PARALLEL MACHINE SCHEDULING SUBJECT TO MACHINE AVAILABILITY CONSTRAINTS

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

By
Kaya Sevindik
January 2006

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

Asst. Prof. Mehmet Rüştü Taner (Advisor)

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

Prof. M. Selim Aktürk

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

Asst. Prof. Yavuz Günalay

Approved for the Institute of Engineering and Sciences:

Prof. Mehmet Baray
Director of Institute of Engineering and Sciences

ABSTRACT

PARALLEL MACHINE SCHEDULING SUBJECT TO MACHINE AVAILABILITY CONSTRAINTS

Kaya Sevindik
M.S. in Industrial Engineering
Supervisor: Asst. Prof. Mehmet Rüştü Taner
January 2006

Within a planning horizon, machines may become unavailable due to unexpected breakdowns or pre-scheduled activities. A realistic approach in constructing the production schedule should explicitly take into account such periods of unavailability. This study addresses the parallel machine-scheduling problem subject to availability constraints on each machine. The objectives of minimizing the total completion time and minimizing the maximum completion time are studied. The problems with both objectives are known to be NP-hard. We develop an exact branch-and-bound procedure and propose three heuristic algorithms for the total completion time problem. Similarly, we propose exact and approximation algorithms also for the maximum completion time problem. All proposed algorithms are tested through extensive computational experimentation, and several insights are provided based on computational results.

Keywords: Scheduling, Parallel Machines, Total Completion Time, Makespan, Availability Constraints, Heuristics.

ÖZET

MAKİNA KULLANIM KISITLARI ALTINDA PARALEL MAKİNA ÇİZELGELEME PROBLEMİ

Kaya Sevindik

Endüstri Mühendisliği, Yüksek Lisans

Tez Yöneticisi: Yard. Doç. Dr. Mehmet Rüştü Taner

Ocak, 2006

Bir planlama çevreninde makinalar beklenmeyen bozulmalar veya daha önceden çizelgelenmiş aktiviteler nedeniyle kullanılabilirliklerini kaybedebilirler. Üretim çizelgelemesini gerçekçi bir yaklaşımla oluştururken bu tür kullanılamama periyotlarını hesaba katmak gerekir. Bu çalışma her makinada kullanım kısıtı altında paralel makina çizelgeleme problemi üzerinedir. Toplam bitirme zamanını enazlama ve en büyük bitirme zamanını enazlama hedef fonksiyonları çalışılmıştır. Her iki problemde NP-zor olarak bilinir. Toplam bitirme zamanını enazlama problemi için kesin bir dallandır-ve-sınırla prosedürü geliştirilmiş, ve üç farklı sezgisel yaklaşım algoritması önerilmiştir. Ayrıca en büyük bitirme zamanını enazlama problemi için bir kesin ve bir sezgisel yaklaşım algoritması önerilmiştir. Önerilen bütün algoritmalar kapsamlı ölçümleme deneylerinde test edilmiş ve ölçümleme sonuçlarından muhtelif bulgular sağlanmıştır.

Anahtar Sözcükler: Çizelgeleme, Paralel Makinalar, Toplam Bitirme Zamanı, En Büyük Bitirme Zamanı, Kullanım Kısıtları, Sezgisel.

To my grandfather Bahattin Çeber…

Acknowledgement

I would like to express my sincere gratitude to Asst. Prof. Dr. Mehmet Rüştü Taner for him supervision and encouragement in this thesis. His vision, guidance and leadership were the driving force behind the work. His endless patience and understanding let this thesis come to an end.

I am indebted to Prof. Dr. M. Selim Aktürk and Asst. Prof. Dr. Yavuz Günalay for accepting to read and review this thesis and also for their valuable comments and suggestions.

I would like to extend my deepest gratitude and thanks to my mom, dad, sister and my whole family for their continuous morale support, endless love and understanding.

And my sincere thanks to my officemates in EA332 and fellows from Industrial Engineering Department with whom I have shared my time during my graduate study for their patience and help.

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

Scheduling literature commonly assumes that machines are continuously available throughout the entire planning horizon. This assumption may not hold in many cases where unexpected breakdowns occur or when some activities are pre-scheduled on machines. Models and findings in the general scheduling literature may become inadequate to find optimum solutions for many cases with machine availability constraints. Although scheduling with availability constraints has gained some popularity during the last decade, there are still gaps in the relevant literature with respect to various machine arrangements and objective functions.

This study considers parallel machine scheduling problem with availability constraints on each machine with the objectives of minimizing total completion time ($\sum C_j$), and minimizing maximum completion time (makespan, $C_{\max}$). The problems with both objectives are known to be $\mathcal{NP}$-hard [18], [35]. There is exactly one unavailability period on each machine. Durations of unavailability periods are deterministic and can be different on different machines. Unavailability periods are non-simultaneous for total completion time problem, however they can be simultaneous in makespan problem.

Chapter 2 gives a short review of the literature on parallel machine scheduling problems with the objectives of minimizing $C_{\max}$ and minimizing $\sum C_j$, and the literature on machine scheduling with availability constraints. Chapter 3 defines the problem of minimizing

$\sum C_j$ with availability constraints, proposes exact and heuristic solution methods for this problem, and presents computational results with the proposed methods. Similarly, Chapter 4 focuses on the problem of minimizing $C_{\max}$ with availability constraints, develops exact and heuristic solution methods for this problem, and presents computational results. Finally, Chapter 5 concludes the thesis with a summary of the major findings, and gives some directions for future research.

Chapter 2

LITERATURE REVIEW

This section provides a literature review of parallel machine scheduling with objectives of minimizing $C_{\max}$ and $\sum C_i$, and machine scheduling with availability constraints.

2.1. PARALLEL MACHINE SCHEDULING

Elaborate reviews of the literature on parallel machine scheduling are provided in [44] and [10]. The problem with the objective of minimizing the total completion time ($\sum C_i$) can be solved in polynomial time [13]. It is well known that the problem of minimizing makespan ($C_{\max}$) on parallel identical machines is $\mathcal{NP}$-hard even with two machines [18], [19]. Dell'Amico and Martello [16] introduce lower bounds, heuristic algorithms and a branch-and-bound algorithm, and Ho and Wong [24] implement a lexicographical search procedure for this problem. The most recent exact procedure in the literature is the branch-and-cut algorithm developed by Mokotoff [46]. Dell'Amico and Martello [17] show that their original algorithm proposed in [16] outperforms this recent algorithm proposed by Mokotoff in [46] by some orders of magnitude.

