow time if it is sequenced at k^{th} position from last on machine *j* and a_j is the start time of the first job on machine *j*.

The constraint sets are as stated below:

$$\sum_{k=1}^{n}\sum_{j=1}^{m} X_{ikj} = 1 \qquad \forall\, i \tag{3.2}$$

$$\sum_{i=1}^{n} X_{ikj} \le 1 \qquad \forall\, j, k \tag{3.3}$$

$$X_{ikj} \in \{0,1\} \qquad \forall\, i, j, k \tag{3.4}$$

Constraint sets (3.2) and (3.3) ensure that each job is scheduled exactly once and each position of each machine is occupied by at most one job. Constraint set (3.4) requires that the jobs cannot be preempted or splitted. Due to the total unimodularity of the constraint set of the network flow models (see Papadimitriou and Steiglitz (1982)), the Linear Programming (LP) relaxation of the model provides all integer solutions. Therefore constraint set (3.4) can be replaced by

$$0 \le X_{ikj} \le 1 \qquad \forall\, i, j, k \tag{3.5}$$

3.3.2 The $R|a_j|WRJ$ problem

WRJ can be forced to its lower bound of zero by applying the right-shift strategy to the initial schedule. The right-shift strategy shifts all jobs on *DM*, *D* time units to the right, while keeping other job assignments the same. The *F* value that solve $R|a_j|WRJ$ problem, i.e., *F* value of the right-shift schedule, gives an upper bound on the *F* values of all efficient schedules.

3.3.3 The $R|a_j, F = F^*|WRJ$ problem

Note that the *F* value that solves the $R|a_j|F$ problem gives a lower bound on the *F* values of all efficient solutions. However the resulting schedule may not be efficient as there may exist alternate optimal schedules to the $R|a_j|F$ problem having smaller *WRJ* values. Among the alternate optimal schedules to the total flow time problem, the one that has the smallest *WRJ* value, hence the efficient schedule requires an exact solution of the $R|a_j, F = F^*|WRJ$ problem. In place of

incorporating $F = F^*$ to our network flow model, we can modify our objective function as $F + e_{WRJ} WRJ$, for a sufficiently small value of $e_{WRJ} > 0$. Theorem 3.1 states this result formally and defines a range for e_{WRJ}.

Theorem 3.1. The $R|a_j, F = F^*|WRJ$ and $R|a_j|F + e_{WRJ}WRJ$ problems are equivalent when $e_{WRJ} < \dfrac{1}{\sum_{i=1}^{n} Max_j\{wr_{ij}\}}$.

Proof. e_{WRJ} should be set small enough so that the total flow time value should not increase even for the largest possible reduction in the total reassignment cost. The minimum increase in the total flow time is one unit due to the integrality of the processing times. The maximum increase in the total reassignment cost is $\sum_{i=1}^{n} Max_j\{wr_{ij}\}$.

Hence $e_{WRJ} \sum_{i=1}^{n} Max_j\{wr_{ij}\} < 1$, i.e., $e_{WRJ} < \dfrac{1}{\sum_{i=1}^{n} Max_j\{wr_{ij}\}}$ should hold.

■

In our experiments, we use $e_{WRJ} = \dfrac{1}{\sum_{i=1}^{n} Max_j\{wr_{ij}\} + 1}$ for the $R|a_j|F + e_{WRJ}WRJ$ problem. Note that, this result is presented in Chapter 2 for BCINF problem, we derived the e for our rescheduling problem using the $\sum_{i=1}^{n} Max_j\{wr_{ij}\}$ as the upper bound, and zero value as the lower bound (See Corollary 2.1).

3.3.4 The $R|a_j, WRJ = WRJ^*|F$ problem

The right-shift schedule solves the $R|a_j|WRJ$ problem, as it produces an *WRJ* value of zero. However, wr_{ij} can be zero, even when a job is reassigned In such a case, there can be a schedule, other than right-shift, having a zero *WRJ* value and smaller *F* value than that of the right-shift schedule. The efficient schedule having smallest *F* value, among the ones having zero *WRJ* value, can be found by solving the $R|a_j, WRJ = WRJ^*|F$ problem. Instead of treating *WRJ*=*WRJ** constraint, we can modify the objective function as $WRJ + e_F F$ for a sufficiently small value of e_F. Theorem 3.2 states this result formally and defines a range for e_F.

Theorem 3.2. The $R|a_j, WRJ = WRJ^*|F$ and $R|a_j|WRJ + e_F F$ problems are equivalent when $e_F < \frac{1}{F_{UB} - F_{LB}}$, where F_{UB} and F_{LB} are the F values that solves the $R|a_j|WRJ$ and $R|a_j|F$ problems respectively.

Proof. e_F should be set small enough so that the total reassignment cost should not increase even for the largest possible reduction in total flow time. The minimum increase in the total reassignment cost is one unit due to integrality of wr_{ij} values. The maximum increase in the total flow time is F_{UB} - F_{LB} units.

Hence $e_F [F_{UB} - F_{LB}] < 1$, i.e, $e_F < \frac{1}{F_{UB} - F_{LB}}$ should hold.

■

In our experiments we use $e_F = \frac{1}{F_{UB} - F_{LB} + 1}$. Note that, this result is previously presented in Chapter 2 for the BCINF problem (See Corollary 2.1).

3.3.5. The constrained optimization problems

The $R|a_j, WRJ \le s|F + e_{WRJ}WRJ$ and $R|a_j, F \le s|WRJ + e_F F$ are singly-constrained assignment problems. The additional constraints to the assignment model, $WRJ \le s$, and $F \le s$ are expressed as follows:

$$\sum_{i,j,k} wr_{ij} X_{ikj} \le s \tag{3.6}$$

$$\sum_{i,j,k} (a_j + kp_{ij}) X_{ikj} \le s \tag{3.7}$$

For arbitrary coefficients, the singly-constrained assignment problem is NP-Hard so are the $R|a_j, WRJ \le s|F + e_{WRJ}WRJ$ and $R|a_j, F \le s|WRJ + e_F F$ problems.

3.3.6 Generation of all extreme supported efficient schedules

We generate the extreme supported efficient solutions using an LP based procedure 2.2. Procedure 3.1 applies the steps of Procedure 2.2, to our rescheduling problem.

Procedure 3.1 Generation of extreme supported efficient solutions

Step 0. Let $s = WRJ^{UB} - 1$

Step 1. If $s \le WRJ^{LB} + 1$, then STOP

Step 2. Solve the LP relaxation of (P_s)

$$(P_s) \quad \text{Min} \sum_{i=1}^{n}\sum_{k=1}^{n}\sum_{j=1}^{m}(kp_{ij} + a_j)X_{ikj} + \frac{1}{\sum_{i=1}^{n} Max_j\{wr_{ij}\} + 1} \sum_{i,j,k} wr_{ij} X_{ikj}$$

$$\text{s.t} \quad \sum_{k=1}^{n}\sum_{j=1}^{m} X_{ikj} = 1 \qquad \forall\, i$$

$$\sum_{i=1}^{n} X_{ikj} \le 1 \qquad \forall\, j, k$$

$$X_{ikj} \in \{0,1\} \qquad \forall\, i, j, k$$

$$\sum_{i,j,k} wr_{ij} X_{ikj} \le s$$

If all decision variables are not integer,

then perform a single simplex iteration by pivoting in the slack variable of $\sum_{i,j,k} wr_{ij} X_{ikj} \leq s$

Let the current solution be (F*, *WRJ**).

ESE = *ESE* $\cup$ (F*, *WRJ**)

s = *WRJ** – 1, Go to Step 1

The following figure illustrates the progress of Procedure 3.1.

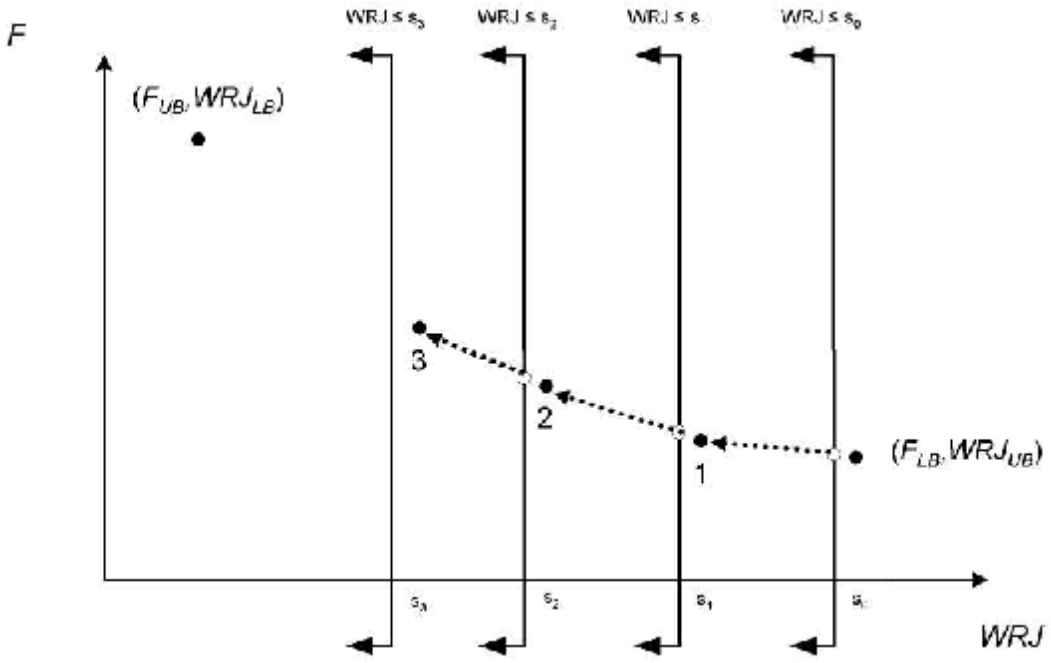

Figure 3.2 Progress of Procedure 3.1

The steps followed by Procedure 3.1 are as follows:

Step 0. Set $s = WRJ_{UB} - 1 = s_0$

Step 2. Solve the LP Relaxation of P_{s_0}

Pivot in the slack variable, identify extreme supported point 1.

Let the current solution be (F_1^*, WRJ_1^*).

$ESE = ESE \cup (F_1^*, WRJ_1^*)$

$s = WRJ_1^* - 1 = s_1$

Step 2. Solve the LP Relaxation of P_{s_1}

Pivot in the slack variable, identify extreme supported point 2.

Let the current solution be (F_2^*, WRJ_2^*).

$ESE = ESE \cup (F_2^*, WRJ_2^*)$

$s = WRJ_2^* - 1 = s_2$

Step 2. Solve the LP Relaxation of P_{s_2}

Pivot in the slack variable, identify extreme supported point 3.

Let the current solution be (F_3^*, WRJ_3^*).

$ESE = ESE \cup (F_3^*, WRJ_3^*)$

$s = WRJ_3^* - 1 = s_3$

The procedure continues to iterate in a similar manner, until it hits to the other boundary point, namely (F_{UB}, WRJ_{LB}).

3.3.7 Generation of all efficient schedules

We develop two approaches to generate the efficient set. First approach, we call Integer Programming Based (IPB) approach, solves singly constrained optimization problems, successively. Second approach is a Branch and Bound method that makes implicit enumeration of the efficient schedules.

Integer Programming Based (IPB) Approach

We generate the efficient schedules through the Procedure 3.2 below that varies the value of s between the WRJ_{LB} and WRJ_{UB}. Note that this procedure is Procedure 2.1 from Chapter 2, modified for our rescheduling problem.

Procedure 3.2. Classical Approach: Finding All Efficient Schedules

Step 0. Solve the $R|a_j|F+\dfrac{1}{\sum_{i=1}^{n} Max_j\{wr_{ij}\}+1}WRJ$ problem and form a right-shift schedule.

WRJ_{LB} = WRJ value of the right-shift schedule, i.e., zero

WRJ_{UB} = WRJ value that solves the

$R|a_j|F+\dfrac{1}{\sum_{i=1}^{n} Max_j\{wr_{ij}\}+1}WRJ$ problem

Let $s = WRJ_{UB} - 1$

Step 1. Solve the $R|a_j, WRJ \le s|F+\dfrac{1}{\sum_{i=1}^{n} Max_j\{wr_{ij}\}+1}WRJ$ problem

Let (F^*, WRJ^*) be the solution

$E = E \cup (F^*, WRJ^*)$

Step 2. If $WRJ^* = WRJ_{LB}$ then STOP

$s = WRJ^* - 1$

Go to Step 1

The following figure illustrates the progress of Procedure 3.2.

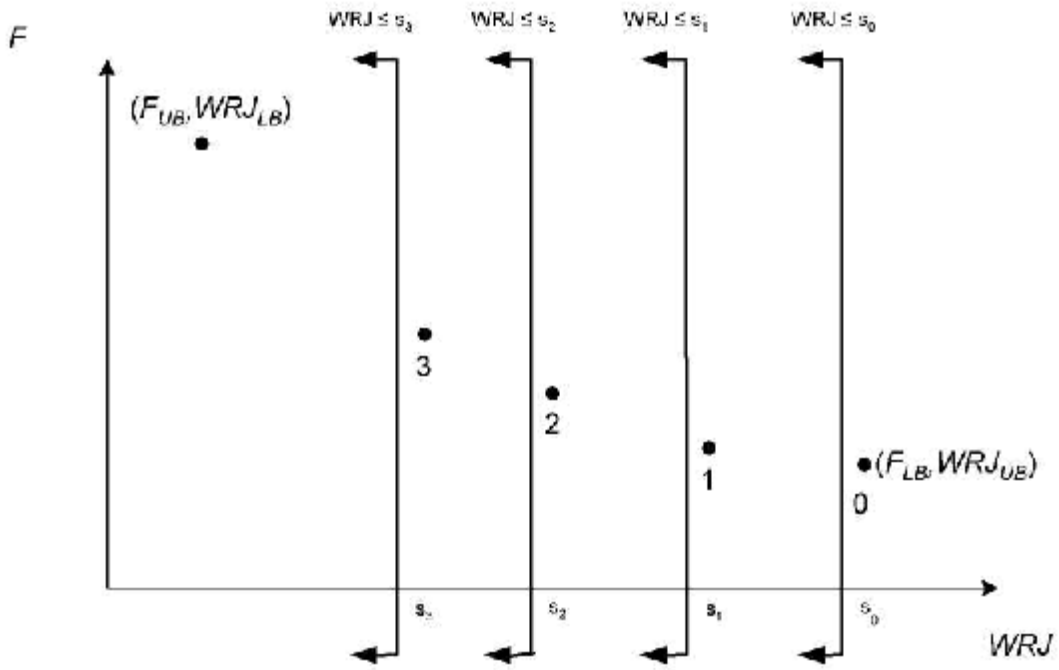

Figure 3.3 Progress of Procedure 3.2

The steps followed by Procedure 3.1 are as follows:

Step 0. Form the right shift schedule, and identify (F_{UB}, WRJ_{LB}).

Solve $R|a_j|F+e_{WRJ}WRJ$ problem and identify (F_{LB}, WRJ_{UB}).

$s = WRJ_{UB}-1=s_0$

Step 1. Solve the $R|a_j, WRJ \le s|F+\dfrac{1}{\sum_{i=1}^{n} Max_j\{wr_{ij}\}+1}WRJ$ problem

Let (F_1^*, WRJ_1^*) be the solution

Point 1 in Figure 3.3 is the corresponding efficient point.

$E = E \cup (F_1^*, WRJ_1^*)$

Step 2. $s = WRJ_1^* - 1$

Step 1. Solve the $R|a_j, WRJ \le s|F+\dfrac{1}{\sum_{i=1}^{n} Max_j\{wr_{ij}\}+1}WRJ$ problem

Let (F_2^*, WRJ_2^*) be the solution

Point 2 in Figure 3.3 is the corresponding efficient point.

$E = E \cup (F_2^*, WRJ_2^*)$

Step 2. $s = WRJ_2^* - 1$

Step 1. Solve the $R|a_j, WRJ \le s|F + \dfrac{1}{\sum_{i=1}^{n} Max_j\{wr_{ij}\} + 1} WRJ$ problem

Let (F_3^*, WRJ_3^*) be the solution

Point 3 in Figure 3.3 is the corresponding efficient point.

$E = E \cup (F_3^*, WRJ_3^*)$

Step 2. $s = WRJ_3^* - 1$

The procedure continues to iterate in a similar manner, until it hits to the other boundary point, namely (F_{LB}, WRJ_{UB}).

Alternately, we could solve the $R|a_j, F \le k|WRJ + e_F F$ problem and vary the value of k between F_{LB} and F_{UB}.

Note that each step of Procedure 3.2 generates an efficient solution. The $R|a_j|F, WRJ$ problem has at most $Min\{F_{UB} - F_{LB} + 1, WRJ_{UB} - WRJ_{LB} + 1\}$, i.e., pseudo-polynomial, number of efficient solutions. Hence the algorithm iterates pseudo-polynomial number of times. In each iteration, one has to solve singly-constrained assignment problem for which polynomial algorithms cannot exist.

A Branch and Bound (BAB) Approach

Recall that the $R|a_j|F, WRJ$ problem is open. This justifies the use of implicit enumeration technique to find the exact set of efficient solutions. We, in this study, propose a branch and bound algorithm.

Our branch and bound algorithm uses the following two phase approach to generate an initial approximate set of efficient solutions.

Phase 1. Generation of extreme supported efficient solutions

Phase 2. Generation of approximate non-extreme supported and unsupported efficient solutions in the neighborhood of the solutions found in Phase 1.

Phase 1:

Recall that we can use Procedure 3.1 to find the extreme supported efficient solutions by using an LP solver. Alternatively, we could generate these solutions through the successive solutions of a linear assignment problem. We start with two known boundary solutions, S_1 and S_2, define ranges for w values of the weighted objective function over which each boundary point is better. In doing so, we solve the following inequality.

$$wF_1 + (1-w)WRJ_1 = wF_2 + (1-w)\,WRJ_2 \qquad (3.8)$$

where (F_i, WRJ_i) is the (F, WRJ) values of S_i and S_i s are ordered such that $F_i < F_{i+1}$ and $WRJ_i > WRJ_{i+1}$.

Note that $w = \dfrac{WRJ_2 - WRJ_1}{F_1 - F_2 + WRJ_2 - WRJ_1}$ solves equation 3.8.

At w, S_1 and S_2 have the same objective function values. In ranges [w, 1] and [0, w], S_1 and S_2 are favored respectively. When a new extreme supported efficient solution is added, we reorder the solutions in such a way that $F_1 < F_2 < F_3$ and $WRJ_1 > WRJ_2 > WRJ_3$ and solve the following two equations simultaneously

$$w_1F_1 + (1-w_1)\,WRJ_1 = w_1F_2 + (1-w_1)\,WRJ_2$$

$$w_2F_2 + (1-w_2)\,WRJ_2 = w_2F_3 + (1-w_2)\,WRJ_3.$$

Then in ranges [w_1,1], [w_2, w_1] and [0, w_2], S_1, S_2 and S_3 are the best schedules respectively. Note that the ranges change once a new schedule is added.