The unrelated parallel machine problem with the objective of minimizing $C_{\max}$ is also widely studied. Exact and approximate algorithms for this problem are presented in Van de Velde [53], and Martello *et al*. [42]. The most recent exact procedure is the cutting plane scheme developed by Mokotoff and Chretienne [45]. Ghirardi and Potts [21] propose a recovering beam search algorithm in a recent study. Other examples of heuristic attempts include Mokotoff and Chretienne [45], Martello *et al.* [42], Horowitz and Sahni [25], Ibarra and Kim [26], De and Morton [15], Davis and Jaffe [14], Potts [48], Lenstra *et al.* [37], and Hariri and Potts [23]. Chang and Jiang [9] address an extension of the problem that incorporates arbitrary precedence constraints and develop a state-space-search procedure for its solution.

Salem *et al.* [50] consider an extension of the problem with machine eligibility restrictions in which a subset of the jobs can be processed only on certain machines. They propose a branch-and-bound algorithm that exploits a customized lower bound. Their algorithm is capable of efficiently solving instances with up to 8 machines and 40 jobs.

The problem of preemptive task scheduling on parallel identical machines with the objective of minimizing the $C_{\max}$ can be solved in polynomial time in most cases even if there are precedence constraints. Muntz and Coffman [47] propose an algorithm for the case in which there are tree-like precedence constraints on parallel machines.

Bruno *et al* [8] show that the problem of scheduling jobs on parallel identical machines with the objective of minimizing the total weighted completion time is $\mathcal{NP}$-hard even with two machines. Azizoglu and Kirca [4] consider both the identical and the uniform machine cases of the multi-machine problem, establish optimality characteristics, develop a lower bound, and construct a branch-and-bound algorithm.

2.2. SCHEDULING WITH AVAILABILITY CONSTRAINTS

Schmidt [52] and Sanlaville and Schmidt [51] provide an excellent review of scheduling with machine unavailability. These reviews present single, parallel, and flow shop models with total completion time, $C_{\max}$, and due date related objectives.

Research on single machine problems under machine availability constraints is very limited. Lee and Liman [34] show that the problem of minimizing $\sum C_i$ on a single machine without preemption is $\mathcal{NP}$-hard. Sadfi *et al*. [49] study the single machine scheduling problem with a given unavailability period and deterministic processing times. They consider the total completion time objective and propose an approximation algorithm with a worst-case error bound of 20/17. Akturk *et al*. [2] study single machine problem with multiple unavailability periods by minimizing total completion time. They provide a number of heuristic methods and discuss the performances of these heuristics through computational experiments. Akturk *et al*. [3] also consider the SPT (Shortest Processing Time) list scheduling for the same problem and they provide worst case bounds under different conditions. Lee and Leon [33] consider the single machine scheduling problem with a rate-modifying activity in which the starting time of the rate modifying activity is a decision variable. They develop polynomial time algorithms for the $C_{\max}$ and total completion time objectives, and a pseudo-polynomial algorithm for a special case of the total weighted completion time problem. Lee and Leon also show in this paper that the problem of minimizing maximum lateness is $\mathcal{NP}$-hard, and they provide a pseudo-polynomial algorithm. Graves and Lee [22] study a single machine scheduling problem with maintenance windows and semi-resumable job processing with the starting time of the maintenance a decision variable. They show that the problem of minimizing the total weighted completion time is $\mathcal{NP}$-hard, while the SPT and EDD (Earliest Due Date) ordering give optimal schedules for the total completion time and $L_{\max}$ objectives, respectively. Liao and Chen [39] study the single machine problem with multiple periods of unavailability with the objective of minimizing maximum tardiness. They develop both an exact branch-and-bound procedure and a heuristic algorithm.

Parallel machine problems with the objectives of minimizing $C_{\max}$, and $L_{\max}$ tend to be polynomially solvable when preemption is allowed. Although polynomial time algorithms can be found for some special cases, most non-preemptive problems are known to be $\mathcal{NP}$-hard.

Lee [29] studies parallel identical machine scheduling problem with initial availability constraints. He studies the objective of minimizing $C_{\max}$, and provides a polynomial time approximation algorithm with a worst-case error bound of 4/3.

Liao *et al.* [40] study the objective of minimizing $C_{\max}$ on two parallel machines with an availability constraint during a given interval on only one machine. They classify the problem into four cases, and use versions of a lexicographical search algorithm originally proposed by Ho and Wong [24] to device an exact method for each case.

Gharbi and Haouari [20] consider the parallel identical machine problem with non-decreasing and non-simultaneous machine availability times, release dates, and delivery times and the objective of minimizing $C_{\max}$. They develop new lower and upper bounds based on max-flow computations, and propose a branch-and-bound algorithm using these bounds. This algorithm is able to solve instances with up to 700 jobs and 20 machines.

Leung and Pinedo [38] study a preemptive problem with identical parallel machines, and deterministic unavailability periods. They consider the total completion time, $C_{\max}$ and maximum lateness objectives. For the total completion time objective, they show in what conditions the optimum solution can be obtained via list scheduling. For the $C_{\max}$ objective, they consider problems with precedence constraints and fixed processing times. They determine conditions on precedence constraints and on unavailability periods to find a polynomial time algorithm. They also give a polynomial time algorithm that gives optimum solutions for the $L_{\max}$ and $C_{\max}$ objectives without any precedence constraints.

Blazewicz *et al.* [5] also consider the preemptive problem case when machines are available for processing for certain time intervals with precedence constraints. They show that the P, staircase/*pmtn*, intree/$C_{\max}$ problem is $\mathcal{NP}$-hard in the strong sense. They propose a polynomial time, linear programming based procedure to solve the case with chain-like precedence constraints and a staircase pattern of machine availability. They also study uniform and unrelated parallel machine problems with arbitrary patterns of unavailability with the $C_{\max}$ and $L_{\max}$ objectives. They propose a network flow approach for the uniform, and a linear programming approach for the unrelated machine problem.

Lee and Liman [34] study the two parallel machine scheduling problem with an unavailability period on one machine with the objective of minimizing the total completion time. They show that the problem is $\mathcal{NP}$-hard, and propose a pseudo-polynomial, online algorithm as well as a constructive heuristic with a worst-case error bound of 3/2. Kaspi and Montreuil [27], and Liman [41] study the same problem, and show that in the special case involving only initial availability constraints, SPT ordering gives the optimum solution.

Lee and Chen [32] study the problem of scheduling jobs and maintenance activities on parallel machines with the objective of minimizing the total weighted completion time. As in [22], the starting times of the maintenance activities are taken as decision variables. The authors consider the two cases of overlapping and non-simultaneous periods of unavailability. They show that the problem is $\mathcal{NP}$-hard in both cases. They propose a branch-and-bound method based on column generation techniques to solve medium sized problems within a reasonable computational time.

Lee and Lin [36] study a single-machine scheduling problem with stochastic breakdowns, and a rate modifying maintenance/repair activity. They consider the objectives of expected $C_{\max}$, total expected completion time, expected maximum lateness, and maximum expected lateness. A machine becomes unavailable due to a maintenance activity triggered by the decision maker who wishes to increase its speed or repair activity required due to a fatal breakdown. The machine assumes its normal speed after the repair/maintenance activity.