In general, once we have k efficient solutions, we solve k-1 equations: one for each adjacent pair and find k ranges. Exact ranges are available when all extreme supported solutions are found.

Each iteration of our procedure either finds a new extreme supported efficient point, or returns a known extreme supported efficient point, by solving a linear assignment problem with weight w_a. If the former case occurs then there exists an efficient point between S_a and S_{a+1} and the weights are updated with the addition of the new schedule. If the latter case occurs then there cannot exist a supported efficient solution between S_a and S_{a+1}. Then we fix w_a and proceed with w_{a+1} with the hope of generating a new extreme supported point. The algorithm terminates whenever all weights are fixed.

Procedure 3.3

Step 0. Find S_1 and S_2 by solving the $R|a_j|WRJ + e_F F$ and $R|a_j|F + e_{WRJ} WRJ$ problems respectively.

r= # of known extreme supported efficient solutions

k=# of extreme supported efficient solutions with fixed ranges

r=2, k= 1

$$w_1 = \frac{WRJ_2 - WRJ_1}{F_1 - F_2 + WRJ_2 - WRJ_1}$$

$S_L = S_2$

Step 1. Solve the assignment problem with the following objective

Min $w_k F + (1 - w_k) WRJ$

Let S_L be the solution

If S_L is one of the extreme solutions (S_1 or S_2) then go to Step 3.

Step 2. If S_L is either S_k or S_{k+1} then fix w_k let k=k+1, go to Step 1

If S_L is a new schedule then reorder the schedules,

update w_k and w_{k+1} as follows

$$w_k = \frac{WRJ_{k+1} - WRJ_k}{F_k - F_{k+1} + WRJ_{k+1} - WRJ_k}$$

$$w_{k+1} = \frac{WRJ_{k+2} - WRJ_{k+1}}{F_{k+1} - F_{k+2} + WRJ_{k+2} - WRJ_{k+1}}$$

If all w_k are fixed go to Step 3.

Go to Step 1

Step 3. Stop, all r supported efficient solutions are generated.

Procedure 3.3 is similar to the methods proposed by Aneja and Nair (1979) for bicriteria transportation and Visee et al. (1998) for bicriteria knapsack problems.

We illustrate the procedure by the following example problem

Example: Consider six efficient solutions on the following figure

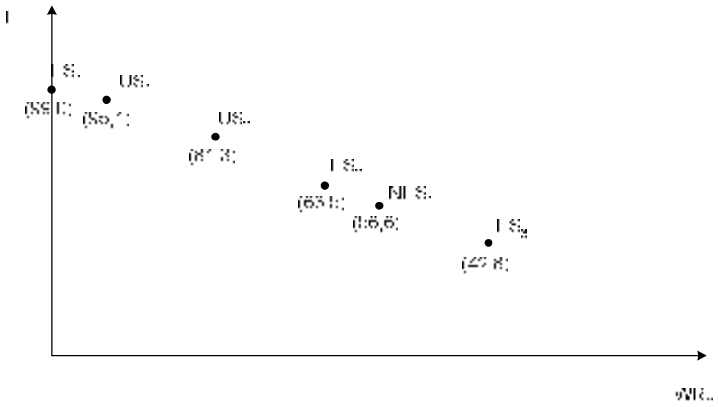

Figure 3.4 Efficient solution of example

Note, that ES_1, ES_2 and ES_3 are extreme supported efficient, NES_1 is nonextreme supported and US_1, US_2 are unsupported efficient solutions. Our algorithm will generate ES_1, ES_2 and ES_3, through the following steps.

Step 0. $S_1 = (99, 0)$ $S_2 = (42, 8)$ $r=2$ $k=1$

$$w_1 = \frac{(8-0)}{(99-42)+(8-0)} = \frac{8}{65} = 0.123$$

w-range	best solution
(0.123, 1]	(42, 8)
[0, 0.123)	(99, 0)

Step 1. Solve the assignment problem with w=0.123.

The optimal solution is at point (*63, 5*)

$r=3$

Ordered set of extreme supported efficient points are

$S_1 = (99, 0)$ $S_2 = (63, 5)$ $S_3 = (42, 8)$

Step 2. $w_1 = \frac{(5-0)}{(99-63)+(5-0)} = \frac{5}{41} = 0.122$

$w_2 = \frac{(8-5)}{(63-42)+(8-5)} = \frac{3}{24} = 0.125$

w-range	best solution
(0.125, 1]	(42, 8)
(0.122, 0.125)	(63, 5)
[0, 0.122)	(99, 0)

Step 1. Solve the assignment problem with w=0.125.

The optimal solution is at points (42, 8) and (63, 5)

Step 2. k=2

Step 1. Solve the assignment problem with w=0.122.

The optimal solution is at points (99, 0) and (63, 5)

Step 2. k=3

All ranges are fixed

Step 3. Stop

r=3 supported points are generated

Ordered set of extreme supported efficient points are

S_1 = (99, 0) S_2 = (63, 5) S_3 = (42, 8)

In solving the assignment problems in Step 0 and Step 2 we use the code generated by Volgenant (1996) designed for the rectangular assignment problems like ours. Note that our problem has n jobs to be assigned to $n*m$ positions. Hence solving regular $n*m$ by $n*m$ assignment problem by defining $n*m-n$ dummy jobs would not be an efficient way. The assignment code of Volgenant (1996) handles this inefficiency by coping with n by $n*m$ rectangular assignment problem. The complexity of the algorithm is n^2m.

Phase 2:

In Phase 2, we start from the first extreme point having minimum total flow time, thereby maximum total reassignment cost of all efficient solutions. For each job that is not on its initial machine, we assign it to its initial machine according to SPT order, while keeping the other assignments fixed. The resulting schedule is added to the list if it is not dominated by any schedule of the list. Among the newly added schedules we select the one having smallest flow time, and compare it with the next extreme supported solution in the list. Among those two schedules, we

continue with the one having smaller total flow time. We repeat the procedure, until the other extreme point of the list is reached. Then we start from this extreme point, i.e., the one having maximum total flow time and zero total reassignment cost and create new schedules by reassigning the jobs from their initial machines to each of the (m-1) machines, while keeping the other assignments fixed. The new schedules, if nondominated, are added to the list. We continue with the new schedule having smallest total reassignment cost or the next schedule of the list whichever has the smallest reassignment cost. We terminate whenever the other extreme point of the list is searched.

We hereafter refer to our two phase procedure as greedy heuristic.

Our Branch and Bound algorithm starts with this list of approximate efficient solutions generated by greedy heuristic, add whenever a nondominated solution is found and remove whenever a solution in the list becomes dominated by a newly generated schedule.

Smith (1956) shows that Shortest Processing Time (SPT) rule minimizes the total flow time on a single machine. Hence, in any efficient sequence SPT should prevail within each machine. We make use of this result in constructing our branch and bound tree.

We generate the partial solutions, i.e., nodes, of the branch and bound tree as follows: At each level, we decide on the job that should be assigned to the first available position of the earliest available machine. We also represent a solution in which no further assignment is made to the earliest available machine, this case corresponds to the removal of that machine. In selecting the first available job we recognize the prevailence of Shortest Processing Time (SPT) rule within each machine. Hence we never branch to a node representing the assignment of job i to machine j if $p_{ij} < p_{lj}$ and job l has assigned to machine j in the partial solution.

Figure 3.2 represents a partial branch and bound tree for n=7 jobs and m=3 machines problem instance whose data are given in Table 3.1.

Table 3.1. **An example problem instance**

i	p_{i1}	p_{i2}	p_{i3}
1	67	6	72
2	85	44	62
3	81	33	55
4	14	21	79
5	54	97	86
6	22	64	61
7	22	94	72

Note that Shortest Processing Time orders of the jobs are as follows:

Machine 1 4-6-7-5-1-3-2

Machine 2 1-4-3-2-6-7-5

Machine 3 3-6-2-1-7-4-5

We assume the initial job assignments are 6-7-5 on machine 1, 1-4-3 on machine 2 and 2 on machine 3. Machine 1 is not available for 98 time units.

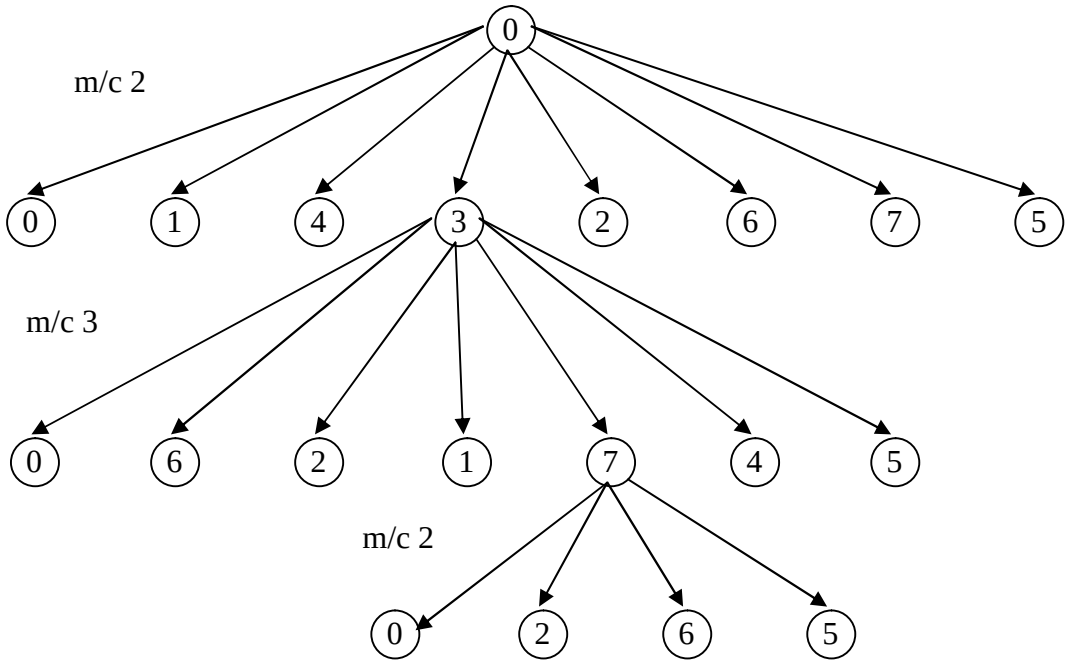

Figure 3.5 The partial branch and bound tree

Note that initially a_1=98, a_2=a_3=0. Machines 2 and 3 are earliest available machines. Assume we arbitrarily select machine 2 for branching. The first node, called 0, represents the case where no further assignments will be made on machine 2. The $(o+1)^{st}$ node at level 1 corresponds to the assignment of the o^{th} job of the SPT sequence on machine 2. Hence the fourth node represents the assignment of job 3. If node 3 is selected for branching then a_2=p_{32}=33 and machine 3 becomes the earliest available machine, emanates six nodes, each node representing the assignment of a particular job to its first available position. The fifth node at level 2, is the fourth unscheduled job of SPT order on machine 3, i.e., job 7. If this node is selected for further branching a_3=p_{73}=72, hence machine 2 becomes the earliest available machine. At level 3, there are four candidate partial solutions, as job 3 was assigned to the first position of machine 2 and there are 3 unscheduled jobs that have higher processing times than that of job 3 on machine 2. These jobs are 2, 6 and 5.

Note that there will be a maximum of $n+m$-1 levels, as n jobs will be assigned and there can be at most m-1, Node 0, selections.

We let M_i denote the set of machines that cannot process job i. Job i cannot be processed by machine j, if such an assignment violates the SPT order or cannot yield an efficient (non-dominated) schedule.

An assignment of job i to machine j violates SPT ordering if $p_{ij} < p_{L_j,j}$ where L_j is the last job assigned to machine j in the partial schedule.

We let $P_F(s)$ and $P_{WRJ}(s)$ be the total flow time and total reassignment cost of partial schedule s. $LB_F(s)$ and $LB_{WRJ}(s)$ are lower bound on the total flowtime and total reassignment cost values of the partial schedule s. $UB_F(WRJ)$ is an upper bound on the F values of the efficient schedules having a total reassignment cost of at least WRJ. Similarly $UB_{WRJ}(F)$ is an upper bound on the WRJ values of the efficient schedules having a total flow time value of at least F units. When job i is assigned to machine j and appended to s, a lower bound on the total flow time value is $P_F(s) + (a_j + p_{ij}) + \sum_{l \in \bar{s}} \text{Min}_{r \notin M_l}\{a_r + p_{lr}\}$ where $\bar{s}$ is the set of unassigned jobs. If this bound is no smaller than $UB_F(LB_{WRJ}(s))$, i.e., an upper bound on the flow time value of the schedules having a total reassignment cost of at least $LB_{WRJ}(s)$ then s is dominated by the approximate efficient schedule in our list having a total flow time value of $UB_F(LB_{WRJ}(s))$. Similarly, if $P_{WRJ}(s) + wr_{ij} + \sum_{l \in \bar{s}} \text{Min}_{r \notin M_l}\{w_{lr}\} \geq UB_{WRJ}(LB_F(s))$ then s is dominated by the schedule in our approximate efficient set having a total reassignment cost of $UB_{WRJ}(LB_F(s))$.

Hence an assignment of job i to machine j is avoided if either

$$P_F(s) + (a_j + p_{ij}) + \sum_{l \in \bar{s}} \text{Min}_{r \notin M_l}\{a_r + p_{lr}\} \geq UB_F(LB_{WRJ}(s)) \text{ or}$$

$$P_{WRJ}(s) + wr_{ij} + \sum_{l \in \bar{s}} \text{Min}_{r \notin M_l}\{w_{lr}\} \geq UB_{WRJ}(LB_F(s))$$

We hereafter refer to the above conditions as efficiency rules.

We let R_j denote the set of jobs that can be processed on machine *j*. Among the machines for which $R_j \neq 0$, we select the earliest available, i.e., the least loaded, one. If the first unsequenced job according to the SPT rule on the selected machine, cannot be assigned to any other machine, we fix that job on that machine and update set M_is, earliest available times and proceed.

For each job in R_j, we calculate a lower bound on *WRJ* and two lower bounds for *F* values. We let s denote the set of jobs appear in the current partial schedule.

Lower bound on WRJ, $LB_{WRJ}(s)$

Note that $LB_{WRJ}(s) = P_{WRJ}(s) + LB_{WRJ}(\bar{s})$

$P_{WRJ}(s)$ = total reassignment cost of jobs in s

$LB_{WRJ}(\bar{s})$= a lower bound on the optimal total reassignment cost of the unscheduled jobs, i.e., the jobs that are not in s .

We let

$$LB_{WRJ}(\bar{s}) = \sum_{i \in \bar{s}} \text{Min}_{j \notin M_i} \{wr_{ij}\}$$

i.e., we choose a weight among the jobs that can be assigned without violating the SPT order and having a potential of generating non-dominated schedules.

Referring to the Branch and Bound tree of Figure 3.2, if no information on the solutions exists, M_js are constructed according to SPT rule.

For a partial schedule where jobs 3 and 7 are assigned to machines 2 and 3 respectively, the lower bound can be calculated as follows:

M_1={2, 3}, M_2={3}, M_4={2}, M_5={ }, M_6={3}

Total reassignment cost of partial schedule = wr_{73}

Lower bound on the total reassignment cost of the remaining jobs =

wr_{11} + Min {wr_{41}, wr_{43}} + Min {wr_{61}, wr_{62}}

as jobs 1, 4 and 6 cannot be assigned to their initial machines, job 1 can only be assigned to machine 1 due to the SPT order.

Lower bound on F

We propose two procedures to find a lower bound on the optimal flow time of unscheduled jobs

i. Lower Bound 1, $LB_{F1}(s)$

We assume all machines are identical and let $p_i = Min_{j \notin M_i}\{p_{ij}\}$. Note that p_i is the minimum processing time for job i, among the machines it can be assigned without violating SPT rule and efficiency rules. Clearly, an optimal total flow time value of the new identical machines problem is a lower bound on the optimal total flow time value of the original unrelated machines problem. The new problem is the $P|a_j|\sum C_i$ problem of the scheduling literature whose optimal solution is due to following rule by Kaspi and Montreuil (1988): Order the jobs by SPT and assign them to the first available machine, in rotation.

Recall our example problem, a lower bound on the total flow time for a partial schedule say node s in which jobs 3 and 7 are assigned to machines 2 and 3 is found as follows:

$p_1 = p_{11} = 67$

$p_2 = \text{Min}\{p_{21}, p_{22}\} = 44$

$p_4 = \text{Min}\{p_{41}\, p_{43}\} = 14$

$p_5 = \text{Min}\{p_{51}, p_{52}, p_{53}\} = 54$

$p_6 = \text{Min}\{p_{61}, p_{62}\} = 22$

SPT order of p_i values is 4-6-2-5-1.

The lower bound schedule has the following assignments:

Machine 1	1			a_1=98
Machine 2	4	6	2	a_2=33
Machine 3	5			a_3=72

$LB_{F1}(\bar{s})$ = (98+67) + (33+14) + (33+14+22) + (33+14+22+44) + (72+54) = 520

$P_F(s)$ = Total Flow Time of the partial schedule = 33 + 72 = 105

$LB_{F1}(s) = LB_{F1}(\bar{s}) + P_F(s) = 625$

If there exists a nondominated schedule s' in the list such that $F(s') \le LB_F(s)$ and $WRJ(s') \le LB_{WRJ}(s)$ then we fathom the node.