All studies on the flowshop problems with machine unavailability focus on the $C_{\max}$ objective. Lee [30] studies a two-machine flowshop scheduling problem of minimizing $C_{\max}$ with an availability constraint. He considers a fully deterministic environment. He shows that the problem is $\mathcal{NP}$-hard and develops a pseudo-polynomial algorithm to solve the problem optimally. In addition, he develops two heuristic algorithms each having a complexity of $O(n \log n)$. The worst-case error bound of his first algorithm, which is proposed for problems with an availability constraint on machine 1, is 3/2, and that of his second algorithm for problems with an availability constraint on machine 2 is 4/3. Cheng and Wang [11] address the same problem in [30] and show that the relative worst-case error bound of 3/2 is tight for the heuristic proposed in [30] when there is an availability constraint on machine 1. Also, they propose an improved heuristic algorithm with a relative worst-case error bound of 4/3.

Blazewicz *et al.* [6] study the same problem with availability constraints on both machines. They analyze several constructive and local search based algorithms in the literature through computational experimentation.

Cheng and Wang [12] study the generalization of the problem studied in [30] in that having an availability constraint imposed on each machine. Availability constraints are consecutive. They identify some characteristics of the problem in the semi-resumable case, and provide a heuristic with a relative worst-case error bound of 5/3 for the non-preemptive case. Breit [7] also studies the preemptive version of the problem in [30] with an availability constraint only on machine 2. He proposes a heuristic algorithm with a worst-case error bound of 5/4.

Lee [31] considers the two-machine flowshop problem and an availability constraint on only one machine and on both machines. Job processing is semi-resumable where if a semi-resumable job is interrupted by an unavailability period, the processing can continue with after a certain setup time. He provides a complexity analysis, develops a pseudo-polynomial dynamic programming algorithm, and proposes heuristics supplemented with error bounds.

Aggoune [1] considers m-machine flowshop scheduling problem with availability constraints with the objective of minimizing $C_{\max}$. He studies two cases of the problem. In the first case, starting times of unavailability periods are fixed, while in the second case starting times are in a time interval. He proposes a genetic algorithm and a tabu search procedure.

2.3. SUMMARY

Total completion time problem on the parallel machines with continuous availability is easy to solve, however, makespan problem turns out to be $\mathcal{NP}$-hard. Extensive research is conducted on makespan problem on parallel machines, which includes identical, uniform and unrelated parallel machines. With availability constraints, single and parallel machine scheduling problems are $\mathcal{NP}$-hard for both total completion time and makespan objectives. Although problems with availability constraints have become very popular for the last decade, the relevant literature is still very limited. For both objectives minimizing $\sum C_i$ and makespan, this study fills the gap in literature of scheduling problems with availability constraints on each of parallel machines.

Chapter 3

TOTAL COMPLETION TIME PROBLEM

This section defines the problem of minimizing $\sum C_j$ on parallel identical machines with availability constraints, develops exact and heuristic solution methods for this problem, and presents computational results with the proposed methods.

3.1. PROBLEM DEFINITION

All jobs are released simultaneously. Processing times are known and deterministic. Each job is to be processed exactly on one of the parallel identical machines. Job processing is non-preemptive. There is exactly one unavailability period on each machine for which the starting time and duration are given. Periods of unavailability on different machines do not overlap. That is, at most one machine may become unavailable at any time instance. This assumption is required since the lower bound to be defined in section 3.3 would not be valid without it.

Since this particular problem is a more general version of the one that is proved to be $\mathcal{NP}$-hard by the Lee and Liman [35], it is also $\mathcal{NP}$-hard.

MATHEMATICAL FORMULATION

The schematic representation of the problem is as follows. Available times of machines for processing before and after the unavailability periods will be denoted as "availability intervals" or only "intervals". Every machine has two "intervals" for processing jobs. There are $2m$ intervals for processing in the system in cumulative.

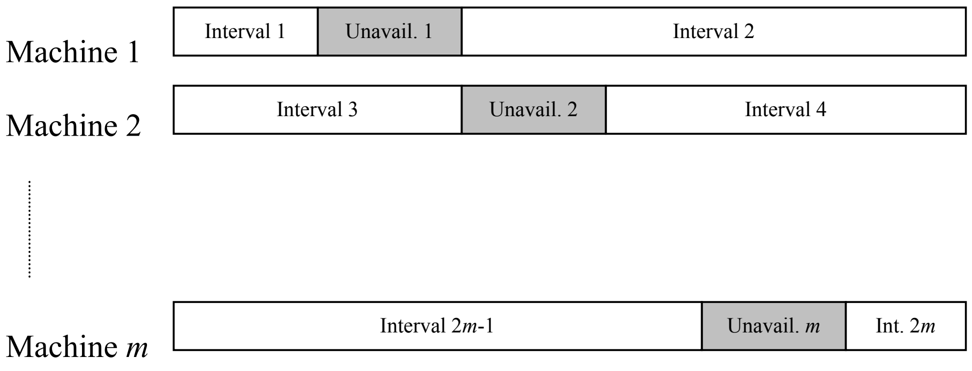

Figure 3-1 Schematic representation of problem

Define a_i as the starting time of the interval i, that is $a_i = 0$ for intervals $i = 1, 3, \dots, 2m-1$; but a_i equals to the summation of starting time and duration of the unavailability period on the corresponding machine for intervals $i = 2, 4, \dots, 2m$.

The notation is as follows.

Indices:

h :	Machine index, $h = 1,2,\dots,m$
j, k, l, g :	Job index, $j, k, l, g = 1,2,\dots,n$
i:	Interval index, $i = 1,2,\dots,2m$

Parameters:

s_h:	Starting unavailability period on machine h
Δ_h:	Duration of unavailability period on machine h

p_k: Processing time of job k

Sets:

P_k: Set of jobs preceding job k in the same interval;

S_k: Set of jobs succeeding job k in the same interval;

Decision variables:

x_{ik}: $= \begin{cases} 1 & \text{if job } k \text{ is processed in interval } i \\ 0 & \text{otherwise} \end{cases}$

b_{jk} $= \begin{cases} 1 & \text{if jobs } j \text{ and } k \text{ are processed in the same interval} \\ 0 & \text{otherwise} \end{cases}$

r_k: Starting time of the processing of job k;

Assume without loss that jobs are indexed in SPT order, that is, $p_1 \le p_2 \le ... \le p_n$. Then, the problem can be formulated as follows.

$$\min \sum_{i=1}^{2m} \sum_{k=1}^{n} x_{ik} \left(a_i + r_k + p_k \right)$$

Subject to

$$\sum_{i=1}^{2m} x_{ik} = 1 \qquad \text{for } \forall k \qquad (3.1)$$

$$b_{jk} \ge x_{ij} + x_{ik} - 1 \qquad \text{for } i=1,...,2m \text{ and } \forall j,k \text{ s.t } j<k \qquad (3.2)$$

$$r_k \ge \sum_{j=1}^{k-1} b_{jk} p_j \qquad \text{for } \forall k \qquad (3.3)$$

$$\sum_{k=1}^{n} x_{2h-1,k} p_k \le s_h \qquad \text{for } h=1, 2,\dots, m \qquad (3.4)$$

$$x_{ik}, b_{jk} \in \{0,1\} \qquad \text{for } \forall i, \text{ and } \forall k$$

$r_k \in \mathbb{N}$ for $\forall k$

Constraint set (3.1) ensures that all jobs are assigned to exactly one of the intervals. Constraint set (3.2) determines which jobs are processed in the same interval. Constraint set (3.3) indicates that the starting time of a job cannot be any smaller than the sum of the processing times of the preceding jobs within the same interval, and a job is preceded by the jobs with smaller processing times. Finally, constraint set (3.4) ensures that all jobs scheduled to be processed before unavailability can be completed so as to allow for the corresponding unavailability period to start on time.

3.2. SOLUTION CHARACTERISTICS

It is shown that SPT ordering minimizes the total completion time on continuously available, parallel identical machines. However, SPT ordering is not the only solution for this case. Consider two-parallel machine case, there are $2^{\lfloor n/2 \rfloor}$ ($\lfloor x \rfloor$ is the greatest integer strictly smaller than *x*) optimal solutions for the problem where *n* is the number of jobs to be completed. As stated before, jobs are indexed in SPT order. Then the completion time of job *j* in the optimal schedule is: $C_j = \sum_{k \in P_j} p_k + p_j$. It can be seen that every job takes place in the calculation of completion time of the jobs succeeding it. If we do some algebraic operations the total completion time turns out to be $\sum_{j=1}^{n} (|S_j| + 1) p_j$ where $|S_j|$ is the number of jobs in S_j. It can be seen that interchanging the place of a job with the place of another job with same number of successors (interchange place of jobs *j* and *k* s.t. $|S_j| = |S_k|$)

does not violate optimality. Since the number of jobs holding this condition is $\lfloor n/2 \rfloor$ in two-parallel machine problem, there are $2^{\lfloor n/2 \rfloor}$ optimal solutions. Therefore, SPT ordering is a sufficient but not a necessary condition for optimality in the case of continuous machine availability. For the same case, a general necessary and sufficient condition can be summarized as follows.

***Lemma 3.1**:* Let k and j be two jobs assigned to different machines with processing times p_k and p_j such that $p_k < p_j$. Then in the optimal solution $|S_k| \geq |S_j|$.

Proof: By contradiction. Suppose jobs k and j with $p_k < p_j$ are scheduled in the optimal solution such that $|S_k| < |S_j|$. Then the objective value is

$$\sum_{g=1}^{n} C_g = \sum_{l \in P_k} C_l + C_k + \sum_{l \in S_k} C_l + \sum_{l \in P_j} C_l + C_j + \sum_{l \in S_j} C_l$$

if we exchange the places of jobs k and j, completion times of jobs that precede these jobs remain the same, and completion times of jobs k and j become

$C_k = r_j + p_k$ and $C_j = r_k + p_j$, respectively.

Thus, the summation of C_k and C_j does not change. However starting times of jobs succeeding job k in the original schedule increase by $p_j - p_k$ and that of jobs succeeding job j in the original schedule decrease by $p_j - p_k$. As a result, the new objective value after the exchange is:

$$\sum_{g=1}^{n} C_g^* = \sum_{l \in P_k} C_l + \sum_{l \in P_j} C_l + C_k + C_j + \sum_{l \in S_k} C_l + \sum_{l \in S_j} C_l + |S_k|(p_j - p_k) - |S_j|(p_j - p_k)$$

$$\sum_{g=1}^{n} C_g^* = \sum_{l=1}^{n} C_l - \left(|S_j| - |S_k|\right)(p_j - p_k)$$

We know that $|S_k| < |S_j|$ and $p_k < p_j$. Hence, the $\sum C_g$ in the new schedule is strictly smaller than that in the original schedule. Therefore, there cannot be an optimal schedule with
$|S_k| < |S_j|$ and $p_k < p_j$.

Corollary 3.1: For the continuous available machines case, jobs assigned to the same machine must be processed in SPT order in the optimal schedule.

Optimality Conditions

Because jobs are non-preemptive and starting time and duration of each interval is fixed, jobs assigned to different intervals may not have easily identifiable precedence relations. Intervals before unavailability periods have a processing capacity; hence certain jobs cannot be processed together in these intervals. This makes it very difficult to find precedence relations valid for every problem instance.

Observations on the parallel machine problem with continuously available machines lead to the following optimality properties.

Corollary 3.2: Jobs must be sequenced in SPT order within each interval.

Proof: The sub-problem within each interval can be thought of as a single machine problem, and it is well known that SPT ordering minimizes the flow time, [3].

Define I_i as the idle time left in interval i in the optimal schedule. Since all remaining jobs can be processed continuously on either machine after the period of unavailability, respective idle times in intervals 2, 4,…,2*m* are always zero. Idle times for intervals before the unavailability period equal to the difference between the initial available processing time of the corresponding interval and cumulative processing time of jobs assigned to that interval.

$$I_i = Av_i - \sum_{j \in As_i} p_j \ .$$

Where $Av_i = s_{(i+1)/2}$ that is starting time of period of unavailability on the corresponding machine, and As_i is the set of jobs assigned to intervals i where ($i = 1,3,\ldots,2m$-1).

Lemma 3.2: Let g, j, k, and l be four jobs with processing times $p_g < p_j < p_k < p_l$ respectively. Suppose job j is assigned to interval x, and job k is assigned to interval y in the optimal solution. Let g be the last job assigned to interval y with a smaller processing time than j, and let l be the first job assigned to interval x with larger processing time than k. Then at least one of the following two conditions must hold for optimality:

i) The places of job j and job k cannot be exchanged due to availability constraint, that is:

$I_x + p_j < p_k$

ii) The total gain obtained from the exchange of the places of job j and job k is not positive that is:

$$|S_j| \times p_j - (|S_g| - 1) \times p_j + |S_k| \times p_k - (|S_l| + 1) \times p_k \leq 0$$

Proof: By contradiction. Suppose neither condition holds in the optimal solution. That is, there exists an optimal schedule S such that

i) $I_x + p_j \geq p_k$, and

ii) $\left(\left|S_g\right|-1\right)\times p_j-\left|S_j\right|\times p_j+\left(\left|S_l\right|+1\right)\times p_k-\left|S_k\right|\times p_k<0$

That $I_x+p_j\geq p_k$ implies interchanging jobs j and k leads to a new feasible solution. If we exclude job j from interval x and job k from interval y the gain is

$\left|S_j\right|\times p_j+\left|S_k\right|\times p_k$

And if we assign job j to interval y and job k to interval x the increase in the objective value is

$\left(\left|S_g\right|-1\right)\times p_j+\left(\left|S_l\right|+1\right)\times p_k$ then total gain is

$\left(\left|S_j\right|-\left(\left|S_g\right|-1\right)\right)p_j+\left(\left|S_k\right|-\left(\left|S_l\right|+1\right)\right)p_k$

Opening of the term above is the same as the term in condition (*ii*), multiplied by (-1). Hence, adding these gives a positive gain in the objective value. Hence, any schedule that satisfies both of the two conditions above cannot be an optimal solution.