If a node cannot be fathomed by $LB_{F1}(s)$, we calculate a more powerful lower bound, $LB_{F2}(s)$ however at an expense of higher computational effort.

ii. Lower Bound 2, $LB_{F2}(s)$

Consider the following assignment model

$$\text{Min} \sum_{i=1}^{n}\sum_{k=1}^{n}\sum_{j=1}^{m}(kp_{ij}+a_j)X_{ikj} + \frac{1}{\sum_{i=1}^{n} Max_j\{wr_{ij}\}+1}\sum_{i,j,k} wr_{ij}X_{ikj}$$

s.t
$$\sum_{k=1}^{n}\sum_{j=1}^{m} X_{ikj} = 1 \qquad \forall\, i$$

$$\sum_{i=1}^{n} X_{ikj} \le 1 \qquad \forall\, j, k$$

$$X_{ikj} \in \{0,1\} \qquad \forall\, i, j, k$$

where

$$x_{ikj} = \begin{cases} 1 & \text{if job } i \text{ is assigned to } k^{th} \text{ position from last on machine } j \\ 0 & \text{otherwise} \end{cases}$$

For a partial schedule s, where s_j is the set of jobs assigned to machine j, and n_j is the cardinality of set s_j we modify a_js as, $a_j = a_j + \sum_{i \in s_j} p_{ij}$, and solve the assignment model with the following objective function

$$\text{Min} \sum_{i \notin s_j}\sum_{k=1}^{n_j'}\sum_{j=1}^{m}(a_j+kp_{ij})X_{ikj} + \frac{1}{\sum_{i=1}^{n} Max_j\{wr_{ij}\}+1}\sum_{i \notin s_j}\sum_{k=1}^{n_j'}\sum_{j=1}^{m} wr_{ij}X_{ikj}$$

where n_j' is an upper bound on the remaining number of jobs to be assigned on machine *j*. If the last job assigned to machine *j* is the l^{th} job of the SPT order then at most *n-l* more jobs can be assigned to machine *j*. Moreover the jobs between *l+1* and *n,* in SPT order, may be assigned to other machines, hence we modify the upper bound, n_j', as the number of unscheduled jobs with no smaller processing time than p_{lj} on machine *j* and do not violate efficiency rules.

Moreover while solving the assignment problem we let $c_{ikj}=M$ if job *i* is the r^{th} unscheduled job of Longest Processing Time (LPT) on machine *j* such that $r < k$, to avoid the assignment of any job to a position that is higher than its index, thereby avoiding a non-SPT ordering. After making these reductions, we solve the $|\bar{s}|$ x $\sum_{\substack{j=1 \\ j\neq N}}^{m} n'_j$ assignment problem using the rectangular assignment algorithm of Volgenant (1996).

The cost coefficients of the assignment model of our example problem for a partial schedule, in which jobs 3 and 7 are assigned to the first position of machines 2 and 3 respectively, are calculated as follows:

Note that $n'_2=3$ as there are 3 unscheduled jobs having higher processing times than p_{32}, these jobs are 2, 6 and 5. As there are two unscheduled jobs having higher processing times than p_{73}, $n'_3=2$. As there are two scheduled jobs, there can be at most *n*-2=5 jobs on machine 1. Hence we solve 5 x 10(5+3+2) assignment problem. Job 1 cannot be assigned to machines 2 and 3 without violating SPT order. Hence $c_{1k2} = c_{1k3} = M$ for all *k*. Jobs 2 and 6 cannot be assigned to machine 3, i.e., $c_{2k2} = c_{6k2} = M$ for k=1, 2. Job 2 cannot be assigned to machine 1, except its first position, i.e., $c_{2k1} = M$ for $k > 1$, as it is the last job of SPT on machine 1. If we have assigned job 2 to a later position we would have violated SPT order, as there is no unscheduled job with higher processing time. Moreover, we set c_{651}=M as job 6 cannot be scheduled fifth on machine 1. Job 4 cannot be assigned to machine 2, i.e., $c_{4k2} = M$ for all *k*. Job 5 is the third longest unscheduled job on machine 1 hence $c_{541} = c_{551} = M$. Similarly job 1 can only be assigned to the first or second position of machine 1 as it is the second longest unscheduled job.

All these information is gathered in Table 3.2.

Table 3.2. Cost Coefficient Matrix of the Assignment Problem

	Machine 1				
	1	**2**	**3**	**4**	**5**
1	$a_1+p_{11}+e\,wr_{11}$	$a_1+2p_{11}+e\,wr_{11}$	M	M	M
2	$a_1+p_{21}+e\,wr_{21}$	M	M	M	M
4	$a_1+p_{41}+e\,wr_{41}$	$a_1+2p_{41}+e\,wr_{41}$	$a_1+3p_{41}+e\,wr_{41}$	$a_1+4p_{41}+e\,wr_{41}$	$a_1+5p_{41}+e\,wr_{41}$
5	a_1+p_{51}	a_1+2p_{51}	a_1+3p_{51}	M	M
6	a_1+p_{61}	a_1+2p_{61}	a_1+3p_{61}	a_1+4p_{61}	M

	Machine 2			**Machine 3**	
	1	2	3	1	2
1	M	M	M	M	M
2	$a_2+p_{22}+e\,wr_{22}$	$a_2+2p_{22}+e\,wr_{22}$	$a_2+3p_{22}+e\,wr_{22}$	M	M
4	M	M	M	$a_3+p_{43}+e\,wr_{43}$	$a_3+2p_{43}+e\,wr_{43}$
5	$a_2+p_{52}+e\,wr_{52}$	M	M	$a_3+p_{53}+e\,wr_{53}$	M
6	$a_2+p_{62}+e\,wr_{22}$	$a_2+2p_{62}+e\,wr_{22}$	M	M	M

where a_1=98, a_2=33, a_3=72

We add $e\,wr_{ij}$ to (i, k, j) when machine j is not the initial machine of job i. For example job 5 was on machine 1 in the initial schedule, hence e appears in all entries for job 5 except the ones associated to machine 1. The optimal solution to the assignment gives the following schedule.

Machine 1 4 - 1

Machine 2 2 - 6

Machine 3 5

$LB_{F2}(\bar{s})$ = (98+14) + (98+14+67) + (33+44) + (33+33+64) + (72+86) = 667

$P_F(s)$ = Total Flow Time of the partial schedule

= 105

$LB_{F2}(s)$ = Lower bound on the total flow time of s

= 772

$F(\bar{s})$ = 772

$WRJ(\bar{s})$ = wr_{41} + wr_{11} + wr_{22} + wr_{62} + wr_{52}

The actual total flow time of the schedule is $F(\bar{s})$ and the actual total reassignment cost is $WRJ(\bar{s})$. Note that actual flow time value is equal to the lower bound on the flow time value found using assignment solution, i.e., $F(\bar{s})$ = $LB_{F2}(s)$. We add the resulting schedule $\bar{s}$ to the list of approximate efficient schedules if there does not exist a schedule s' such that $F(s') \le F(\bar{s})$ and $WRJ(s') \le WRJ(\bar{s})$. If there exists a schedule $\hat{s}$ such that $F(\hat{s}) \ge F(\bar{s})$ and $WRJ(\hat{s}) \ge WRJ(\bar{s})$, then $\hat{s}$ is dominated by $\bar{s}$, and therefore is taken out of the list.

Note that $Max\{ LB_{F1}(s), LB_{F2}(s)\}$ is a lower bound on the optimal F values of the nodes emanating from s. Hence when we proceed to "the next level, say node s_c, we first check whether there exists a schedule s' such that $F(s') \le Max\{ LB_{F1}(s), LB_{F2}(s)\}$ and $WRJ(s') \le WRJ(s)$. If such a schedule s' exists, we fathom the node. Otherwise, we calculate $LB_{F1}(s_c)$ and proceed.

Our algorithm returns the set of all efficient solutions after evaluating all nodes, implicitly.

3.3.8 The $R|a_j|f(F,WRJ)$ problem

In this section, we address the problem of finding an optimal solution for a specified general non-decreasing function of F and WRJ.

When, the function, f, is a linear function of F and WRJ then one can use an assignment model with the following objective function

$$\text{Min } w_1 \sum_{i,j,k} (a_j + kp_{ij}) X_{ikj} + w_2 \sum_{i,j,k} wr_{ij} X_{ikj} \equiv \text{Min } w_1 F + w_2 WRJ$$

and find an optimal solution in polynomial time.

When f is a non-linear function, finding an optimal solution to our model with constraint sets (3.2), (3.3), (3.4) and binary decision variables, would not be possible by available mathematical programming softwares. For non-linear f, one can generate all efficient solutions and select the one that minimizes the objective function value. However such an approach may not be time-efficient as each generation requires a solution of a singly-constrained assignment problem in exponential time. To overcome this difficulty, we develop two optimization algorithms that implicitly generate the efficient set. The first algorithm, we call

Integer Programming Based (IPB) algorithm, solves successive constrained optimization problem. The second algorithm is a branch and bound approach that makes implicit enumeration of the efficient schedules.

Integer Programming Based (IPB) Approach

An IPB algorithm starts with an initial feasible solution that is found by generating the extreme supported efficient solutions. Each iteration of the algorithm generates an efficient schedule by setting an upper limit on the F and WRJ values of any schedule that can improve the best known solution, namely f_{BEST}. By setting these limits, we eliminate some portions of the objective space that cannot reside the optimal solution. Kondakci et al. (1996) implement an idea of imposing upper limits on one criterion for their bicriteria single machine scheduling problem.

Moreover, we set lower limits on the F and WRJ values by solving the LP relaxations of the singly-constrained assignment problem. If the f value found by setting the lower limits is no better than f_{BEST}, then we terminate by recording the optimality of the best known schedule.

The smallest f value among the extreme supported efficient schedules is used as an initial f_{BEST}. We update f_{BEST} whenever a feasible schedule with smaller f value is reached. Procedure 3.4, below, is the stepwise description of our approach.

Procedure 3.4 Finding an Optimal Solution by Integer Programming Based Algorithm

Step 0. Solve the $R|a_j|F+\dfrac{1}{\sum_{i=1}^{n} Max_j\{wr_{ij}\}+1}WRJ$ problem and form a right-shift schedule.

Let $F_{LB} = F$ value that solves the $R|a_j|F+\dfrac{1}{\sum_{i=1}^{n} Max_j\{wr_{ij}\}+1}WRJ$ problem

$F_{UB} = F$ value of the right-shift schedule

$WRJ_{LB} = WRJ$ value of the right-shift schedule, i.e., zero

WRJ_{UB}=WRJ value that solves the $R \mid a_j \mid F + \frac{1}{\sum_{i=1}^{n} Max_j\{wr_{ij}\}+1} WRJ$ problem

Apply Procedure 3.1 to generate the set of

extreme supported efficient schedules (*ESE*)

$f_{BEST} = Min_{s \in ESE}\{f(WRJ(s), F(s))\}$

Step 1. If $f(F_{LB}, WRJ_{LB}) \geq f_{BEST}$ then STOP

Find WRJ_a that solves $f(F_{LB}, WRJ_a) = f_{BEST}$

$WRJ_{UB} = \lceil WRJ_a \rceil - 1$

If $WRJ_{UB} \leq WRJ_{LB}$ then STOP

Solve the LP Relaxation of the $R \mid a_j, WRJ \leq WRJ_{UB} \mid F + e_{WRJ} WRJ$ problem

Let (F^*, WRJ^*) be the solution

$F_{LB} = \lceil F^* \rceil$

If $f(F_{LB}, WRJ_{LB}) \geq f_{BEST}$ then STOP

If the resulting solution is integer then

f_{BEST} = Min {f_{BEST} , $f(F^*, WRJ^*)$}

$WRJ_{UB} = WRJ^* - 1$

If $WRJ_{UB} \leq WRJ_{LB}$ then STOP

Repeat Step 1

Step 2. Find F_a value that solves $f(F_a, WRJ_{LB}) = f_{BEST}$

$F_{UB} = \lceil F_a \rceil - 1$

If $F_{UB} \leq F_{LB}$ then STOP

Solve the LP Relaxation of the $R \mid a_j, F \leq F_{UB} \mid WRJ + e_F F$ problem

Let (F^*, WRJ^*) be the solution

$WRJ_{LB} = \lceil WRJ^* \rceil$

If $f(F_{LB}, WRJ_{LB}) \geq f_{BEST}$ then STOP

If the resulting solution is integer then

f_{BEST} = Min {f_{BEST} , $f(F^*, WRJ^*)$}

$F_{UB} = F^* - 1$

If $F_{UB} \leq F_{LB}$ then STOP

Go to Step 1

Step 3. Solve the $R \mid a_j, WRJ \leq WRJ_{UB} \mid F + e_{WRJ} WRJ$ problem

Let (F^*, WRJ^*) be the solution

$F_{LB} = F^* + 1$

$WRJ_{UB} = WRJ^* - 1$

If $F_{UB} \leq F_{LB}$ or $WRJ_{UB} \leq WRJ_{LB}$ then STOP

f_{BEST} = Min {f_{BEST} , $f(F^*, WRJ^*)$}

Solve the $R \mid a_j, F \leq F_{UB} \mid WRJ + e_F F$ problem

Let (F^*, WRJ^*) be the solution

$WRJ_{LB} = WRJ^* + 1$

$F_{UB} = F^* - 1$

If $F_{UB} \leq F_{LB}$ or $WRJ_{UB} \leq WRJ_{LB}$ then STOP

f_{BEST} = Min {f_{BEST} , $f(F^*, WRJ^*)$}

Go to Step 1

The following figure illustrates the progress of Procedure 3.4.

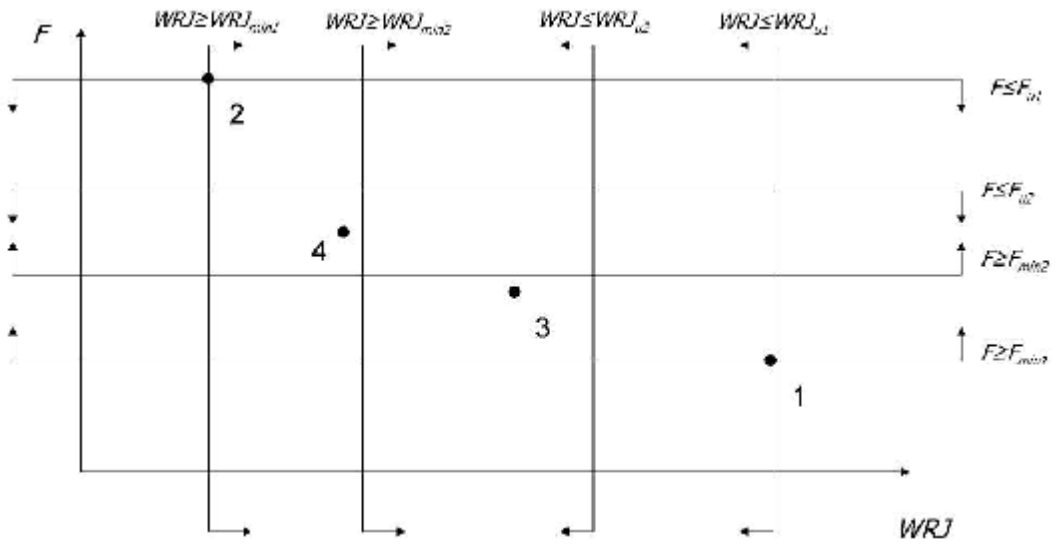

Figure 3.6 Progress of Procedure 3.4

Step 0. Initialize WRJ_{LB}, F_{LB}, and f_{BEST}, by identifying boundary points and generating set of extreme supported efficient solutions.

Step 1. Find WRJ_a that solves $f(F_{LB}, WRJ_a) = f_{BEST}$

$WRJ_{UB} = \lceil WRJ_a \rceil - 1 = WRJ_{u1}$

Solve the LP Relaxation of the $R \mid a_j, WRJ \le WRJ_{u1} \mid F + e_{WRJ} WRJ$ problem

Let (F_1^*, WRJ_1^*) be the solution

Point 1 in Figure 3.6 is the corresponding efficient point

$F_{LB} = \lceil F_1^* \rceil = F_{min1}$

$f_{BEST} = \text{Min } \{f_{BEST}, f(F_1^*, WRJ_1^*)\}$

$WRJ_{UB} = WRJ_1^* - 1$

Step 1. Find WRJ_a that solves $f(F_{LB}, WRJ_a) = f_{BEST}$

$WRJ_{UB} = \lceil WRJ_a \rceil - 1$

Step 2. Find F_a value that solves $f(F_a, WRJ_{LB}) = f_{BEST}$

$F_{UB} = \lceil F_a \rceil - 1 = F_{u1}$

Solve the LP Relaxation of the $R \mid a_j, F \leq F_{U1} \mid WRJ + e_F F$ problem

Let (F_2^*, WRJ_2^*) be the solution

Point 2 in Figure 3.6 is the corresponding efficient point

$WRJ_{LB} = \lceil WRJ^* \rceil = WRJ_{min1}$

f_{BEST} = Min {f_{BEST} , $f(F_2^*, WRJ_2^*)$}

$F_{UB} = F_2^* - 1$

Step 1. Find WRJ_a that solves $f(F_{LB}, WRJ_a) = f_{BEST}$

$WRJ_{UB} = \lceil WRJ_a \rceil - 1 = WRJ_{u2}$

Step 2. Find F_a value that solves $f(F_a, WRJ_{LB}) = f_{BEST}$

$F_{UB} = \lceil F_a \rceil - 1 = F_{u2}$

Step 3. Solve the $R \mid a_j, WRJ \leq WRJ_{u2} \mid F + e_{WRJ} WRJ$ problem

Let (F_3^*, WRJ_3^*) be the solution

Point 3 in Figure 3.6 is the corresponding efficient point

$F_{LB} = F_3^* + 1$

$WRJ_{UB} = WRJ_3^* - 1$

f_{BEST} = Min {f_{BEST} , $f(F^*, WRJ^*)$}

Solve the $R \mid a_j, F \leq F_{u2} \mid WRJ + e_F F$ problem

Let (F_4^*, WRJ_4^*) be the solution

Point 4 in Figure 3.6 is the corresponding efficient point

$WRJ_{LB} = WRJ_4^* + 1$

$F_{UB} = F_4^* - 1$

f_{BEST} = Min {f_{BEST} , $f(F^*, WRJ^*)$}

The procedure continues to iterate in a similar manner, until the upper bound and lower bound constraints hits to each other, thus the current problem becomes infeasible.

A Branch and Bound (BAB) Approach

We employ the branching scheme designed for the $R|a_j|F,WRJ$ problem to solve the $R|a_j|f(F,WRJ)$ problem. In doing so, we use the efficiency rules and lower bounds designed for the $R|a_j|f(F,WRJ)$ with the following modifications.

Efficiency rules

We put machine j to M_i if

$f(P_F(s) + (a_j + p_{ij}) + \sum_{l\in\bar{s}} \text{Min}_{r\notin M_{il}}\{a_r + p_{lr}\},\ P_{WRJ}(s) + wr_{ij} + \sum_{l\in\bar{s}} \text{Min}_{r\notin M_l}\{w_{lr}\}) \geq$

f_{BEST} where f_{BEST} is the best known objective function value.

Note that $P_F(s) + (a_j + p_{ij}) + \sum_{l\in\bar{s}} \text{Min}_{r\notin M_{il}}\{a_r + p_{lr}\}$, is a lower bound on F values of the efficient schedules emanating from s when job i is assigned to machine j. Similarly, $P_{WRJ}(s) + wr_{ij} + \sum_{l\in\bar{s}} \text{Min}_{r\notin M_l}\{w_{lr}\}$, is a lower bound on the associated WRJ values. This leads to a lower bound of $f(P_F(s) + (a_j + p_{ij}) + \sum_{l\in\bar{s}} \text{Min}_{r\notin M_{il}}\{a_r + p_{lr}\},\ P_{WRJ}(s) + wr_{ij} + \sum_{l\in\bar{s}} \text{Min}_{r\notin M_l}\{w_{lr}\})$ on the function value, which is compared with f_{BEST}. If it is no smaller, then the assignment of job i to machine j should be avoided, and this information should be used in further bound computations.