3.3. LOWER BOUND

Leung and Pinedo [38] show that Shortest Remaining Processing Time (SRPT) order gives an optimum solution to the parallel machine problem subject to machine unavailability and preemptive processing times.

Since all solutions for non-preemptive case is also feasible solutions for preemptive case, the optimum solution of the preemptive case dominates all solutions for the non-preemptive case. Hence, the preemptive solution constitutes a lower bound for our problem.

This lower bound is expected to perform better in those problems in which the periods of unavailability on all machines occur both close to either the beginning or the end of the planning horizon. Keeping in mind that the lower bound sets the idle time immediately preceding the interval of unavailability on each machine equal to zero, this insight can be explained by the following two observations on the optimum solution.

Observation 1: The length of the idle time tends to be larger as the starting time of the unavailability gets larger.

Observation 2: The number of succeeding jobs amplifies the impact of the length of the idle time on the objective, when the starting time of unavailability gets smaller.

The total impact on the objective tends to assume its maximum level when the starting time of unavailability lies in the medium term, and both factors are in play. Hence, the expected poorer performance in the lower bound for such cases.

3.4. SOLUTION METHODS

We identify some problem characteristics in Section 3.2. We use these characteristics to develop exact and heuristic algorithms for the problem. In this section we present our solution methods to find exact and approximate solutions for the problem and we discuss our exact and approximate algorithms in detail.

3.4.1. BRANCH-AND-BOUND ALGORITHM

We propose a branch–and–bound algorithm to solve small- and medium-sized problems optimally. As mentioned before, given the job assignments to the intervals *SPT* ordering within each interval gives an optimal solution. Hence, the key decision is how to assign the jobs to the intervals.

3.4.1.1. Lower Bound for a Partial Schedule

A modification of the overall lower bound obtained from the solution to the preemptive problem gives a lower bound for each particular schedule corresponding to the nodes of the branch-and-bound tree. Level 0 in the branch-and-bound tree defines that no job is assigned any of the intervals. In the level 1, decision of assigning the job with *SPT* is done, and for the next levels this continues with remaining jobs in *SPT* order. Fixing the assignment of job k (level k) to the corresponding interval and assigning the remaining jobs by SRPT, calculate lower bound for a node at level k. While calculating the lower bund for a node at level k, the key point is jobs fixed to intervals should be the first k jobs in *SPT* order of all jobs.

Obviously, processing the jobs not included in each particular schedule in *SRPT* order gives a lower bound for that particular schedule. This lower bound can be strengthened through a modification based on the following observations.

Fact 1(Look Ahead Factor)*:* Let S be a given partial schedule and, let J_k be the next job to be assigned, that is, first $k-1$ jobs are fixed and job k is the job with the *SRPT*. This scheme belongs to $(k-1)^{\text{th}}$ level of the branch-and-bound tree. If J_k cannot be assigned to interval i ($i \in \{1,3\ldots 2m\}$) completely, that is the idle time of interval i ($i \in$ $\{1,3\ldots 2m$

– 1}) for the partial schedule P is smaller than p_k, for this and next levels, no partial or complete assignment will be done to interval i.

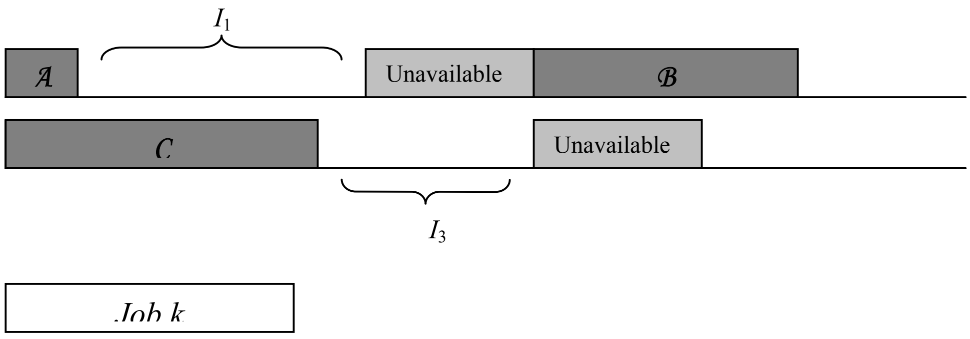

Figure 3-2 Representation of the partial schedule

Consider the two-machine case above. Suppose that $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ are set of jobs assigned respectively intervals 1, 2, and 3 in partial schedule P. Suppose that I_1 is greater than or equal to p_k whereas the I_3 is smaller than p_k. According to fact, there will be no assignments to interval 3 until the branch-and-bound algorithm back-processes to $(k – 1)^{st}$ level.

3.4.1.2. Node Generation Process

Let n be the number of jobs and assume without loss jobs are indexed in *SPT* order. Number of jobs in the partial schedule designates the level in the branch-and-bound tree, and the nodes within each level correspond to the intervals to which the next job can be assigned. Since the jobs are indexed in *SPT* order, level 1 corresponds to job 1, level 2 corresponds to job 2 and so on. At each level a new set of nodes are generated by assigning the next job to each one of the $2m$ possible intervals. Then the lower bound is taken as the best promising node. This best promising node is selected as the parent node for the next level, and the process repeats itself until all jobs are scheduled. Hence, a job is assigned to an interval; the algorithm passes on the next job with the smallest processing time.

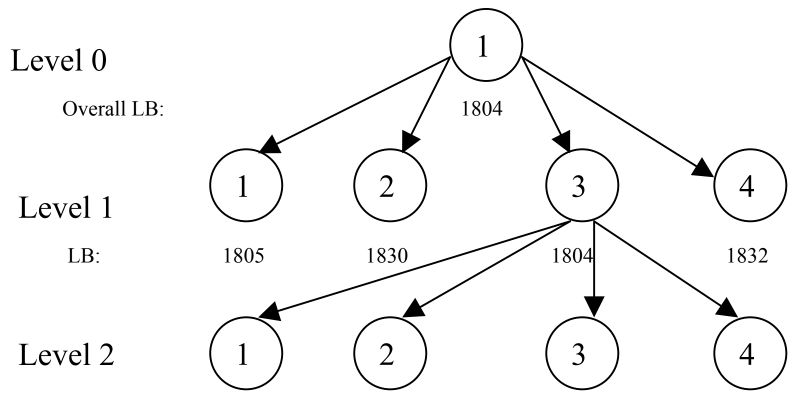

Figure 3-3 Node generation process

Figure 3-3 depicts levels 0-2 for a branch-and-bound tree for a hypothetic scenario of a two-machine problem example, the overall lower bound for the problem is 1804, that is also the lower bound for node 1 at level 0. At level 1, the first job is assigned to each of the four intervals, and the corresponding lower bounds are calculated. At level 2, four new nodes are generated from the node gave the smallest lower bound in level 1.

When it is not possible to generate new nodes from the current node, back-processing runs. For instance, when the last job is inserted to an interval, there remains no jobs to be inserted to generate nodes or when all possible nodes that can be generated from a node do not give a lower bound that is better than the current feasible solution.

In the back- processing current node's lower bound is updated as the best of the lower bounds of its child nodes, and process returns to the previous node (level).

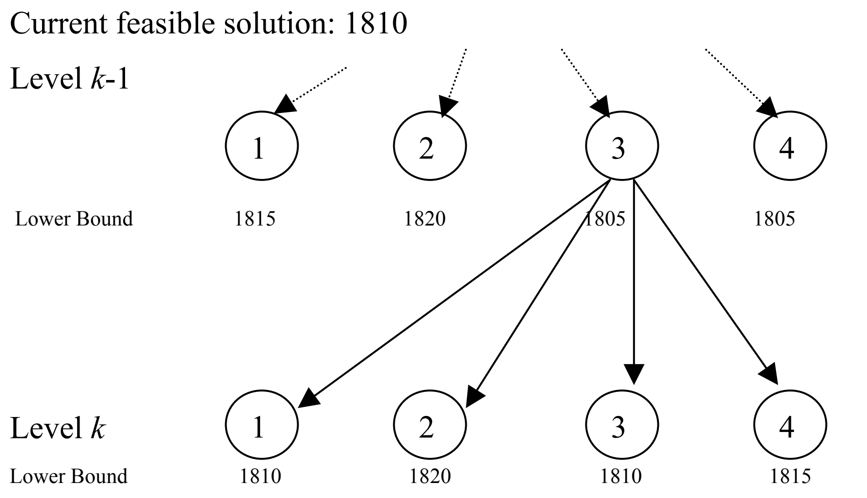

Figure 3-4 Back-processing

Consider a two-machine problem; at level k we have a feasible solution with a total completion time ($\sum C_j$) value of 1810. As it can be seen in the Figure 3-3, no other node at the same level has a potential to lead to a better solution. Thus, the algorithm proceeds at level $k - 1$, equal to lower bounds of its child nodes as 1810, and deletes all of the lower bounds of nodes in level k.

The resulting partial graph is shown in Figure 3-5

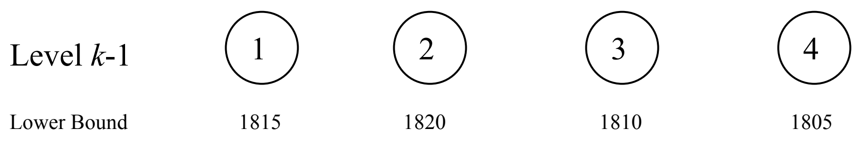

Figure 3-5 Partial graph

Since only node 4 in level $k - 1$ has a corresponding lower bound that is smaller than that of the best incumbent solution, the process continues by generating 4 new branches, emanating from this node. The algorithm stops with the optimal solution, when all updated lower bounds of the nodes in level 1 is greater than or equal to the best incumbent solution so far found.

3.4.2. HEURISTIC ALGORITHMS

Since our problem is $\mathcal{NP}$-hard exploration of heuristic approaches is justified. Consequently, in this section, we exploit the previously identified optimality properties to propose a constructive heuristic algorithm, a neighborhood search mechanism, and a simulated annealing application.

3.4.2.1. Constructive Heuristic Algorithm

This algorithm, called Greedy Assignment (Greedy) is based on greedy assignments of jobs to the intervals by using the "lower bound" of a partial schedule defined in Branch-and-Bound section. The algorithm starts with the first job (the job with SPT), assigns this job to the interval, which gives the minimum lower bound, and continues with next jobs in the order of SPT. The solution obtained from Greedy is used as the initial feasible solution in the branch-and-bound algorithm.

Although the algorithm assigns jobs with respect to lower bounds, resulting schedule has some intelligence due to the look ahead factor in the lower bound calculation. This factor allows the algorithm not to assign jobs to the intervals before the unavailability periods resulting large idle times in these intervals.

Worst Case Error Bound of Greedy

Let *POPT* be the optimal solution for the preemptive case of the problem and *GSOL* and *NOPT* are the heuristic and optimum solutions for the non-preemptive case of the problem. We know that *SRPT* ordering of preemptive case gives a lower bound for the non-preemptive case that is, $POPT \leq NOPT$ for all problem instances.

Define $\text{minsum}_{(x)}(y_1,\ldots, y_z)$ that equals to summation of first x smallest values of the set $(y_1,\ldots, y_z)$. For instance, $\text{minsum}_{(2)}(2, 3, 4) = 2 + 3$. Similarly, define $\text{maxsum}_{(x)}(y_1,\ldots, y_z)$ that equals to summation of first x largest values of the set $(y_1,\ldots, y_z)$. For instance $\text{maxsum}_{(2)}(2, 3, 4) = 3 + 4$. Also define x_y that equals to "$x - \lfloor x / y \rfloor \times x$". For instance, $4_2 = 2$ and $3_2 = 1$. Let $\lceil z \rceil$ be the smallest integer that is greater than or equal to z.

Theorem 3.1: When the periods of unavailability on different machines are of equal length, the worst case error bound of the Greedy algorithm is 2.

***Proof*:** The greatest deviation between *GSOL* and *POPT* is obtained when there is no job assignment in intervals before unavailability periods. Figure 3-6 depicts the particular 2-machine case, where Δ is the length of a period of unavailability.