We initially take f_{BEST} as the minimum f value of the extreme supported schedules and the approximate schedules generated by the greedy heuristic. We update f_{BEST} whenever we find a complete solution with smaller f value.

Lower Bounds

We fathom node s, if $f(Max\{LB_{F1}(s_P), LB_{F2}(s_P)\}, LB_{WRJ}(s_P)) \geq f_{BEST}$ where s_P is the parent node of s. If not, we first check whether $f(LB_F(s), LB_{WRJ}(s)) \geq f_{BEST}$. If $f(LB_{F1}(s), LB_{WRJ}(s)) \geq f_{BEST}$, we fathom the node. Otherwise we compute the assignment bound, $LB_{F2}(s)$. If $f(LB_{F2}(s), LB_{WRJ}(s)) \geq$

f_{BEST}, we fathom the node else we list the nodes in their nondecreasing order of f(Max($LB_{F1}(s)$, $LB_{F2}(s)$), $LB_{WRJ}(s)$) values and select the node at the top of the list for branching.

When we solve the assignment problem at a particular node, we evaluate the resulting schedule, $\bar{s}$. If $f(F(\bar{s}),WRJ(\bar{s})) < f_{BEST}$, we update f_{BEST}.

Whenever a need for rescheduling arises, i.e., machine disruption occurs, one can employ the above procedures. In this sense, they can be classified as on-line procedures. Moreover multiple simultaneous disruptions can be handled by modifying a_j values.

3.4 Computational Experience

We conduct a computational experiment to assess the efficiency of our algorithms. We generate random problem instances having n = 40, 60, 80, 100 jobs and m = 4, 8, 12 machines. The job processing times, p_{ij}s, are drawn from two discrete uniform distributions between [1,100] and [50,100]. We select two levels for processing times, to see the effect of processing time variability and magnitude on the performance of our algorithms. Similarly, to see the effect of the variability and magnitude of the reassignment costs, wr_{ij}s, are drawn from two discrete uniform distributions between [1,60] and [30,60].

The disruption duration, D, is set to three levels: Long (L), Medium (M) and Short (S). The aim here is to study the effect of the disruption duration on algorithm performances. For level L, D is set to the completion time of the last job on the disrupted machine in the initial schedule. Level M has the half of the duration of level L. Level S has half of the duration of level M.

We consider the following two non-linear objective functions that are non-decreasing in F and WRJ, similar to Kondakçı et al. (1996).

$$f_1 = \left(\frac{F - F_{LB}}{F_{UB} - F_{LB}} \right)^2 + \left(\frac{WRJ - WRJ_{LB}}{WRJ_{UB} - WRJ_{LB}} \right)^2$$

$$f_2 = \left(\frac{F - F_{LB}}{F_{UB} - F_{LB}} \right)^8 + \left(\frac{WRJ - WRJ_{LB}}{WRJ_{UB} - WRJ_{LB}} \right)^8$$

We refer to f_1 and f_2 as quadratic and quasi-chebyschev functions respectively.

To generate all efficient solutions, we generate 4x4x2x2x3=144 problem combinations, and to solve a nonlinear function we generate 144x2=288 problem combinations. For each problem combination, we consider 10 instances. Hence as a total of 1440 and 2880 problem instances are generated and solved for efficient set generation and nonlinear function minimization problems, respectively.

We conduct all experiment on a PC with Intel Pentium 4 2.8 Ghz processor and 1 GB of RAM running under Linux, specifically Fedora Core 5, operating system. We implement our optimization and Branch and Bound algorithms in C, compiled with GCC 4 and utilized Borland C++BuilderX as the development environment. We solve our integer and linear programming models using CPLEX 8.1.1.

We set a termination limit of 2 hours for 60 jobs and 3 hours for 80 and 100 jobs for generation of efficient set algorithms (both classical approach and Branch and Bound algorithm). To our optimization algorithms, we set a termination limit of 1 hour for 60 jobs and 1.5 hours for 80 and 100 jobs for both Integer Programming Based and Branch and Bound approaches. We use different termination limits due to different complexity levels of the problems.

We first investigate the performances of the algorithms we used in generation of the efficient set: The Classical Approach (CA) and Branch and Bound Algorithm (BAB). Tables 3.3 through 3.8 report the average and maximum computation times of the CA and BAB algorithm. The average and maximum number of nodes generated by the Branch and Bound algorithm are included. The tables also give the average and maximum number of efficient solutions, and the number of times BAB algorithm finds the efficient solutions quicker than CA. In Tables 3.3 and 3.4 the results associated to short disruption duration are reported for p_{ij} ~ U[1,100] and p_{ij} ~ U[50,100] respectively. In Tables 3.5 and 3.6, and Tables 3.7 and 3.8, the same results are given for medium and long disruption duration cases, respectively. The tables do not include the instances for n=80, m=12 when the disruption duration is long, i.e. $D=L$, and p_{ij}~U[50,100], as our preliminary experiments have revealed that the majority of the instances could not be solved

within termination limit of 3 hours. Table 3.9 summarizes the average case results, in particular average number of efficient solutions, average CPU times of both algorithms. The table also includes the number of instances out of 10, where the Branch and Bound algorithm outperforms Classical Approach, in terms of solutions times.

Table 3.3 Performance of Efficient Set Generation Algorithms, p_{ij} ~U[1,100], $D=S$

n	*m*	*wrij*	Classical Approach				Branch&Bound			
			CPU time (sec)		# Efficient Solutions		CPU time (sec)		# Nodes	
			Avg	Max	Avg	Max	Avg	Max	Avg	Max
40	4	U[1,60]	0.6	1.9	4	11	0.0	0.1	621	1,410
		U[30,60]	0.7	1.1	4	8	0.0	0.1	974	3,908
	8	U[1,60]	0.7	1.5	3	7	0.1	0.3	1,151	5,905
		U[30,60]	0.4	0.8	2	3	0.0	0.0	97	675
	12	U[1,60]	0.8	2.0	3	5	0.0	0.2	387	1,690
		U[30,60]	0.8	3.9	2	5	0.0	0.1	281	2,577
60	4	U[1,60]	3.4	6.0	9	12	0.3	0.5	2,573	5,581
		U[30,60]	4.6	9.0	8	13	0.3	0.5	3,238	5,814
	8	U[1,60]	1.6	7.7	3	9	0.3	1.6	1,294	7,268
		U[30,60]	3.8	19.9	4	10	0.2	0.9	1,657	9,061
	12	U[1,60]	3.2	11.5	3	5	0.2	0.5	1,435	6,557
		U[30,60]	1.7	4.9	2	5	0.2	1.2	802	5,856
80	4	U[1,60]	11.1	28.7	11	17	1.2	3.2	5,342	11,679
		U[30,60]	36.2	169.6	13	26	1.5	4.4	11,950	35,184
	8	U[1,60]	13.5	51.3	7	14	1.2	3.5	4,836	12,354
		U[30,60]	15.8	41.7	6	12	1.0	2.9	5,221	12,047
	12	U[1,60]	13.2	54.8	4	10	1.4	4.5	2,900	9,603
		U[30,60]	21.9	52.1	4.7	7	1.5	5.7	6,347	17,183
100	4	U[1,60]	48.8	111.8	18	35	4.9	11.5	19,412	41,109
		U[30,60]	96.3	348.7	17	36	7.7	27.8	43,430	180,011
	8	U[1,60]	48.0	127.2	10	17	8.3	38.2	17,370	80,791
		U[30,60]	56.7	190.3	7	12	3.2	10.8	15,298	51,824
	12	U[1,60]	51.5	159.8	4	11	9.4	38.3	10,311	46,039
		U[30,60]	29.6	135.8	4	5	2.2	6.3	7,884	20,167

Table 3.4 Performance of Efficient Set Generation Algorithms, p_{ij} ~U[50,100], $D=S$

n	m	$wrij$	Classical Approach				Branch&Bound			
			CPU time (sec)		# Efficient Solutions		CPU time (sec)		# Nodes	
			Avg	Max	Avg	Max	Avg	Max	Avg	Max
40	4	U[1,60]	1.4	4.0	8	16	0.1	0.5	879	2,462
		U[30,60]	1.1	1.8	6	9	0.1	0.3	959	1,950
	8	U[1,60]	1.8	3.8	5	7	0.3	0.7	894	1,480
		U[30,60]	2.2	4.3	5	7	0.2	1.0	1,426	4,621
	12	U[1,60]	0.3	1.0	2	4	0.0	0.0	17	120
		U[30,60]	0.9	4.4	3	8	0.1	0.7	380	2,179
60	4	U[1,60]	6.5	10.5	14	19	0.5	0.6	3,316	4,648
		U[30,60]	9.7	20.3	13	22	1.2	1.8	8,846	20,266
	8	U[1,60]	9.1	19.1	10	17	3.8	21.0	5,894	30,092
		U[30,60]	24.5	56.0	12	25	3.9	7.6	17,722	46,155
	12	U[1,60]	8.0	21.6	4	7	1.9	8.7	2,366	12,666
		U[30,60]	14.3	44.7	6	9	1.4	4.3	2,859	5,624
80	4	U[1,60]	30.2	69.0	20	26	3.8	7.0	11,469	21,990
		U[30,60]	54.2	134.2	19	33	7.1	13.6	46,158	110,102
	8	U[1,60]	44.6	104.2	15	24	13.6	32.9	19,378	45,044
		U[30,60]	48.7	91.5	10	14	10.5	26.5	20,110	43,581
	12	U[1,60]	52.7	90.7	10	16	42.4	142.5	34,375	108,049
		U[30,60]	95.2	243.7	10	23	38.0	243.4	78,654	501,407
100	4	U[1,60]	125.6	286.0	29	39	18.9	38.3	55,502	120,156
		U[30,60]	401.4	959.6	38	64	58.1	111.2	316,406	802,814
	8	U[1,60]	150.7	229.9	22	32	176.2	926.6	150,466	766,661
		U[30,60]	218.3	369.5	17	29	53.0	127.8	134,893	345,285
	12	U[1,60]	139.9	262.4	13	28	133.5	375.3	73,136	203,288
		U[30,60]	207.8	735.7	13	27	105.1	255.0	98,821	246,566

Table 3.5 Performance of Efficient Set Generation Algorithms, p_{ij}~U[1,100], D=M

n	m	$wrij$	Classical Approach				Branch&Bound			
			CPU time (sec)		# Efficient Solutions		CPU time (sec)		# Nodes	
			Avg	Max	Avg	Max	Avg	Max	Avg	Max
40	4	U[1,60]	1.9	7.0	12	35	0.2	0.6	2,097	8,265
		U[30,60]	1.8	3.1	10	16	0.2	0.5	2,362	6,180
	8	U[1,60]	1.3	2.7	5	11	0.1	0.2	1,534	3,439
		U[30,60]	0.9	1.9	4	6	0.1	0.3	1,014	2,684
	12	U[1,60]	2.1	4.8	5	10	0.2	0.5	1,404	3,866
		U[30,60]	1.3	3.6	3	8	0.1	0.4	1,094	6,576
60	4	U[1,60]	18.5	47.5	23	42	1.6	2.8	10,495	15,835
		U[30,60]	34.7	93.6	26	36	2.6	8.3	27,013	66,931
	8	U[1,60]	8.9	32.0	9	26	2.7	13.5	11,037	46,762
		U[30,60]	7.6	23.9	9	20	0.7	1.8	8,114	28,192
	12	U[1,60]	5.4	14.3	5	10	0.9	3.0	3,401	10,461
		U[30,60]	4.0	8.0	4	7	0.3	1.0	2,313	5,833
80	4	U[1,60]	136.4	731.9	28	45	10.2	41.6	39,543	112,202
		U[30,60]	1008.5	7212.2	40	85	22.3	89.2	199,268	988,844
	8	U[1,60]	26.0	58.1	13	24	6.9	32.3	22,869	114,237
		U[30,60]	42.8	93.1	15	28	7.2	28.8	47,371	204,906
	12	U[1,60]	24.3	54.3	8	19	10.3	28.3	18,606	46,245
		U[30,60]	56.5	115.6	11	20	11.8	46.4	59,240	249,393
100	4	U[1,60]	1443.6	7792.0	43	74	55.4	223.9	165,032	646,804
		U[30,60]	3968.7	10800.0	47	79	299.8	971.8	1,855,189	6,256,106
	8	U[1,60]	202.3	538.9	30	52	309.7	2076.4	433,433	2,806,813
		U[30,60]	228.8	534.5	22	31	35.2	110.0	176,871	599,235
	12	U[1,60]	124.4	552.3	14	45	256.4	1992.9	273,965	2,176,723
		U[30,60]	114.2	199.1	11	17	17.4	29.3	61,242	170,224

Table 3.6 Performance of Efficient Set Generation Algorithms, p_{ij} ~U[50,100], D=M

n	m	$wrij$	Classical Approach				Branch&Bound			
			CPU time (sec)		# Efficient Solutions		CPU time (sec)		# Nodes	
			Avg	Max	Avg	Max	Avg	Max	Avg	Max
40	4	U[1,60]	4.5	15.6	19	38	0.5	1.4	3,498	9,842
		U[30,60]	4 4	6 9	17	27	0,6	1 7	8,412	18,117
	8	U[1,60]	4 8	7 3	13	19	1,3	3 9	5,270	14,718
		U[30,60]	6 2	9 7	12	19	0,9	1 5	5,759	13,609
	12	U[1,60]	2 3	6 1	7	13	0,3	0 8	1,608	4,860
		U[30,60]	6 4	15 2	8	16	1,5	10 5	4,990	25,435
60	4	U[1,60]	30.9	74.8	31	45	5.3	12.6	34,541	86,563
		U[30,60]	68.8	131.1	41	68	33.5	69.4	422,995	996,867
	8	U[1,60]	30.8	49.4	20	29	28.6	91.7	58,128	180,013
		U[30,60]	80.1	153.9	29	61	78.1	167.3	386,276	965,357
	12	U[1,60]	28.3	52.7	13	21	19.1	40.0	22,242	74,170
		U[30,60]	49 4	92 1	15	22	10,2	22 7	26,268	70,742
80	4	U[1,60]	488 5	2963 2	51	71	92,8	331 0	402,577	1,414,968
		U[30,60]	694 4	2090 7	65	99	1477,5	2733 0	14,082,670	33,656,578
	8	U[1,60]	140 2	232 9	32	45	212,4	845 6	501,417	2,329,652
		U[30,60]	235 8	291 6	31	44	244,4	482 6	902,601	2,128,374
	12	U[1,60]	158.7	277.9	26	40	2003.7	5865.3	1,494,865	4,035,765
		U[30,60]	274.1	556.7	25	48	588.8	2410.1	1,311,452	5,702,283
100	4	U[1,60]	2439.5	10800.0	81	114	3088.9	10800.0	11,594,294	35,121,073
		U[30,60]	7205 4	10800 0	107	186	10379,8	10800 3	57,201,180	74,512,924
	8	U[1,60]	404.2	660.7	46	61	4753.0	10800.0	5,412,054	17,291,895
		U[30,60]	913.6	1340.3	47	60	4620,7	10800.1	14,240,991	39,039,979
	12	U[1,60]	432.5	684.4	37	61	6179.3	10800.3	4,277,677	10,947,764
		U[30,60]	1084.5	1761.0	39	66	4008.1	10800.1	7,485,475	30,328,598

Table 3.7 Performance of Efficient Set Generation Algorithms, p_{ij}~U[1,100], D=L

n	m	$wrij$	Classical Approach				Branch&Bound			
			CPU time (sec)		# Efficient Solutions		CPU time (sec)		# Nodes	
			Avg	Max	Avg	Max	Avg	Max	Avg	Max
40	4	U[1,60]	8.0	17.2	29	56	1.8	5.4	16,003	46,408
		U[30,60]	18.1	69.6	33	59	1.8	5.5	43,079	145,734
	8	U[1,60]	4.4	10.3	13	21	0.9	3.1	12,165	47,855
		U[30,60]	3.7	8.7	10	19	0.6	3.2	8,186	38,671
	12	U[1,60]	5.6	13.2	10	20	1.8	6.3	17,692	63,676
		U[30,60]	3.7	7.9	7	11	0.3	1.5	7,007	33,583
60	4	U[1,60]	138.0	378.7	60	110	42.4	119.9	281,800	601,032
		U[30,60]	1952.2	6488.6	81	120	235.8	1211.1	2,764,979	12,544,461
	8	U[1,60]	37.3	109.5	27	60	691.6	6597.3	2,571,030	24,438,002
		U[30,60]	37.8	63.0	23	35	12.2	35.0	195,407	769,750
	12	U[1,60]	33.7	74.3	16	26	38.6	156.8	242,825	1,104,784
		U[30,60]	21.6	46.0	11	17	3.7	14.0	28,638	156,677
80	4	U[1,60]	3331.5	10800.0	91	139	2039.9	9485.4	9,776,836	54,037,769
		U[30,60]	5396.2	10800.0	120	209	5574.8	10800.0	46,346,138	106,799,165
	8	U[1,60]	117.3	244.2	35	61	527.6	3141.1	1,259,538	7,477,454
		U[30,60]	221.2	563.6	39	85	2155.6	10800.0	14,459,373	72,270,898
	12	U[1,60]	134.7	406.9	28	72	1924.1	10800.0	3,658,208	20,278,864
		U[30,60]	153.1	435.3	25.9	56	853.8	6429.0	5,878,416	44,150,979

Table 3.8 Performance of Efficient Set Generation Algorithms, p_{ij}~U[50,100], D=L

n	m	$wrij$	Classical Approach				Branch&Bound			
			CPU time (sec)		# Efficient Solutions		CPU time (sec)		# Nodes	
			Avg	Max	Avg	Max	Avg	Max	Avg	Max
40	4	U[1,60]	20.4	61.6	43	75	14.4	29.0	146,221	387,086
		U[30,60]	31.2	65.9	54	77	58.8	187.1	1,022,010	3,083,568
	8	U[1,60]	13.5	25.8	27	36	31.8	94.7	178,319	483,608
		U[30,60]	25.2	38.9	37	55	23.9	79.9	257,857	887,923
	12	U[1,60]	11.1	19.6	20	31	19.1	60.9	81,972	217,419
		U[30,60]	21.1	79.8	21	47	113.8	990.7	333,135	2,559,015
60	4	U[1,60]	506.2	3664.4	76	87	1591.2	7200.1	10,155,466	48,449,896
		U[30,60]	1402.3	7200.0	127	168	-	-	-	-
	8	U[1,60]	84.1	116.8	45	71	2111.3	7200.0	4,902,522	16,016,238
		U[30,60]	208.9	361.0	66	108	5387.6	7200.0	37,844,372	61,773,380
	12	U[1,60]	82.7	137.7	33	48	1936.1	7200.0	3,328,701	12,941,628
		U[30,60]	154.8	204.8	40	56	796.8	2274.5	3,013,844	7,764,133

Table 3.9 Average Performance of Efficient Set Generation Algorithms

n	m	$wrij$	Average CPU Times (sec)																	
			$d = S$						$d = M$						$d = L$					
			$p_{ij} \sim U(1, 100)$			$p_{ij} \sim U(50, 100)$			$p_{ij} \sim U(1, 100)$			$p_{ij} \sim U(50, 100)$			$p_{ij} \sim U(1, 100)$			$p_{ij} \sim U(50, 100)$		
			CL	BAB	CNT*	CL	BAB	CNT	CL	BAB	CNT	CL	BAB	CNT	CL	BAB	CNT	CL	BAB	CNT
40	4	U[1,60]	0.6	0.0	10	1.4	0.1	10	1.9	0.2	10	4.5	0.5	10	8.0	1.8	10	20.4	14.4	7
		U[30,60]	0.7	0.0	10	1.1	0.1	10	2	0.2	10	4.4	0.6	10	18.1	1.8	10	31.2	58.8	2
	8	U[1,60]	0.7	0.1	10	1.8	0.3	10	1	0.1	10	4.8	1.3	10	4.4	0.9	10	13.5	31.8	3
		U[30,60]	0.4	0.0	10	2.2	0.2	10	1	0.1	10	6.2	0.9	10	3.7	0.6	10	25.2	23.9	8
	12	U[1,60]	0.8	0.0	10	0.3	0.0	10	2	0.2	10	2.3	0.3	10	5.6	1.8	10	11.1	19.1	5
		U[30,60]	0.8	0.0	10	0.9	0.1	10	1	0.1	10	6.4	1.5	10	3.7	0.3	10	21.1	113.8	7
60	4	U[1,60]	3.4	0.3	10	6.5	0.5	10	19	1.6	10	30.9	5.3	10	138.0	42.4	10	506.2	1591.2	0
		U[30,60]	4.6	0.3	10	9.7	1.2	10	35	2.6	10	68.8	33.5	9	1952.2	235.8	9	1402.3	-	-
	8	U[1,60]	1.6	0.3	10	9.1	3.8	9	9	2.7	10	30.8	28.6	9	37.3	691.6	6	84.1	2111.3	0
		U[30,60]	3.8	0.2	10	24.5	3.9	10	8	0.7	10	80.1	78.1	7	37.8	12.2	10	208.9	5387.6	0
	12	U[1,60]	3.2	0.2	10	8.0	1.9	10	5	0.9	10	28.3	19.1	10	33.7	38.6	7	82.7	1936.1	0
		U[30,60]	1.7	0.2	10	14.3	1.4	10	4	0.3	10	49.4	10.2	10	21.6	3.7	10	154.8	796.8	1
80	4	U[1,60]	11.1	1.2	10	30.2	3.8	10	136	10.2	10	488.5	92.8	10	3331.5	2039.9	8			
		U[30,60]	36.2	1.5	10	54.2	7.1	10	1,008	22.3	10	694.4	1477.5	1	5396.2	5574.8	3			
	8	U[1,60]	13.5	1.2	10	44.6	13.6	10	26	6.9	9	140.2	212.4	6	117.3	527.6	5			
		U[30,60]	15.8	1.0	10	48.7	10.5	10	43	7.2	10	235.8	244.4	5	221.2	2155.6	5			
	12	U[1,60]	13.2	1.4	10	52.7	42.4	8	24	10.3	10	158.7	2003.7	2	134.7	1924.1	5			
		U[30,60]	21.9	1.5	10	95.2	38.0	10	57	11.8	9	274.1	588.8	6	153.1	853.8	2			
100	4	U[1,60]	48.8	4.9	10	125.6	18.9	10	1,444	55.4	10	2,440	3088.9	3						
		U[30,60]	96.3	7.7	10	401.4	58.1	10	3,969	299.8	10	7,205	10379.8	4						
	8	U[1,60]	48.0	8.3	10	150.7	176.2	7	202	309.7	6	404	4753.0	0						
		U[30,60]	56.7	3.2	10	218.3	53.0	9	229	35.2	10	914	4620.7	1						
	12	U[1,60]	51.5	9.4	10	139.9	133.5	7	124	256.4	6	433	6179.3	1						
		U[30,60]	29.6	2.2	10	207.8	105.1	10	114	17.4	10	1,085	4008.1	4						

* Number of instances BAB outperforms CA (out of 10)

As can be observed from the tables, as n increases the number of efficient solutions increases, for all problem combinations. From Table 3.6, we can observe the increase in the average number of efficient solutions with increasing n. On average there are 13, 25, 38 and 59 efficient solutions, for problems having 40, 60, 80, and 100 jobs respectively. The difficulty of attaining an efficient solution increases considerably when n increases. In Table 3.4, we have two settings having the same average number of efficient solutions; n=60, m=8, wr_{ij}~U[1,60] and n=80, m=8, wr_{ij}~U[30,60]. The Classical algorithm generates the efficient set five times quicker for case 1 when compared to case 2. Similarly Branch and Bound algorithm generates the efficient set for the first case with three times of more computational effort, compared to the case of 60 jobs. This is due to the fact, the number of integer variables increases with an increase in n for classical approach. For Branch and Bound algorithm, the number of choices increases as a function of n.

As m increases, the ranges of F and WRJ decrease and that leads to a decrease in the number of efficient solutions. This behavior can be observed from Table 3.5, for the problems with 100 jobs, the average number of efficient solutions decrease with the increase in the number of machines, the average number of efficient solutions are 45, 26, and 13, for problems with 4, 8, and 12 machines, respectively. For fixed n, the performance of generating efficient set by the classical approach deteriorates with an increase in the number of efficient solutions. Note from Table 3.8 that where n=40, m=8, it takes 14 CPU seconds to generate 27 efficient solutions, however, time increases to 25 CPU seconds where 37 efficient solutions are generated. As m increases, the efficient solutions are generated in higher computational times, due to the increase in the number of integer decision variables, which is n^2m. Note that the same number of efficient solutions is generated in less effort when m is small. In Table 3.4, we can observe this effect significantly, for the problems with 80 jobs, and reassignment cost in range between 30 and 60, 10 efficient solutions exist on average for the cases with 8 and 12 machines. The Classical approach generates the efficient set in 48 CPU seconds on average when m=8, and in 95 CPU seconds on average when m=12. However, the performance of Branch and Bound algorithm (does not degrade) as m increases. Note that the number of levels of the Branch and Bound tree is $n + m - 1$, and is less

sensitive to m which increases in very small increments and which is small compared to n.

In general, the performance of the classical approach is dependent on the number of integer variables (which increases with n and m) and number of efficient solutions. The effects of other parameters, the disruption duration, processing time variability, and reassignment cost variability are not as dominant, as these parameters do not change the number of integer variables.

We observe that the disruption duration, processing time and reassignment cost distributions significantly affect the performance of the Branch and Bound algorithm. When the disruption duration is longer, the sequencing choices for the jobs are much more and this causes weak differentiation of the partial solutions which in turn increases the difficulty of attaining optimal solutions. This significant behavior can be easily observed when Tables 3.3 and 3.5 are compared. Note that the average CPU time of Branch and Bound algorithm to generate efficient set is equal 1.9 CPU seconds where the disruption duration is short (see Table 3.3). The CPU time increases to 43.8 seconds where the disruption duration is medium (see Table 3.5). Whenever the processing times are higher the disruption durations are longer and thus the problems are harder to solve.

When the variability of the processing times or reassignment costs decreases, the differentiation powers of the lower bounds decrease as the solutions become closer. As the power of the lower bounds directly affects the performance of the Branch and Bound algorithm, we observe smaller computational times when the ranges are wider. This relation is quite obvious from Table 3.5 the performance of the algorithm depends on reassignment cost variation. Note that when there are 100 jobs and 4 machines, the efficient set is generated in 55 seconds for low variation case, and in 300 seconds when the variation is high. Moreover, we observe more significant affect of the processing time variability, as the processing time defines the range of efficient solutions more often. One can point out some exceptions which can be attributed to the randomness effect like dominant contributions of few instances to average performance. As can be more clearly seen from our summary table, i.e., Table 3.9, the Branch and Bound algorithm outperforms classical approach in vast majority of the problem combinations, (1031 times in 1260

instances). The only exception is $D=L$ and p_{ij} ~ U[50,100] combination where performance of the classical approach is better (87 times in 120 instances).

We next analyze the performance of the algorithms used in finding an optimal solution for defined quadratic and quasi-chebyschev objective functions. Tables 3.10 through 3.15 report the maximum and average CPU times, and the number of nodes for the Branch and Bound algorithm. Specifically Tables 3.10 reports the statistics for short disruption durations and processing time distribution in between 1 and 100, for quadratic and quasi-chebyschev objective functions. Tables 3.11 provides the p_{ij}~U[50,100] counterpart of this table. Tables 3.12 through 3.15 are organized in a similar manner, and provide the results, for medium and long disruption durations, and processing time values within the ranges [1,100] and [50,100]. Tables 3.16 through 3.21 report the maximum and average CPU times and the percentage of the efficient solutions generated by the IP based algorithm. In a similar fashion to the results of Branch and Bound algorithm, Tables 3.16 and 3.17 report the results for the short disruption duration, and two distributions of the processing times. Tables 3.18 through 3.21 provide the results for the medium and long disruption duration cases in the same order. We also include summary tables, Tables 3.22 through 3.27, for the average CPU times of the Classical Approach, IP based algorithm and Branch and Bound algorithm that could be used to find an optimal solution for any nondecreasing function of F and WRJ. Tables 3.22, and 3.23 provide the results for the small disruption duration case, and for p_{ij}~U[1,100] and p_{ij}~U[50,100], respectively. Tables 3.24 through 3.27 summarize the statistics for medium and long disruption durations.

Table 3.10 Performance of the Branch and Bound Algorithm, $p_{ij} \sim U[1,100]$, $D = S$

n	m	$wrij$	Quadratic Function				Quasi-Chebychev Function			
			CPU time (sec)		# Nodes		CPU time (sec)		# Nodes	
			Avg	Max	Avg	Max	Avg	Max	Avg	Max
40	4	U[1,60]	0.0	0.1	141	613	0.0	0.1	165	527
		U[30,60]	0.0	0.1	427	794	0.0	0.1	327	769
	8	U[1,60]	0.0	0.1	288	677	0.1	0.2	288	1,017
		U[30,60]	0.0	0.0	48	406	0.0	0.0	42	348
	12	U[1,60]	0.0	0.1	176	765	0.0	0.1	140	470
		U[30,60]	0.0	0.1	148	1,406	0.0	0.2	183	1,771
60	4	U[1,60]	0.1	0.3	938	1,986	0.1	0.3	794	1,947
		U[30,60]	0.1	0.2	1,353	2,060	0.1	0.3	1,142	2,380
	8	U[1,60]	0.1	0.3	307	1,525	0.1	0.3	206	1,529
		U[30,60]	0.1	0.5	729	5,205	0.1	0.4	451	2,522
	12	U[1,60]	0.1	0.5	707	3,214	0.2	0.7	532	2,724
		U[30,60]	0.0	0.1	87	633	0.0	0.1	186	1,296
80	4	U[1,60]	0.3	0.5	1,228	2,756	0.3	0.7	1,434	2,636
		U[30,60]	0.7	2.0	4,044	12,696	0.5	1.5	3,145	9,452
	8	U[1,60]	0.4	1.0	1,385	3,367	0.4	1.3	1,498	5,520
		U[30,60]	0.3	0.8	1,805	5,134	0.3	0.9	1,342	5,356
	12	U[1,60]	0.3	0.9	692	1,956	0.4	1.2	718	2,012
		U[30,60]	0.6	3.3	2,328	8,046	0.7	3.4	2,218	8,599
100	4	U[1,60]	1.3	2.6	3,752	7,113	1.2	2.7	3,410	7,851
		U[30,60]	3.0	10.0	14,989	51,857	2.1	5.9	9,484	26,067
	8	U[1,60]	2.5	12.9	4,816	20,213	2.1	10.6	3,916	14,505
		U[30,60]	1.2	2.8	6,113	15,763	1.0	1.7	4,015	7,377
	12	U[1,60]	3.8	17.1	4,087	17,557	3.5	14.4	4,132	15,811
		U[30,60]	0.6	1.5	2,844	4,672	0.5	1.8	1,987	5,745

Table 3.11 Performance of the Branch and Bound Algorithm, $p_{ij} \sim$ U[50,100], $D = S$

n	m	$wrij$	Quadratic Function				Quasi-Chebychev Function			
			CPU time (sec)		# Nodes		CPU time (sec)		# Nodes	
			Avg	Max	Avg	Max	Avg	Max	Avg	Max
40	4	U[1,60]	0.0	0.1	166	498	0.0	0.1	206	721
		U[30,60]	0.1	0.1	311	816	0.0	0.1	208	827
	8	U[1,60]	0.1	0.1	313	677	0.1	0.1	263	510
		U[30,60]	0.1	0.3	479	1,139	0.1	0.4	491	1,900
	12	U[1,60]	0.0	0.0	11	58	0.0	0.0	15	100
		U[30,60]	0.0	0.1	64	542	0.0	0.2	61	545
60	4	U[1,60]	0.2	0.3	663	1,657	0.2	0.3	672	1,616
		U[30,60]	0.3	0.5	2,231	4,045	0.3	0.4	1,797	2,154
	8	U[1,60]	0.3	1.0	695	2,690	0.3	0.7	640	1,605
		U[30,60]	0.8	2.0	3,348	9,856	0.6	1.9	2,558	9,760
	12	U[1,60]	0.5	3.1	734	4,870	0.5	2.9	674	4,021
		U[30,60]	0.4	0.9	984	1,882	0.4	0.9	762	1,841
80	4	U[1,60]	0.6	1.3	1,744	4,846	0.5	1.0	1,505	3,895
		U[30,60]	2.5	4.7	15,468	34,156	1.9	6.4	12,093	51,704
	8	U[1,60]	1.4	3.7	2,666	8,141	1.1	2.0	2,014	3,851
		U[30,60]	1.4	4.6	3,679	12,821	1.2	2.9	2,805	6,221
	12	U[1,60]	2.5	9.7	4,003	21,695	1.9	5.2	2,635	10,279
		U[30,60]	3.3	19.4	7,019	43,710	2.3	13.0	5,183	32,879
100	4	U[1,60]	2.9	5.6	9,709	30,761	2.3	4.8	7,359	23,866
		U[30,60]	20.9	52.9	107,790	329,186	15.5	34.9	78,273	189,655
	8	U[1,60]	9.4	26.8	12,708	30,822	7.4	18.7	10,144	23,742
		U[30,60]	10.4	36.8	28,008	89,078	7.4	31.0	20,026	84,437
	12	U[1,60]	4.8	27.5	2,330	9,889	4.2	13.7	2,689	11,295
		U[30,60]	5.7	27.6	9,515	54,859	3.6	14.8	5,520	28,060

Table 3.12 Performance of the Branch and Bound Algorithm, $p_{ij} \sim U[1,100]$, $D = M$

n	m	$wrij$	Quadratic Function				Quasi-Chebychev Function			
			CPU time (sec)		# Nodes		CPU time (sec)		# Nodes	
			Avg	Max	Avg	Max	Avg	Max	Avg	Max
40	4	U[1,60]	0.1	0.3	761	3,647	0.1	0.3	618	2,704
		U[30,60]	0.1	0.2	1,079	2,685	0.1	0.2	823	2,121
	8	U[1,60]	0.1	0.1	534	1,089	0.1	0.2	565	1,401
		U[30,60]	0.0	0.1	354	1,390	0.0	0.1	271	1,359
	12	U[1,60]	0.1	0.2	342	972	0.1	0.3	306	1,232
		U[30,60]	0.1	0.2	506	2,513	0.1	0.3	595	3,741
60	4	U[1,60]	0.7	1.7	3,652	7,922	0.6	1.4	2,627	5,578
		U[30,60]	1.1	3.0	10,827	25,774	0.9	2.0	7,928	14,774
	8	U[1,60]	0.7	3.2	2,796	10,638	0.7	2.9	2,466	9,337
		U[30,60]	0.3	0.6	2,508	10,149	0.2	0.5	1,506	3,735
	12	U[1,60]	0.2	0.7	771	3,520	0.2	0.8	685	2,627
		U[30,60]	0.1	0.5	988	2,728	0.2	0.6	753	2,733
80	4	U[1,60]	3.0	13.5	10,629	35,004	2.3	8.4	8,050	20,105
		U[30,60]	8.7	33.2	66,251	309,302	6.1	24.0	41,147	185,348
	8	U[1,60]	1.4	6.0	4,065	18,101	1.3	5.9	3,350	16,998
		U[30,60]	2.5	8.6	17,045	62,000	2.4	9.5	15,140	60,631
	12	U[1,60]	2.1	7.3	3,499	6,884	2.1	8.9	3,239	12,549
		U[30,60]	3.0	7.0	17,425	30,990	2.9	9.9	13,751	40,111
100	4	U[1,60]	17.3	72.5	41,445	184,191	12.0	46.8	26,458	109,363
		U[30,60]	102.6	416.3	573,452	2,418,580	69.0	270.8	355,073	1,453,211
	8	U[1,60]	137.8	1012.9	166,588	1,196,745	93.6	628.7	104,540	669,231
		U[30,60]	11.5	31.4	57,876	175,493	9.4	25.4	43,852	126,763
	12	U[1,60]	102.5	798.4	75,680	528,087	89.0	710.5	65,874	465,659
		U[30,60]	3.8	11.6	19,291	71,219	4.1	11.2	18,359	57,819