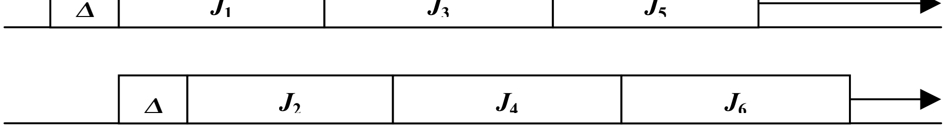

Figure 3-6 Worst case solution obtained from "Greedy" for 2 machines

Assume without loss of generality $s_1 < s_2 < \ldots < s_m$. For m-machine case, greedy gives a solution displaying this characteristic if and only if $p_1 \geq s_m$ where s_m is the starting time of unavailability period on machine m. Also we know that $s_m \geq (m-1)\Delta$, as the unavailability periods do not overlap. For the worst case defined above, "Greedy" finds a solution ($GSOL$) of:

$$\left\lceil \frac{n}{m} \right\rceil \left(p_1 + \ldots + p_{n_m}\right) + \left(\left\lceil \frac{n}{m} \right\rceil - 1\right)\left(p_{n_m+1} + \ldots + p_{n_m+m}\right) +$$
$$\left(\left\lceil \frac{n}{m} \right\rceil - 2\right)\left(p_{n_m+m+1} + \ldots + p_{n_m+2m}\right) + \ldots + \left(p_{n-m+1} + \ldots + p_n\right) +$$
$$\left(s_1 + \Delta + s_2 + \Delta + \ldots + s_{n_m} + \Delta\right) + \left(\left\lceil \frac{n}{m} \right\rceil - 1\right)\left(s_1 + \Delta + s_2 + \Delta + \ldots + s_m + \Delta\right) \qquad (3.5)$$

In the other hand, $SRPT$ finds a solution ($POPT$) for preemptive case:

$$p_1 + \ldots + p_{m-1} + \left[\left(p_m - s_m + (m-1)\Delta\right) + \min\left(s_m + \Delta, p_1\right)\right] +$$
$$\left(\left\lceil \frac{n}{m} \right\rceil - 1\right) \text{minsum}_{(n_m)} \begin{pmatrix} \max\left(s_m + \Delta, p_1\right), p_2, p_3, \ldots, p_{m-1}, \\ \left[\left(p_m - s_m + (m-1)\Delta\right) + \min\left(s_m + \Delta, p_1\right)\right] \end{pmatrix} +$$

$$\left(\left\lceil \frac{n}{m} \right\rceil - 2\right) \text{maxsum}_{(m-n_m)} \begin{pmatrix} \max\left(s_m + \Delta, p_1\right), p_2, p_3, \ldots, p_{m-1}, \\ \left[\left(p_m - s_m + (m-1)\Delta\right) + \min\left(s_m + \Delta, p_1\right)\right] \end{pmatrix} +$$

$$\left(\left\lceil \frac{n}{m} \right\rceil - 1\right)\left(p_{m+1} + \ldots + p_{m+n_m}\right) + \left(\left\lceil \frac{n}{m} \right\rceil - 2\right)\left(p_{m+n_m+1} + \ldots + p_{2m+n_m}\right) + \ldots +$$

$$\left(p_{n-m+1} + \ldots + p_n\right) \qquad (3.6)$$

Since $(m-1)\Delta \leq s_m$, for $s_1 < s_2 < \ldots < s_m < p_1 \leq p_2 \leq \ldots \leq p_n$, The deviation between *GSOL* and *POPT* is maximized when $\Delta \to 0$, $s_1 \to p_1$, $p_n \to p_1$. In these limits the equations (3.5) and (3.6) tend to

$$n_m \left\lceil \frac{n}{m} \right\rceil p_1 + m\left(\left\lceil \frac{n}{m} \right\rceil - 1\right) p_1 + m\left(\left\lceil \frac{n}{m} \right\rceil - 2\right) p_1 + \ldots + m \times p_1 + n_m p_1 + m\left(\left\lceil \frac{n}{m} \right\rceil - 1\right) p_1$$

(3.5) becomes (3.5') and

$$mp_1 + n_m\left(\left\lceil \frac{n}{m} \right\rceil - 1\right) p_1 + (m - n_m)\left(\left\lceil \frac{n}{m} \right\rceil - 2\right) p_1 + n_m\left(\left\lceil \frac{n}{m} \right\rceil - 1\right) p_1 + m\left(\left\lceil \frac{n}{m} \right\rceil - 2\right) p_1 + m\left(\left\lceil \frac{n}{m} \right\rceil - 3\right) p_1 + \ldots + mp_1$$

(3.6) becomes (3.6').

After reordering the terms in both equations and dividing (3.5') to (3.6'), we obtain (3.5')/(3.6'):

$$\frac{n_m p_1 + n_m \left\lceil \frac{n}{m} \right\rceil p_1 + 2m\left(\left\lceil \frac{n}{m} \right\rceil - 1\right) p_1 + m\left(\left\lceil \frac{n}{m} \right\rceil - 2\right) p_1 + \ldots + m \times p_1}{n_m \left\lceil \frac{n}{m} \right\rceil p_1 + m\left(\left\lceil \frac{n}{m} \right\rceil - 1\right) p_1 + m\left(\left\lceil \frac{n}{m} \right\rceil - 2\right) p_1 + \ldots + m \times p_1} \qquad (3.7)$$

After doing some algebraic operations, this ratio becomes

$$\frac{n_m\left(1 + \left\lceil \frac{n}{m} \right\rceil\right) + m\left(\left\lceil \frac{n}{m} \right\rceil - 1\right) + \frac{1}{2}\left(\left\lceil \frac{n}{m} \right\rceil\right)\left(\left\lceil \frac{n}{m} \right\rceil - 1\right)}{n_m\left(\left\lceil \frac{n}{m} \right\rceil\right) + \frac{1}{2}\left(\left\lceil \frac{n}{m} \right\rceil\right)\left(\left\lceil \frac{n}{m} \right\rceil - 1\right)} \qquad (3.8)$$

We can see that this ratio is maximized at $n \le m$, with the value of 2.

From the above result, we know that $\frac{GSOL}{POPT} \le 2$ for all problem instances. Since $NOPT \ge POPT$ for all problem instances,

$\frac{GSOL}{NOPT} \leq \frac{GSOL}{POPT} \leq 2$ for all problem instances. Therefore, 2 is a worst case error bound for the Greedy algorithm. □

Corollary 3.3: Greedy Algorithm's worst case error bound of 2 is tight for the 2-machine case.

Proof: Now consider the problem instance with 4 jobs with ($p_1=\varepsilon$, $p_2=\varepsilon$, $p_3=2\varepsilon$, $p_4=s_2$), $4\varepsilon \leq s_1$, $7\varepsilon \leq s_2$ and $s_1 + \Delta = s_2$.
The Greedy algorithm produces the following solution.