Table 3.13 Performance of the Branch and Bound Algorithm, $p_{ij} \sim$ U[50,100], $D = M$

n	m	$wrij$	Quadratic Function				Quasi-Chebychev Function			
			CPU time (sec)		# Nodes		CPU time (sec)		# Nodes	
			Avg	Max	Avg	Max	Avg	Max	Avg	Max
40	4	U[1,60]	0.1	0.2	563	2,374	0.1	0.2	391	1,162
		U[30,60]	0.2	0.6	2,187	7,038	0.1	0.3	1,127	2,825
	8	U[1,60]	0.2	0.4	766	1,998	0.1	0.4	524	1,496
		U[30,60]	0.2	0.4	1,092	2,285	0.2	0.4	948	2,912
	12	U[1,60]	0.1	0.2	384	1,041	0.1	0.2	366	977
		U[30,60]	0.1	0.4	474	1,060	0.1	0.4	414	1,236
60	4	U[1,60]	0.9	3.1	6,601	24,801	0.7	2.4	4,368	15,652
		U[30,60]	9.4	27.6	112,168	333,945	4.9	13.0	52,698	149,543
	8	U[1,60]	1.6	6.4	4,386	13,481	1.6	4.9	4,610	13,580
		U[30,60]	12.7	36.2	80,985	237,469	6.6	18.6	39,605	104,538
	12	U[1,60]	1.1	3.7	1,773	8,448	1.4	5.4	2,167	9,011
		U[30,60]	0.8	1.4	2,279	4,546	0.7	1.8	1,874	6,731
80	4	U[1,60]	18.7	70.1	76,904	193,749	13.1	47.7	48,129	110,658
		U[30,60]	510.0	964.1	4,599,371	10,976,880	242.5	482.3	2,041,710	3,544,523
	8	U[1,60]	24.0	165.2	67,767	501,507	16.1	102.1	39,436	271,231
		U[30,60]	37.8	72.1	168,184	361,737	21.7	34.5	98,755	167,699
	12	U[1,60]	79.5	404.4	123,848	784,621	82.3	562.3	129,488	1,003,219
		U[30,60]	98.3	371.9	293,389	918,580	43.5	156.2	140,629	517,825
100	4	U[1,60]	564.0	1994.5	2,682,871	12,625,600	384.4	1324.7	1,553,432	7,301,662
		U[30,60]	5114.4	5400.0	50,710,566	82,936,065	4595.6	5400.0	38,251,353	58,676,260
	8	U[1,60]	373.6	2869.8	620,099	4,554,507	252.5	1796.8	379,192	2,486,018
		U[30,60]	1450.6	4978.9	5,341,417	18,661,348	444.7	1816.3	1,705,840	6,408,390
	12	U[1,60]	780.8	4710.4	878,035	5,807,128	793.4	4670.8	818,490	5,160,561
		U[30,60]	634.9	2497.6	1,636,775	6,548,700	203.4	773.7	603,587	2,556,812

Table 3.14 Performance of the Branch and Bound Algorithm, $p_{ij} \sim$ U[1,100], $D = L$

n	m	$wrij$	Quadratic Function				Quasi-Chebychev Function			
			CPU time (sec)		# Nodes		CPU time (sec)		# Nodes	
			Avg	Max	Avg	Max	Avg	Max	Avg	Max
40	4	U[1,60]	0.4	1.4	4,546	17,274	0.3	1.0	2,872	9,906
		U[30,60]	0.8	2.7	16,263	54,339	0.7	2.1	10,585	32,047
	8	U[1,60]	0.4	2.1	4,237	25,002	0.5	2.2	3,146	17,168
		U[30,60]	0.2	1.4	3,343	19,308	0.2	1.2	2,466	12,765
	12	U[1,60]	0.7	3.3	4,919	23,151	1.1	6.4	5,266	30,593
		U[30,60]	0.1	0.5	1,771	9,665	0.2	0.6	1,366	8,251
60	4	U[1,60]	20.0	79.9	111,496	331,462	15.2	57.9	70,253	199,284
		U[30,60]	120.4	616.4	1,289,880	6,074,665	77.7	359.1	699,024	3,014,404
	8	U[1,60]	340.4	3295.2	944,385	9,042,751	280.0	2710.5	593,184	5,641,007
		U[30,60]	3.7	8.9	53,264	178,362	2.8	6.3	28,851	68,734
	12	U[1,60]	8.4	34.0	37,087	175,661	9.5	52.6	31,476	188,359
		U[30,60]	0.8	2.9	7,648	37,716	1.0	3.3	7,170	30,229
80	4	U[1,60]	856.2	4794.0	3,303,311	14,398,547	568.8	3125.9	1,809,977	7,650,353
		U[30,60]	2464.4	5400.0	19,661,738	61,899,584	2112.6	5400.0	13,982,858	50,139,803
	8	U[1,60]	153.1	893.5	311,904	1,871,915	95.0	508.8	158,538	837,932
		U[30,60]	851.5	4888.8	5,741,512	36,028,091	513.2	2769.9	2,934,158	17,084,437
	12	U[1,60]	784.6	5400.0	1,474,004	10,665,006	750.7	5400.1	1,028,399	7,298,240
		U[30,60]	301.2	2365.8	1,845,108	13,440,357	250.9	1947.7	1,280,983	9,299,039

Table 3.15 Performance of the Branch and Bound Algorithm, $p_{ij} \sim$ U[50,100], $D = L$

n	m	$wrij$	Quadratic Function				Quasi-Chebychev Function			
			CPU time (sec)		# Nodes		CPU time (sec)		# Nodes	
			Avg	Max	Avg	Max	Avg	Max	Avg	Max
40	4	U[1,60]	2.7	9.3	33,538	128,851	2.1	6.9	21,470	74,843
		U[30,60]	14.3	35.7	256,294	624,546	7.0	10.2	111,658	164,301
	8	U[1,60]	3.3	8.7	21,698	67,931	2.4	6.6	12,638	35,691
		U[30,60]	5.8	16.6	67,895	183,161	2.5	7.6	25,116	81,964
	12	U[1,60]	1.8	6.2	11,088	52,499	2.0	8.0	8,982	45,644
		U[30,60]	3.8	24.3	23,790	109,688	1.5	7.1	9,048	35,929
60	4	U[1,60]	493.0	3600.0	3,608,929	24,921,959	414.8	3200.9	2,516,504	18,488,539
		U[30,60]	3426.7	3600.0	45,344,540	61,975,700	2797.1	3600.0	30,113,167	45,937,410
	8	U[1,60]	192.8	784.5	680,918	2,112,327	123.8	461.8	382,724	1,106,605
		U[30,60]	1717.3	3600.0	16,493,839	35,978,323	438.9	1158.5	4,047,710	10,853,027
	12	U[1,60]	65.5	259.6	161,521	942,256	49.2	267.8	92,081	353,858
		U[30,60]	65.1	128.7	358,525	627,365	19.9	30.9	104,826	181,207

Table 3.16 Performance of the IP Based Algorithm, $p_{ij} \sim$ U[1,100], $D=S$

n	m	$wrij$	Quadratic Function				Quasi-Chebychev Function			
			CPU time (sec)		% Generated		CPU time (sec)		% Generated	
			Avg	Max	Avg	Max	Avg	Max	Avg	Max
40	4	U[1,60]	0.5	1.1	60	75	0.6	1.2	63	75
		U[30,60]	0.6	1.3	56	75	0.6	1.0	53	75
	8	U[1,60]	0.9	1.2	51	67	0.9	1.2	51	67
		U[30,60]	0.9	1.0	27	67	0.9	1.0	27	67
	12	U[1,60]	1.5	2.0	44	75	1.6	2.4	44	75
		U[30,60]	2.1	4.7	25	80	1.8	3.4	25	80
60	4	U[1,60]	2.6	8.6	56	80	2.8	11.0	57	80
		U[30,60]	2.2	8.9	44	63	2.0	3.8	44	56
	8	U[1,60]	2.3	4.1	35	75	2.2	3.7	35	75
		U[30,60]	3.2	5.1	52	67	3.3	6.5	52	67
	12	U[1,60]	5.3	14.9	57	80	5.6	12.8	57	80
		U[30,60]	3.5	4.1	36	60	3.4	3.9	36	60
80	4	U[1,60]	5.5	14.5	52	83	5.6	10.3	53	86
		U[30,60]	4.3	9.5	37	45	7.4	13.5	40	56
	8	U[1,60]	6.4	12.0	63	80	8.8	24.9	67	83
		U[30,60]	7.5	29.2	51	67	10.0	29.9	56	75
	12	U[1,60]	7.8	13.4	52	80	9.6	16.3	54	80
		U[30,60]	6.8	8.6	39	75	6.9	9.6	41	75
100	4	U[1,60]	8.6	15.9	47	57	12.5	26.8	49	64
		U[30,60]	16.2	47.9	46	80	21.0	115.0	46	80
	8	U[1,60]	9.2	20.3	48	83	13.1	33.2	52	83
		U[30,60]	18.0	70.2	52	80	15.7	39.4	53	80
	12	U[1,60]	31.9	102.5	51	67	34.7	148.4	54	71
		U[30,60]	13.0	21.1	60	80	22.2	103.0	60	80

Table 3.17 Performance of the IP Based Algorithm, $p_{ij} \sim U[50,100]$, $D = S$

n	m	$wrij$	Quadratic Function				Quasi-Chebychev Function			
			CPU time (sec)		% Generated		CPU time (sec)		% Generated	
			Avg	Max	Avg	Max	Avg	Max	Avg	Max
40	4	U[1,60]	0.6	1.4	52	70	0.7	1.2	54	70
		U[30,60]	0.6	1.2	59	75	0.6	1.0	60	75
	8	U[1,60]	1.2	3.2	63	75	1.2	3.1	63	75
		U[30,60]	1.6	4.6	51	83	1.8	4.9	51	83
	12	U[1,60]	2.7	4.2	21	75	2.6	3.9	21	75
		U[30,60]	2.8	4.6	20	83	3.0	4.6	20	83
60	4	U[1,60]	1.9	6.5	48	71	1.9	4.7	50	71
		U[30,60]	3.9	12.8	47	67	5.2	12.3	52	70
	8	U[1,60]	4.4	12.6	56	90	4.5	12.5	59	80
		U[30,60]	3.7	10.8	40	57	5.1	8.7	42	57
	12	U[1,60]	6.7	18.0	59	75	6.6	14.1	60	75
		U[30,60]	13.5	49.6	62	80	13.8	52.4	57	80
80	4	U[1,60]	4.3	13.6	44	60	5.9	22.3	44	55
		U[30,60]	12.2	27.1	41	62	12.0	28.9	40	54
	8	U[1,60]	9.7	26.6	42	53	12.7	26.9	45	57
		U[30,60]	13.8	73.6	44	75	14.1	40.7	46	60
	12	U[1,60]	12.4	34.4	55	75	13.2	30.4	56	75
		U[30,60]	17.4	55.6	46	75	21.7	72.2	47	75
100	4	U[1,60]	9.4	20.2	40	58	15.3	37.7	44	58
		U[30,60]	28.6	103.5	26	42	44.7	123.9	30	42
	8	U[1,60]	17.2	33.8	46	80	33.4	88.1	50	80
		U[30,60]	14.2	32.4	40	80	34.0	71.2	43	80
	12	U[1,60]	18.0	48.2	47	67	23.2	51.7	51	67
		U[30,60]	16.1	27.7	37	50	33.8	161.5	44	57

Table 3.18 Performance of the IP Based Algorithm, $p_{ij} \sim$ U[1,100], $D = M$

n	m	$wrij$	Quadratic Function				Quasi-Chebychev Function			
			CPU time (sec)		% Generated		CPU time (sec)		% Generated	
			Avg	Max	Avg	Max	Avg	Max	Avg	Max
40	4	U[1,60]	0.6	1.9	58	100	0.8	2.9	59	100
		U[30,60]	0.5	1.6	61	83	0.6	2.3	63	83
	8	U[1,60]	0.8	1.7	84	100	0.7	1.4	84	100
		U[30,60]	0.7	1.5	75	100	0.8	1.5	77	100
	12	U[1,60]	1.0	1.9	64	100	1.2	3.1	64	100
		U[30,60]	1.2	2.0	86	100	1.3	1.9	86	100
60	4	U[1,60]	2.7	7.8	43	53	4.2	8.5	44	53
		U[30,60]	3.0	9.5	35	47	8.0	36.2	38	53
	8	U[1,60]	2.0	6.4	68	100	3.6	14.3	74	100
		U[30,60]	2.5	7.0	65	100	2.8	7.5	67	100
	12	U[1,60]	3.3	8.2	92	100	3.4	8.4	92	100
		U[30,60]	3.1	6.7	88	100	3.3	6.9	88	100
80	4	U[1,60]	6.9	16.0	42	52	8.5	22.0	42	57
		U[30,60]	8.6	44.3	28	47	20.0	58.5	32	53
	8	U[1,60]	5.7	20.8	59	80	6.4	17.3	60	80
		U[30,60]	14.5	41.3	53	75	19.0	39.5	57	83
	12	U[1,60]	8.6	38.4	79	100	12.7	39.5	80	100
		U[30,60]	9.4	22.9	59	100	10.4	25.0	64	100
100	4	U[1,60]	30.2	221.3	33	43	53.2	238.4	34	43
		U[30,60]	102.5	505.6	30	58	643.0	2442.6	32	64
	8	U[1,60]	27.8	63.3	36	45	52.3	201.6	41	50
		U[30,60]	18.8	41.9	36	50	23.9	67.0	37	50
	12	U[1,60]	24.8	55.4	51	70	35.8	122.1	52	70
		U[30,60]	32.1	123.7	55	75	33.5	124.6	56	75

Table 3.19 Performance of the IP Based Algorithm, $p_{ij} \sim$ U[50,100], $D = M$

n	m	$wrij$	Quadratic Function				Quasi-Chebychev Function			
			CPU time (sec)		% Generated		CPU time (sec)		% Generated	
			Avg	Max	Avg	Max	Avg	Max	Avg	Max
40	4	U[1,60]	0.5	1.7	53	86	0.7	3.0	55	86
		U[30,60]	0.8	2.0	48	64	1.2	4.1	50	79
	8	U[1,60]	1.2	3.1	58	75	1.1	2.7	57	73
		U[30,60]	1.8	4.3	52	78	1.9	5.0	51	78
	12	U[1,60]	1.6	6.5	76	100	2.0	7.0	79	100
		U[30,60]	1.6	4.6	62	100	2.2	5.2	73	100
60	4	U[1,60]	1.7	4.3	41	58	4.5	24.0	42	58
		U[30,60]	2.8	7.9	30	56	6.3	17.8	32	50
	8	U[1,60]	4.5	10.1	54	75	6.6	14.1	56	73
		U[30,60]	12.3	32.7	44	89	15.8	43.6	43	89
	12	U[1,60]	4.0	7.0	55	78	6.5	17.9	58	78
		U[30,60]	4.4	11.5	41	70	5.5	15.4	42	70
80	4	U[1,60]	14.2	51.9	38	48	18.5	56.4	39	53
		U[30,60]	32.6	214.8	25	37	36.8	186.4	25	33
	8	U[1,60]	5.8	8.7	39	50	9.8	27.2	40	50
		U[30,60]	22.9	57.1	35	48	34.7	85.7	37	48
	12	U[1,60]	12.1	31.8	44	73	16.8	35.0	46	73
		U[30,60]	36.8	94.0	46	63	51.9	126.7	46	69
100	4	U[1,60]	24.7	102.3	29	45	40.5	157.2	30	51
		U[30,60]	36.2	101.5	19	28	70.9	265.6	21	44
	8	U[1,60]	15.4	20.8	33	46	35.0	74.9	35	54
		U[30,60]	71.3	231.9	29	42	58.1	184.4	29	50
	12	U[1,60]	30.3	83.0	37	49	60.3	249.8	39	54
		U[30,60]	36.7	138.3	28	41	64.8	252.7	30	52

Table 3.20 Performance of the IP Based Algorithm, $p_{ij} \sim$ U[1,100], D=L

n	m	$wrij$	Quadratic Function				Quasi-Chebychev Function			
			CPU time (sec)		% Generated		CPU time (sec)		% Generated	
			Avg	Max	Avg	Max	Avg	Max	Avg	Max
40	4	U[1,60]	0.9	2.8	41	67	0.8	2.1	41	67
		U[30,60]	0.9	2.0	32	53	2.2	4.6	37	65
	8	U[1,60]	1.3	3.5	58	100	1.7	2.8	61	100
		U[30,60]	1.2	3.0	65	100	1.7	6.9	69	100
	12	U[1,60]	1.7	4.5	67	100	2.2	7.3	68	100
		U[30,60]	1.5	2.8	72	100	1.5	3.4	73	100
60	4	U[1,60]	10.1	42.1	29	41	25.9	82.1	31	46
		U[30,60]	77.5	696.4	20	31	7.3	21.9	20	26
	8	U[1,60]	5.0	8.8	50	83	7.7	18.0	50	83
		U[30,60]	4.3	10.1	36	44	6.9	13.8	39	50
	12	U[1,60]	4.5	9.4	57	100	6.3	12.1	56	88
		U[30,60]	4.0	8.4	61	75	5.9	17.0	61	75
80	4	U[1,60]	18.2	60.5	23	29	242.8	818.7	25	33
		U[30,60]	706.3	5400.0	17	21	87.1	414.0	17	23
	8	U[1,60]	9.2	20.1	39	58	11.0	22.7	40	58
		U[30,60]	21.5	73.1	34	45	13.5	41.3	33	45
	12	U[1,60]	25.5	83.6	47	83	24.6	59.1	48	83
		U[30,60]	16.1	41.2	47	67	19.8	47.8	47	67

Table 3.21 Performance of the IP Based Algorithm, $p_{ij} \sim$ U[50,100], D=L

n	m	$wrij$	Quadratic Function				Quasi-Chebychev Function			
			CPU time (sec)		% Generated		CPU time (sec)		% Generated	
			Avg	Max	Avg	Max	Avg	Max	Avg	Max
40	4	U[1,60]	1.4	2.8	40	74	2.2	5.1	40	63
		U[30,60]	2.3	5.7	30	39	3.7	7.2	32	39
	8	U[1,60]	1.3	3.1	47	56	2.5	4.5	50	67
		U[30,60]	3.0	8.3	35	46	4.0	17.8	36	56
	12	U[1,60]	1.5	4.6	48	62	2.6	6.6	52	62
		U[30,60]	1.7	3.9	51	88	3.4	13.4	55	88
60	4	U[1,60]	3.2	9.4	27	38	4.9	9.8	28	38
		U[30,60]	6.4	24.3	17	23	22.5	72.4	20	25
	8	U[1,60]	9.5	32.0	38	49	19.9	56.7	42	51
		U[30,60]	11.9	34.7	27	38	12.7	38.8	27	44
	12	U[1,60]	5.5	13.1	44	56	8.5	20.1	45	56
		U[30,60]	5.1	12.7	28	34	14.8	39.5	33	47
80	4	U[1,60]	10.7	28.8	23	30	53.4	214.3	25	32
		U[30,60]	211.6	1927.7	13	16	599.6	5330.5	14	18
	8	U[1,60]	21.0	48.3	32	46	22.3	49.0	32	46
		U[30,60]	29.9	171.8	19	25	72.6	173.7	22	27
	12	U[1,60]	26.4	53.1	29	38	34.9	70.0	30	38
		U[30,60]	56.2	224.1	27	35	46.1	170.0	26	39

Table 3.22 Comparison of Average Performances of Optimization Algorithms, p_{ij} ~U[1,100], D=S

n	m	$wrij$	Classic Approach	Average CPU Times (sec)					
				Quadratic Function			Quasi-Chebychev Function		
				IPB	BAB	CNT	IPB	BAB	CNT
40	4	U[1,60]	0.6	0.5	0.0	10	0.6	0.0	10
		U[30,60]	0.7	0.6	0.0	10	0.6	0.0	10
	8	U[1,60]	0.7	0.9	0.0	10	0.9	0.1	10
		U[30,60]	0.4	0.9	0.0	10	0.9	0.0	10
	12	U[1,60]	0.8	1.5	0.0	10	1.6	0.0	10
		U[30,60]	0.8	2.1	0.0	10	1.8	0.0	10
60	4	U[1,60]	3.4	2.6	0.1	10	2.8	0.1	10
		U[30,60]	4.6	2.2	0.1	10	2.0	0.1	10
	8	U[1,60]	1.6	2.3	0.1	10	2.2	0.1	10
		U[30,60]	3.8	3.2	0.1	10	3.3	0.1	10
	12	U[1,60]	3.2	5.3	0.1	10	5.6	0.2	10
		U[30,60]	1.7	3.5	0.0	10	3.4	0.0	10
80	4	U[1,60]	11.1	5.5	0.3	10	5.6	0.3	10
		U[30,60]	36.2	4.3	0.7	10	7.4	0.5	10
	8	U[1,60]	13.5	6.4	0.4	10	8.8	0.4	10
		U[30,60]	15.8	7.5	0.3	10	10.0	0.3	10
	12	U[1,60]	13.2	7.8	0.3	10	9.6	0.4	10
		U[30,60]	21.9	6.8	0.6	10	6.9	0.7	10
100	4	U[1,60]	48.8	8.6	1.3	10	12.5	1.2	10
		U[30,60]	96.3	16.2	3.0	10	21.0	2.1	10
	8	U[1,60]	48.0	9.2	2.5	9	13.1	2.1	10
		U[30,60]	56.7	18.0	1.2	10	15.7	1.0	10
	12	U[1,60]	51.5	31.9	3.8	10	34.7	3.5	10
		U[30,60]	29.6	13.0	0.6	10	22.2	0.5	10

Table 3.23 Comparison of Average Performances of Optimization Algorithms, p_{ij} ~U[50,100], $D=S$

n	m	$wrij$	Classic Approach	Average CPU Times (sec)					
				Quadratic Function			Quasi-Chebychev Function		
				IPB	BAB	CNT	IPB	BAB	CNT
40	4	U[1,60]	1.4	0.6	0.0	10	0.7	0.0	10
		U[30,60]	1.1	0.6	0.1	10	0.6	0.0	10
	8	U[1,60]	1.8	1.2	0.1	10	1.2	0.1	10
		U[30,60]	2.2	1.6	0.1	10	1.8	0.1	10
	12	U[1,60]	0.3	2.7	0.0	10	2.6	0.0	10
		U[30,60]	0.9	2.8	0.0	10	3.0	0.0	10
60	4	U[1,60]	6.5	1.9	0.2	10	1.9	0.2	10
		U[30,60]	9.7	3.9	0.3	10	5.2	0.3	10
	8	U[1,60]	9.1	4.4	0.3	10	4.5	0.3	10
		U[30,60]	24.5	3.7	0.8	10	5.1	0.6	10
	12	U[1,60]	8.0	6.7	0.5	10	6.6	0.5	10
		U[30,60]	14.3	13.5	0.4	10	13.8	0.4	10
80	4	U[1,60]	30.2	4.3	0.6	10	5.9	0.5	10
		U[30,60]	54.2	12.2	2.5	9	12.0	1.9	10
	8	U[1,60]	44.6	9.7	1.4	10	12.7	1.1	10
		U[30,60]	48.7	13.8	1.4	10	14.1	1.2	10
	12	U[1,60]	52.7	12.4	2.5	10	13.2	1.9	10
		U[30,60]	95.2	17.4	3.3	9	21.7	2.3	9
100	4	U[1,60]	125.6	9.4	2.9	10	15.3	2.3	10
		U[30,60]	401.4	28.6	20.9	6	44.7	15.5	8
	8	U[1,60]	150.7	17.2	9.4	8	33.4	7.4	9
		U[30,60]	218.3	14.2	10.4	7	34.0	7.4	10
	12	U[1,60]	139.9	18.0	4.8	9	23.2	4.2	10
		U[30,60]	207.8	16.1	5.7	9	33.8	3.6	10

Table 3.24 Comparison of Average Performances of Optimization Algorithms, p_{ij} ~U[1,100], $D=M$

n	m	$wrij$	Classic Approach	Average CPU Times (sec)					
				Quadratic Function			Quasi-Chebychev Function		
				IPB	BAB	CNT	IPB	BAB	CNT
40	4	U[1,60]	1.9	0.6	0.1	10	0.8	0.1	10
		U[30,60]	1.8	0.5	0.1	10	0.6	0.1	10
	8	U[1,60]	1.3	0.8	0.1	10	0.7	0.1	10
		U[30,60]	0.9	0.7	0.0	10	0.8	0.0	10
	12	U[1,60]	2.1	1.0	0.1	10	1.2	0.1	10
		U[30,60]	1.3	1.2	0.1	10	1.3	0.1	10
60	4	U[1,60]	18.5	2.7	0.7	10	4.2	0.6	10
		U[30,60]	34.7	3.0	1.1	6	8.0	0.9	10
	8	U[1,60]	8.9	2.0	0.7	9	3.6	0.7	9
		U[30,60]	7.6	2.5	0.3	10	2.8	0.2	10
	12	U[1,60]	5.4	3.3	0.2	10	3.4	0.2	10
		U[30,60]	4.0	3.1	0.1	10	3.3	0.2	10
80	4	U[1,60]	136.4	6.9	3.0	9	8.5	2.3	9
		U[30,60]	1008.5	8.6	8.7	4	20.0	6.1	9
	8	U[1,60]	26.0	5.7	1.4	9	6.4	1.3	9
		U[30,60]	42.8	14.5	2.5	9	19.0	2.4	9
	12	U[1,60]	24.3	8.6	2.1	9	12.7	2.1	10
		U[30,60]	56.5	9.4	3.0	10	10.4	2.9	10
100	4	U[1,60]	1443.6	30.2	17.3	5	53.2	12.0	6
		U[30,60]	3968.7	102.5	102.6	2	643.0	69.0	6
	8	U[1,60]	202.3	27.8	137.8	5	52.3	93.6	7
		U[30,60]	228.8	18.8	11.5	9	23.9	9.4	9
	12	U[1,60]	124.4	24.8	102.5	6	35.8	89.0	7
		U[30,60]	114.2	32.1	3.8	10	33.5	4.1	10

Table 3.25 Comparison of Average Performances of Optimization Algorithms, p_{ij} ~U[50,100], $D=M$

n	m	$wrij$	Classic Approach	Average CPU Times (sec)					
				Quadratic Function			Quasi-Chebychev Function		
				IPB	BAB	CNT	IPB	BAB	CNT
40	4	U[1,60]	4.5	0.5	0.1	10	0.7	0.1	10
		U[30,60]	4.4	0.8	0.2	10	1.2	0.1	10
	8	U[1,60]	4.8	1.2	0.2	10	1.1	0.1	10
		U[30,60]	6.2	1.8	0.2	10	1.9	0.2	10
	12	U[1,60]	2.3	1.6	0.1	10	2.0	0.1	10
		U[30,60]	6.4	1.6	0.1	10	2.2	0.1	10
60	4	U[1,60]	30.9	1.7	0.9	8	4.5	0.7	10
		U[30,60]	68.8	2.8	9.4	1	6.3	4.9	4
	8	U[1,60]	30.8	4.5	1.6	9	6.6	1.6	10
		U[30,60]	80.1	12.3	12.7	6	15.8	6.6	6
	12	U[1,60]	28.3	4.0	1.1	10	6.5	1.4	10
		U[30,60]	49.4	4.4	0.8	10	5.5	0.7	10
80	4	U[1,60]	488.5	14.2	18.7	7	18.5	13.1	7
		U[30,60]	694.4	32.6	510.0	0	36.8	242.5	0
	8	U[1,60]	140.2	5.8	24.0	4	9.8	16.1	7
		U[30,60]	235.8	22.9	37.8	3	34.7	21.7	8
	12	U[1,60]	158.7	12.1	79.5	5	16.8	82.3	5
		U[30,60]	274.1	36.8	98.3	4	51.9	43.5	5
100	4	U[1,60]	2439.5	24.7	564.0	0	40.5	384.4	1
		U[30,60]	7205.4	36.2	5114.4	0	70.9	4595.6	0
	8	U[1,60]	404.2	15.4	373.6	4	35.0	252.5	5
		U[30,60]	913.6	71.3	1450.6	0	58.1	444.7	2
	12	U[1,60]	432.5	30.3	780.8	2	60.3	793.4	2
		U[30,60]	1084.5	36.7	634.9	0	64.8	203.4	3

Table 3.26 Comparison of Average Performances of Optimization Algorithms, p_{ij} ~U[1,100], $D=L$

n	m	$wrij$	Classic Approach	Average CPU Times (sec)					
				Quadratic Function			Quasi-Chebychev Function		
				IPB	BAB	CNT	IPB	BAB	CNT
40	4	U[1,60]	8.0	0.9	0.4	7	0.8	0.3	9
		U[30,60]	18.1	0.9	0.8	7	2.2	0.7	7
	8	U[1,60]	4.4	1.3	0.4	10	1.7	0.5	10
		U[30,60]	3.7	1.2	0.2	9	1.7	0.2	10
	12	U[1,60]	5.6	1.7	0.7	10	2.2	1.1	10
		U[30,60]	3.7	1.5	0.1	10	1.5	0.2	10
60	4	U[1,60]	138.0	10.1	20.0	2	25.9	15.2	7
		U[30,60]	1952.2	77.5	120.4	2	7.3	77.7	1
	8	U[1,60]	37.3	5.0	340.4	7	7.7	280.0	6
		U[30,60]	37.8	4.3	3.7	5	6.9	2.8	7
	12	U[1,60]	33.7	4.5	8.4	5	6.3	9.5	6
		U[30,60]	21.6	4.0	0.8	10	5.9	1.0	10
80	4	U[1,60]	3331.5	18.2	856.2	1	242.8	568.8	4
		U[30,60]	5396.2	706.3	2464.4	2	87.1	2112.6	0
	8	U[1,60]	117.3	9.2	153.1	3	11.0	95.0	6
		U[30,60]	221.2	21.5	851.5	2	13.5	513.2	4
	12	U[1,60]	134.7	25.5	784.6	5	24.6	750.7	5
		U[30,60]	153.1	16.1	301.2	2	19.8	250.9	2

Table 3.27 Comparison of Average Performances of Optimization Algorithms, p_{ij} ~U[50,100], $D=L$

n	m	$wrij$	Classic Approach	Average CPU Times (sec)					
				Quadratic Function			Quasi-Chebychev Function		
				IPB	BAB	CNT	IPB	BAB	CNT
40	4	U[1,60]	20.4	1.4	2.7	6	2.2	2.1	7
		U[30,60]	31.2	2.3	14.3	0	3.7	7.0	1
	8	U[1,60]	13.5	1.3	3.3	5	2.5	2.4	5
		U[30,60]	25.2	3.0	5.8	2	4.0	2.5	6
	12	U[1,60]	11.1	1.5	1.8	7	2.6	2.0	8
		U[30,60]	21.1	1.7	3.8	6	3.4	1.5	8
60	4	U[1,60]	506.2	3.2	493.0	0	4.9	414.8	0
		U[30,60]	1402.3	6.4	3426.7	0	22.5	2797.1	0
	8	U[1,60]	84.1	9.5	192.8	3	19.9	123.8	3
		U[30,60]	208.9	11.9	1717.3	0	12.7	438.9	0
	12	U[1,60]	82.7	5.5	65.5	3	8.5	49.2	3
		U[30,60]	154.8	5.1	65.1	1	14.8	19.9	4

When IPB and Branch and Bound algorithms are compared, no consistent dominance of one algorithm over the other can be observed, we find that, BAB is better than IPB for 1943 times in 2520 problem instances solved. We see that the performances of both algorithms are highly dependent on the number of efficient solutions. The performance of IPB algorithm is more significantly dependent on the number of integer variables that increase with the number of jobs and the number of machines. There are some exceptions where the performance deteriorates with decreasing m which can be attributed to the superior performance of the integer programs that have many integer variables in their linear programming relaxations. The IPB algorithm performs better for quadratic function compared to the quasi-chebyschev function. We can compare the average CPU times from Table 3.21, where the statistics for the IPB algorithm, for, p_{ij}~U[50,100], and $D=L$ are reported, for quadratic and quasi-chebyschev functions. We observe from the tables that the average CPU times for quadratic function is smaller than those of quasi-chebyschev function. This behavior can be attributed to the fact that in quasi-chebyschev function case, very few optimal solutions are in extreme supported solution set.

As can be observed from the tables the disruption duration, the processing time and reassignment cost distributions significantly affect the performance of the Branch and Bound algorithm. When the disruption duration is longer, the differentiation power of the partial solutions are weaker due to the existence of more sequencing alternatives. This adds to the complexity of the algorithm. This behavior can be observed from all tables. When the processing times are between 50 and 100, the disruption durations are longer, thus the CPU times are higher. When the processing times and reassignment costs are less variable, the solutions are more closer and hence their differentiation is harder. As a result, we observe the highest CPU times whenever the disruption duration are long, the processing times are in range [50,100] and the reassignment costs are in range [30,60] (see Table 3.15). The lowest CPU times are observed when the disruption duration is short, processing times are in range [1,100] and reassignment costs are in range [1,60] (see Table 3.10).

Quasi-chebyschev utility function usually leads to quicker solutions than quadratic utility function. Note from Table 3.13, that the average CPU times are smaller for quadratic function case compared to quasi-chebyschev function. This can be attributed to the fact that the partial solutions are not very close to each other, which increases the power of differentiation.

In general, for all parameter combinations and both objective function types, the IPB algorithm finds the optimal schedule by generating a small percentage of all efficient solutions. The higher percentages are associated to the cases with smaller number of efficient solutions, where the number of efficient solutions visited is also very small, and most of the solutions are extreme supported, which are generated at the initial step of the algorithm. Note that when n=80, the percentages are lower, as the number of efficient solutions is higher, and there exist many non-extreme supported, and unsupported solutions. The parameter effects on the performance of the Integer Programming Based algorithm is not as dominant as those of the Branch and Bound algorithm. We can conclude from the tables that the behavior of IPB algorithm is more consistent. Note that the worst performances of IPB algorithm are closer to their average counterparts when compared with those of BAB performances. The results on all tables reveal that both algorithms solve all

instances in much smaller CPU time than that of spent in generating all efficient solutions by classical approach.

We finally find the percentage of nodes evaluated till reaching the optimal solution and report the average case results. Table 3.28 reports the ratio of the optimality node to the total number of evaluated nodes.

Table 3.28 Optimality Node Percentages

n	*m*	*wrij*	Optimal Node/Total Node*100 (average)											
			p_{ij}~U[1,100]						p_{ij}~U[50,100]					
			D=S		*D=M*		*D=L*		*D=S*		*D=M*		*D=L*	
			Quad.	Q.-Cheb.	Quad.	Q.-Cheb.	Quad.	Q.-Cheb.	Quad.	Q.-Cheb.	Quad.	Q.-Cheb.	Quad.	Q.-Cheb.
40	4	U[1,60]	0.17	0.17	0.00	7.01	0.00	0.72	0.00	9.34	0.00	0.00	0.00	0.00
		U[30,60]	9.25	10.16	0.00	0.00	0.00	7.90	4.77	5.00	0.00	18.76	6.30	13.99
	8	U[1,60]	0.00	0.00	0.00	0.00	0.00	2.33	9.80	9.79	0.00	0.00	0.00	10.85
		U[30,60]	0.00	0.00	0.00	0.39	0.00	2.12	0.00	9.97	7.99	26.77	5.49	12.82
	12	U[1,60]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.24	0.00	0.00	1.31	2.01
		U[30,60]	0.00	0.00	0.00	0.00	0.00	0.00	7.54	17.63	9.15	29.01	7.02	9.38
60	4	U[1,60]	0.00	0.00	0.00	0.00	0.00	0.41	0.00	9.17	0.00	10.45	0.00	0.00
		U[30,60]	9.51	18.59	0.00	3.02	0.38	0.43	0.00	37.86	0.00	7.76	1.73	2.52
	8	U[1,60]	0.00	0.00	0.00	0.00	0.00	1.02	0.00	6.72	0.00	11.71	0.00	0.82
		U[30,60]	0.44	10.46	7.12	25.01	0.00	9.28	0.00	19.98	0.35	11.10	9.17	0.47
	12	U[1,60]	0.00	0.00	0.00	0.00	0.00	0.24	9.77	3.22	4.39	0.68	0.00	0.00
		U[30,60]	0.00	0.00	0.00	0.00	0.00	2.30	28.41	31.57	0.00	12.66	2.18	8.66
80	4	U[1,60]	0.00	0.00	0.00	2.02	0.00	1.06	0.00	0.00	0.00	1.10		
		U[30,60]	0.00	9.38	0.00	2.43	1.39	0.87	10.26	7.96	0.00	5.12		
	8	U[1,60]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00		
		U[30,60]	5.10	4.20	4.75	7.93	0.00	12.19	13.39	15.07	0.39	13.33		
	12	U[1,60]	0.00	5.79	0.00	0.00	0.00	0.00	0.00	0.00	0.00	12.11		
		U[30,60]	0.00	5.56	1.05	7.32	4.44	6.30	9.95	9.94	0.91	15.38		
100	4	U[1,60]	0.00	0.00	0.00	3.08			6.28	8.22	0.28	2.16		
		U[30,60]	0.00	0.00	0.00	3.39			8.30	17.83	5.84	8.44		
	8	U[1,60]	0.00	0.00	0.00	14.63			0.00	5.03	0.00	1.55		
		U[30,60]	0.00	11.48	0.00	6.18			0.00	24.19	0.00	5.84		
	12	U[1,60]	0.00	0.00	0.00	0.00			0.00	8.12	0.00	4.98		
		U[30,60]	0.00	0.56	0.00	6.81			0.00	1.76	0.00	0.00		

As can be observed from the tables in majority of the problem combinations the optimal solutions are found at the very early nodes of the search. Note that on average, the majority of the optimal solutions is found before searching 10 percent of all nodes. Moreover in many combinations we observe a ratio value zero, indicating the optimality of the initial solution. Note that the average percentages are higher when p_{ij}~U[50,100]. Due to the fact that, the lower bounds are weaker and hence give less reliable guide. For the case where p_{ij}~U[1,100] the averages are lower. This leads us to conclude that the lower bounds are good estimators of the optimal solutions and hence guide right solution paths. The solutions that are found at a preset termination limit are likely to be optimal or near optimal and hence a truncated Branch and Bound algorithm that terminates after a preset CPU time or number of node evaluations, can be a powerful alternative if the decision maker is interested with a near optimal, but not necessarily an optimal, solution.

CHAPTER 4

CONCLUSION AND FUTURE RESEARCH DIRECTIONS

This study considers bicriteria approaches to the minimum cost network flow problem and a rescheduling problem where those approaches find their applications.

For the bicriteria integer minimum cost network flow problem, we generate all efficient solutions in two phases. The first phase generates the extreme supported efficient points that are the extreme points of the objective space of the continuous BCNF problem. Our generation method differs from previous methods that are based on parametric analysis, in the sense that the efficient set is generated each time moving to the next adjacent point. Hence this phase may be preferred for the continuous BCNF problem if the decision maker is more interested with a specified portion of the objective space. In the second phase, we generate the other efficient points, i.e., nonextreme supported efficient points, unsupported efficient points by Integer Programming Based approaches.

Our rescheduling problem assumes parallel unrelated machine environments. The criteria are the total flow time as an efficiency measure and the total reassignment cost as a stability measure. We show that any linear combination of the two criteria can be represented by a bicriteria minimum cost integer network flow model (BCINF). Hence we use the results derived for the BCINF problem to tackle with our rescheduling problem.

In our rescheduling studies, we provide polynomial time algorithms to solve the hierarchical optimization problems. To generate all extreme supported efficient solutions, we use LP-based approach using slack pivoting and a weighted approach that are based on optimal assignment solutions. To generate all efficient solutions we propose two approaches. The first approach is an Integer Programming based and uses optimal solutions of the singly-constrained assignment problem. The second

approach implicitly enumerates all efficient solutions by Branch and Bound approach. Our Branch and Bound algorithm uses lower bounds on the total flow time and total reassignment cost. To find an initial set of approximate efficient solutions, we generate extreme supported efficient set by weighted approach and extend the set in a defined neighborhood.

The results of our computational tests have revealed that our Branch and Bound algorithm is superior than the classical approach, for majority of the test problems.

We use the same branching scheme and same bounds to minimize a composite function of the total flow time and total reassignment cost. The results of our computational tests have revealed that the Branch and Bound algorithm can solve problems up to 100 jobs and 12 machines. We also propose an algorithm that is based on Integer Programming. The algorithm eliminates a portion of the solution set that cannot reside an improved objective function value. The IP based algorithm also performs quite satisfactory and generates only a small portion of all efficient solutions.

The models we have studied represent growth in the network flow and rescheduling areas. There are many further research directions most of noteworthy of which are discussed below:

1) Addressing a stochastic version of the problem where the maintenance duration is not known with certainty.

2) Addressing a Tricriteria integer minimum cost network flow (TCINF) problem

Let $f_1(x)$, $f_2(x)$ and $f_3(x)$ be tricriteria. Finding optimal solutions to the following problems may be of help in developing the solution procedures for generating all efficient solutions and minimizing a composite function of tricriteria.

a) Unconstrained problems

i. Min $f_i(x) + \varepsilon_j f_j(x)$ $\quad i=1, 2, 3 \quad j=1, 2, 3 \quad i \neq j$

for appropriately selected values of ε_j.

ii. Min $f_i(x) + \varepsilon_j f_j(x) + \varepsilon_k f_k(x)$

$i=1, 2, 3 \quad j=1, 2, 3 \quad k=1, 2, 3 \quad i \neq j \; j \neq k \; i \neq k$

for appropriately selected values of ε_j, ε_k.

b) Constrained Problems

i. Min $f_i(x) + \varepsilon_j f_j(x) + \varepsilon_k f_k(x)$

s.t. $f_j(x) \leq b_j$

$i=1, 2, 3 \quad j=1, 2, 3 \quad k=1, 2, 3 \quad i \neq j \; j \neq k \; i \neq k$

for appropriately selected values of ε_j, ε_k.

ii. Min $f_i(x) + \varepsilon_j f_j(x) + \varepsilon_k f_k(x)$

s.t. $f_j(x) \leq b_j$

$f_k(x) \leq b_k$

$i=1, 2, 3 \quad j=1, 2, 3 \quad k=1, 2, 3 \quad i \neq j \; j \neq k \; i \neq k$

for appropriately selected values of ε_j, ε_k.

3) A Tricriteria rescheduling problem exploiting network flow structures

Once we set the criteria to total flow time, number of reassigned jobs, and total reassignment cost, the problem can be represented as a Tricriteria MCNF model., and hence the approaches derived for the TCINF problem can be used. Moreover by recognizing the special structures of rescheduling, hence scheduling, problems efficient enumeration schemes can be designed. We can benefit from the branching structure designed for our two criteria rescheduling problem as the decisions do not differ with an increase in the number of criteria.

4) Bicriteria or Tricriteria problems with different efficiency and/or stability measures

In this study, we consider total flow time as an efficiency measure. When the jobs do have different priorities, a more suitable objective would be to minimize total weighted flow time. The incorporation of the weights destroy the network flow nature of the model, so the properties and procedures extended from network flows would not be of help. However total weighted flow time has also a very nice property that the optimal solution of the sequencing problem (which is weighted

shortest processing time rule) is known. This implies that we can employ our branching scheme for the total flow time problem to solve its weighted version. The only modification would be the modification of the bounding schemes.

As long as the efficient measures are concerned, in addition to our producer related performance measure of total flow time, we can consider a customer related performance measure, like maximum lateness, total tardiness. In such a case, the rescheduling problem will be treated as a tri-criteria problem together with our stability measure. Once the due-dates of the problem is accepted as the promises given according to the initial schedule's completion times, any due-date related performance can serve as a stability measure. For example, maximum lateness, can be interpreted as the maximum completion time difference between initial and new schedules, which can be interpreted as the maximum delay in the delivery times.

5) Constructing the initial schedule

In this study, we assume that the initial schedule is optimal according to our efficiency measure. It does not reside any idle time and any non-optimal allocations which would be favored by the new schedule. An initial schedule construction, by predicting the disruption time and duration would be another future research area. In construction the initial schedule, the idle times that will serve as buffers and light loading of the machines that are more likely to be disrupted, should be considered.

6) Addressing the bicriteria assignment problem

In chapter 2, we develop solution approaches to the bicriteria minimum cost integer network flow problem. MCNF problems resides shortest path and assignment problems as special cases. In the literature, there is some research on bicriteria assignment problem, like singly constrained assignment problem. However, we are unaware of any reported study, on simultaneous optimization for assignment problems. Recognizing this gap, designing optimization algorithms for generation all efficient solutions and minimizing a composite function of the two criteria will be a worth-studying research area.

As a starting point, we can employ the following classical approach to generate all efficient solutions. We assume the two criteria are $\sum_{i,j} c_{ij}x_{ij}$ and $\sum_{i,j} d_{ij}x_{ij}$

where

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

and c_{ij} and d_{ij} are two different costs of assigning object i to resource j.

Procedure for Generating all efficient solutions of $\sum_{i,j} c_{ij}x_{ij}$ and $\sum_{i,j} d_{ij}x_{ij}$ criteria

Step 0**.** Solve the following assignment problems (P1) and (P2) to get two extreme efficient solutions

$$\text{(P1)} \quad \text{Min} \sum_{i,j} c_{ij}x_{ij} + e_1 \sum_{i,j} d_{ij}x_{ij}$$

$$\text{s.t.} \quad \sum_i x_{ij} = 1 \qquad \forall j$$

$$\sum_j x_{ij} = 1 \qquad \forall i$$

$$x_{ij} \in \{0,1\}$$

$$\text{(P2)} \quad \text{Min} \sum_{i,j} d_{ij}x_{ij} + e_2 \sum_{i,j} c_{ij}x_{ij}$$

$$\text{s.t.} \quad \sum_i x_{ij} = 1 \qquad \forall j$$

$$\sum_j x_{ij} = 1 \qquad \forall i$$

$$x_{ij} \in \{0,1\}$$

For appropriately selected values of e_1 and e_2.

Note that (P1) and (P2) give lower and upper bounds on the criteria values of all efficient solutions.

Let C_L and C_U be the lower and upper bounds of $\sum_{i,j} c_{ij}x_{ij}$ for all efficient schedules.

Let $k = C_U - 1$

Step 1. Solve the following singly constrained assignment problem

$$\text{Min} \sum_{i,j} d_{ij}x_{ij} + e_2 \sum_{i,j} c_{ij}x_{ij}$$

$$\text{s.t.} \quad \sum_{i} x_{ij} = 1 \qquad \forall j$$

$$\sum_{j} x_{ij} = 1 \qquad \forall i$$

$$\sum_{i,j} c_{ij}x_{ij} \le k$$

$$x_{ij} \in \{0,1\}$$

An optimal solution is an efficient point. Let x_{ij}* be the optimal values of the decision variables.

Step 2. If $\sum_{i,j} c_{ij}x_{ij}^* \ge C_L + 1$ then

$$k = \sum_{i,j} c_{ij}x_{ij}^* + 1$$

Go to Step 1

Stop, all efficient solutions are generated.

Alternatively in Step 0, we can set $k=D_U-1$ where D_U is an upper bound on the $\sum_{i,j} d_{ij}x_{ij}$ values of all efficient solutions and solve the singly constrained assignment problem with the objective $\text{Min} \sum_{i,j} c_{ij}x_{ij} + e_1 \sum_{i,j} d_{ij}x_{ij}$ and constraint $\sum_{i,j} d_{ij}x_{ij} \le k$.

REFERENCES

Abumaizar, R. J. and Svestka, J. A., 1997. Rescheduling job shops under random disruptions. *International Journal of Production Research*, 35, 2065-2082.

Aggarwal, V., 1985. A lagrangean-relaxation method for the constrained assignment problem. *Computers and Operations Research*, 12, 97-106.

Ahuja R.K.,. Magnanti T.L and Orlin J.B., 1993. *Network Flows Theory, Algorithms and Applications*. Prentice Hall.

Aktürk, M. S. and Görgülü, E., 1999. Match-up scheduling under a machine breakdown. *European Journal of Operational Research*, 112, 81-97.

Alagöz, O., and Azizoğlu, M., 2003. Rescheduling of identical parallel machines under machine eligibility constraints. *European Journal of Operational Research*, 149, 523-532.

Aneja, Y.P., and Nair K.P.K., 1979. Bicriteria transportation problem. *Management Science*, 25, 73-78.

Azizoğlu, M., and Alagöz, O., 2005. Parallel machine rescheduling with machine disruptions. *IIE Transactions*, 37, 1113-1118.

Aytuğ, H., Lawley, M.A., McKay, K., Mohan, S., and Uzsoy, R., 2005. Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research*, 161, 86-110.

Bean, J. C., Birge, J. R., Mittenthal, J., and Noon, C. E., 1991. Matchup scheduling with multiple resources, release dates and disruptions. *Operations Research* 39, 470-483.

Church L.K. and Uzsoy R., 1992. Analysis of periodic and event-driven rescheduling policies in dynamic shops, *International Journal of Computer Integrated Manufacturing*, 5, 153-163.

Curry J. and Peters B., 2005. Rescheduling parallel machines with stepwise increasing tardiness and machine assignment stability objectives, *International Journal of Production Research*, 43, 3231-3246.

Daniels, R. L., and Kouvelis, P., 1995. Robust scheduling to hedge against processing time uncertainty in single stage production. *Management Science,* 41, 363-376.

Ehrgott M., and Gandibleux X., 2000. A survey and annotated bibliography of multiobjective combinatorial optimization, *OR Spectrum*, 22, 425-460.

Glover, F., Karney, D., Klingman, D., and Russell R., 1978. Solving singly constrained transshipment problem. *Transportation Science*, 12, 277-297.

Hall, N. G., and Potts, C. N., 2005. Rescheduling for new orders. *Operations Research*, 52, 440-453.

Haimes, Y. Y., Wismer, D. A. and Lasdon, L. S. 1971. On bicriterion formulation of the integrated systems identifica-tion and system optimization, *IEEE Transactions on Systems, Man. and Cybernetics*, 1, 296-97.

Hamacher, H. W., Pedersen, C. R., and Ruzika S., 2007. Multiple objective minimum cost flow problems: a review. *European Journal of Operational Research*, 176, 1404-1422.

Henning, G. P., and Cerda, J., 2000. Knowledge-based predictive and reactive scheduling in industrial environments. *Computers and Chemical Engineering,* 24, 2315-2338.

Isermann H., 1974. Proper efficiency and the linear vector maximum problem. *Operations Research*, 22, 189-191.

Kaspi M., and Montreuil B., 1988. *On the scheduling of identical parallel processes with arbitrary initial processor available time*, Research Report 88-12, School of Industrial Engineering, Purdue University;.

Klingman, D, and Russell, R., 1978. A streamlined simplex approach to the singly constrained transportation problem. *Naval Research Logistics*, 25, 681-695.

Kondakçı S., Azizoğlu M., and Köksalan M., 1996. Note: Bicriteria Scheduling for Minimizing Flow Time and Maximum Tardiness, *Naval Research Logistics,* 43, 929-936.

Kutanoğlu, E., and Sabuncuoğlu, I., 2001. Routing-based Reactive Scheduling Policies for Machine Failures in Job Shops. *International Journal of Production research*, 39, 3141-3158.

Lawler, E. L., and Lenstra, J. K., Rinnooy Kan, A. H. G., Shmoys, D. B., 1989. *Sequencing and scheduling: Algorithms and complexity*, Reports BS-R8909, Centre for Mathematics and Computers Science, Amsterdam.

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

Lee H., and Pulat P.S., 1991. Bicriteria network flow problems: continous case, *European Journal of Operational Research*, 51, 119-236.

Lee H., and Pulat P.S., 1993. Bicriteria network flow problems: integer case, *European Journal of Operational Research*, 66, 148-157.

Leung, J. Y.-T., and Pinedo, M., 2004. A note on scheduling parallel machines subject to breakdown and repair. *Naval Research Logistics*, 51, 60-71.

Li E. and Shaw W., 1996. Flow-time performance of modified-scheduling heuristics in a dynamic rescheduling environment, *Computers and Industrial Engineering*, 31, 213 - 216.

Mason S. J., Jin, S., and Wessels, C. M., 2004. Rescheduling strategies for minimizing total weighted tardiness in complex job shops. *International Journal of Production Research*, 42, 613-628.

Mehta, S. V., and Uzsoy, R. M., 1998. Predictable scheduling of a job shop subject to breakdowns. *IEEE Transactions on robotics and automation*, 14, 365-378.

Mosheiov G., 1994. Minimizing the sum of job completion times on capacitated parallel machines, *Mathematical and Computer Modeling*, 20, 91-99.

Mustafa A. and Goh M., 1998. Finding integer efficient solutions for bicriteria and tricriteria network flow problem using DINAS, *Computers and Operations Research*, 25, 139-157.

Nikolova M., 1998. Properties of the effective solutions of the multicriteria network flow problem, *Problems of Engineering Cybernetics and Robotics*,. 47, 104-111.

O'Donovan, R., Uzsoy, R., and McKay, K. N., 1999. Predictable scheduling of a single machine with breakdowns and sensitive jobs. *International Journal of Production Research*, 18, 4217-4233.

O'Kane, J. F., 2000. A knowledge-based system for reactive scheduling decision-making in FMS. *Journal of Intelligent Manufacturing*, 11, 461-474.

Papadimitriou, C. H., and Steiglitz, K., 1982. *Combinatorial Optimization: Algorithms and Complexity*. Prentice Hall, NJ.

Przybylski A., Gandibleux X. and Ehrgott M., 2006. The bi-objective integer minimum cost flow problem – incorrectness of Sedeno-Noda and Gonzales-Martin's algoritm, *Computers and Operations Research*, 33, 1459-1463.

Pulat P.S., Huarng F., and Lee H., 1992. Efficient solutions for the bicriteria network flow problem, *Computers and Operations Research*,19, 649-655.

Raheja, A. S., and Subramaniam, V., 2002. Reactive recovery of job shop schedules – A review. *International Journal of Advanced Manufacturing Technologies,* 19, 756-763.

Rangsaritratsamee, R., Ferrell Jr., W. G., and Kurz, M. B., 2004. Dynamic rescheduling that simultaneously considers efficiency and stability. *Computers and Industrial Engineering*, 46, 1-15.

Sabuncuoğlu, I., and Bayız, M., 2000. Analysis of reactive scheduling problems in a job shop environment. *European Journal of Operational Research*, 126, 567-586.

Sedeno-Noda A., and Gonzales-Martin C., 2000. The biobjective minimum cost flow problem, *European Journal of Operational Research*, 124, 591-600.

Sedeno-Noda A., and Gonzales-Martin C., 2001. An algorithm for the biobjective integer minimum cost flow problem, *Computers and Operations Research*, 28, 139-156.

Sedeno-Noda A. and Gonzales-Martin C., 2003. An alternative method to solve the biobjective minimum cost flow problem, *Asia-Pacific Journal of Operational Research*, 20, 241-260.

Smith, W. E., 1956. Various optimizers for single stage production. *Naval Research Logistics Quarterly*, 70, 93-113.

Steuer, R. E., 1986. *Multiple criteria optimization: theory, computation and application*. John Wiley & Sons, Inc.

Visee, M., Teghem, J., Pirlot, M., and Ulungu, E.L., 1998. Two-phases method and branch and bound procedures to solve the bi-objective knapsack problem. *Journal of Global Optimization*, 12, 139-155.

Ünal, A. T., Uzsoy, R., and Kıran, A. S., 1997. Rescheduling on a single machine with part-type dependent setup times and deadlines. *Annals of Operations Research,* 70, 93-113.

Vieira, G. E., Herrmann, J. W., and Lin, E., 2003. Rescheduling manufacturing systems: A framework of strategies, policies and methods. *Journal of Scheduling,* 6, 39-62.

Volgenant, A. 1996. Linear and semi-assignment problems: a core oriented approach. *Computers and Operations Research,* 23, 917-932.

Wu S.D., Storer R.H. and Chang P.C., 1993. One-machine rescheduling heuristics with efficiency and stability as criteria, *Computers and Operations Research,* 20, 1-14.

VITA

Melih Özlen was born in Ankara on March 30, 1980. He graduated in 1997 from T.E.D. Ankara College. He received his B.S. degree in July 2001 and M.S. degree in August 2003 in Industrial Engineering from the Middle East Technical University. He has worked as a teaching and research assistant in the department from September 2002 to November 2006. His main areas of interest are combinatorial optimization, scheduling, and supply chain management.