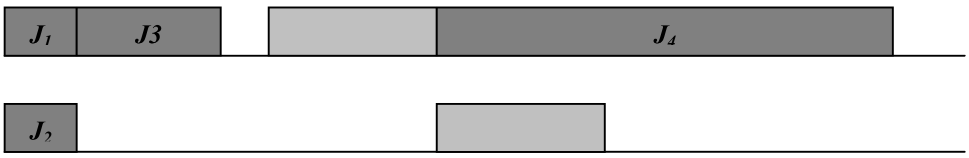

Figure 3-7: Worst case example of "Greedy"

with an objective value $5\varepsilon + 2s_2$. But optimal schedule has an objective value $7\varepsilon + s_2$. Then the ratio between *NOPT* and *GSOL* is:

$$\frac{5\varepsilon + 2s_2}{7\varepsilon + s_2}.$$

$\lim_{\varepsilon \to 0} \frac{5\varepsilon + 2s_2}{7\varepsilon + s_2} = 2$ is the error bound for this instance, and this equals to the worst-case bound of the problem. From this result we can conclude that 2 is the worst-case bound of Greedy algorithm and it is tight. □

Theorem 3.2: The Greedy algorithm has a worst-case error bound of 3/2 on the 2-parallel machine problem case, which has only one period of unavailability.

Proof: In the worst case of this type of problem, two criteria must be hold. First, there is no job assignment to the available period before unavailability period, and, second, the number of jobs assigned to the continuously available period is maximized. To hold these criteria, $p_1 > s$ where t is the starting time of unavailability, and $\sum_{j=1}^{n-1} p_j \leq s + \Delta$.

Under these conditions, the solution obtained from Greedy algorithm is
$n \times p_1 + (n-1) \times p_2 + ... + p_n$ (3.9),
and solution obtained from *SRPT*,
$p_1 + (n-1) \times (p_1 + p_2 - s) + (n-2) \times p_3 + ... + p_n =$
$n \times p_1 + (n-1) \times (p_2 - s) + (n-2) \times p_3 + ... + p_n$ (3.10). If we divide (3.9) to (3.10), we obtain,
$\frac{n \times p_1 + (n-1) \times p_2 + ... + p_n}{n \times p_1 + (n-1) \times (p_2 - s) + (n-2) \times p_3 + ... + p_n}$ for $t < p_1 \leq p_2 \leq \ldots \leq p_n$.
This ratio is largest
where $s \to p_1, p_2 \to p_1, \ldots, p_n \to p_1$. Then the ratio becomes

$\frac{\frac{1}{2} n(n+1) p_1}{\frac{1}{2} n(n+1) p_1 - (n-1) p_1}$ and this ratio is maximum at $n=2$, or $n=3$, that is 3/2.

We know that $\frac{GSOL}{POPT} \leq \frac{3}{2}$ for all problem instances from above result. Since $NOPT \geq POPT$ for all problem instances,
$\frac{GSOL}{NOPT} \leq \frac{GSOL}{POPT} \leq \frac{3}{2}$ for all problem instances. Therefore, 3/2 is a worst case upper bound to Greedy algorithm.

Now consider the problem instance with 5 jobs with ($p_1=\varepsilon$, $p_2=\varepsilon$, $p_3=\varepsilon$, $p_4=t$, $p_5=s + \varepsilon$), and $3\varepsilon \leq s$, the greedy algorithm gives a solution of $\sum C_j = 3s + 10\varepsilon$ while optimal solution is $2s + 10\varepsilon$. Then the ratio is $\frac{3s+10\varepsilon}{2s+10\varepsilon}$. While $\varepsilon\to 0$, the ratio is 3/2. Hence, 3/2 is a tight upper bound for Greedy. □

Corollary 3.4: A more general worst-case error bound for the problems that every machine does not have unavailability periods can be written. Let u be the number of machines that have unavailability periods on it. Then worst-case bound of this algorithm is $1+\frac{u}{m}$. This can be proved by the same approach used in Theorem 2.

3.4.2.2. Neighborhood Search Algorithm

This algorithm is based on searching for improving solutions from a neighborhood of a given initial feasible solution. Since searching entire solution space is too costly, a small neighborhood of a solution is determined and a search is performed on this neighborhood. An initial feasible solution, a neighborhood structure, and a search structure between the neighborhoods of a feasible solution are defined to construct this neighborhood search mechanism.

Neighborhood Definition

As mentioned before assigning job to intervals are preferred rather than assigning jobs to machines while defining neighborhood structure. Since jobs assigned to each interval of machine availability are to be processed in SPT order, a feasible assignment of all jobs to these intervals constitutes a solution. Within this framework, the neighborhood of a given feasible solution is defined as the set of new solutions that can be generated through two-, three-, and four-way job exchanges among intervals of machine availability. Two-way job exchanges consist of one to two, three-way job exchanges consist of one to two, and four-way job exchanges consist of two to two job exchanges between 2*m* intervals defined in section 2.

Whole neighborhood is not searched in only one iteration. To make this search easier and more efficient we divide the whole neighborhood of a feasible solution into sub-neighborhoods. A sub-neighborhood consists of only job exchanges between only fixed 2 intervals. Actually a sub-neighborhood is a neighborhood defined only between two intervals. Local optimum of a sub-neighborhood of a feasible solution is the minimum solution that can be found by making two-, three-, and four-way job exchanges between the corresponding two intervals. In a particular *m*-machine problem, there are $C(m,2)$ sub-neighborhoods.

For two-machine problem case, in addition to these sub-neighborhoods, 4 dummy sub-neighborhoods are generated for fine-tuning of the algorithm. These 4 sub-neighborhoods search job exchanges between 3 intervals. For a particular two-machine case these intervals are:
[from 2 to 1 and from 1 to 4], [from 4 to 1 and from 1 to 2], [from 2 to 3 and from 3 to 4], [from 4 to 3 and from 3 to 2].
Totally 10 sub-neighborhoods are generated to search the solution space.

The sub-neighborhoods are searched sequentially until all sub-neighborhoods reach their local-optimum that is no two-, three-, or four-way job exchange can be done.

Finding an Initial Solution

The solution found from the constructive algorithm is used as the starting solution for the NS algorithm. Also, the solution found from the NS algorithm is used as the initial feasible solution in the branch-and-bound algorithm.

Job Exchange in a Neighborhood Move:

A neighborhood move involves selecting one or two jobs from an interval and interchanging them by one or two jobs from another interval. Jobs in an interval are considered for an exchange in SPT order, and only those exchanges that would result in a reduction in the $\sum C_j$ value are executed. When an exchange is executed, jobs in the affected intervals are reordered to maintain the SPT sequence, and the process repeats itself starting from the jobs with the shortest processing time within each interval until all job pairs are considered for exchange with no exchange being executed. Always two jobs are selected from both intervals for an exchange. To allow two- and three-way exchanges one dummy job with processing time 0 is added to each interval. The completion times of these dummy jobs are not considered while calculating the objective value.

Fact 2: Optimal schedule for a given instance has a total idle time in interval 1 and 3 $(I_1^* + I_3^*)$ that is smaller than or equal to the initial cumulative idle times on interval 1 and interval 3. That is;

$I_1^* + I_3^* \leq I_1 + I_3$

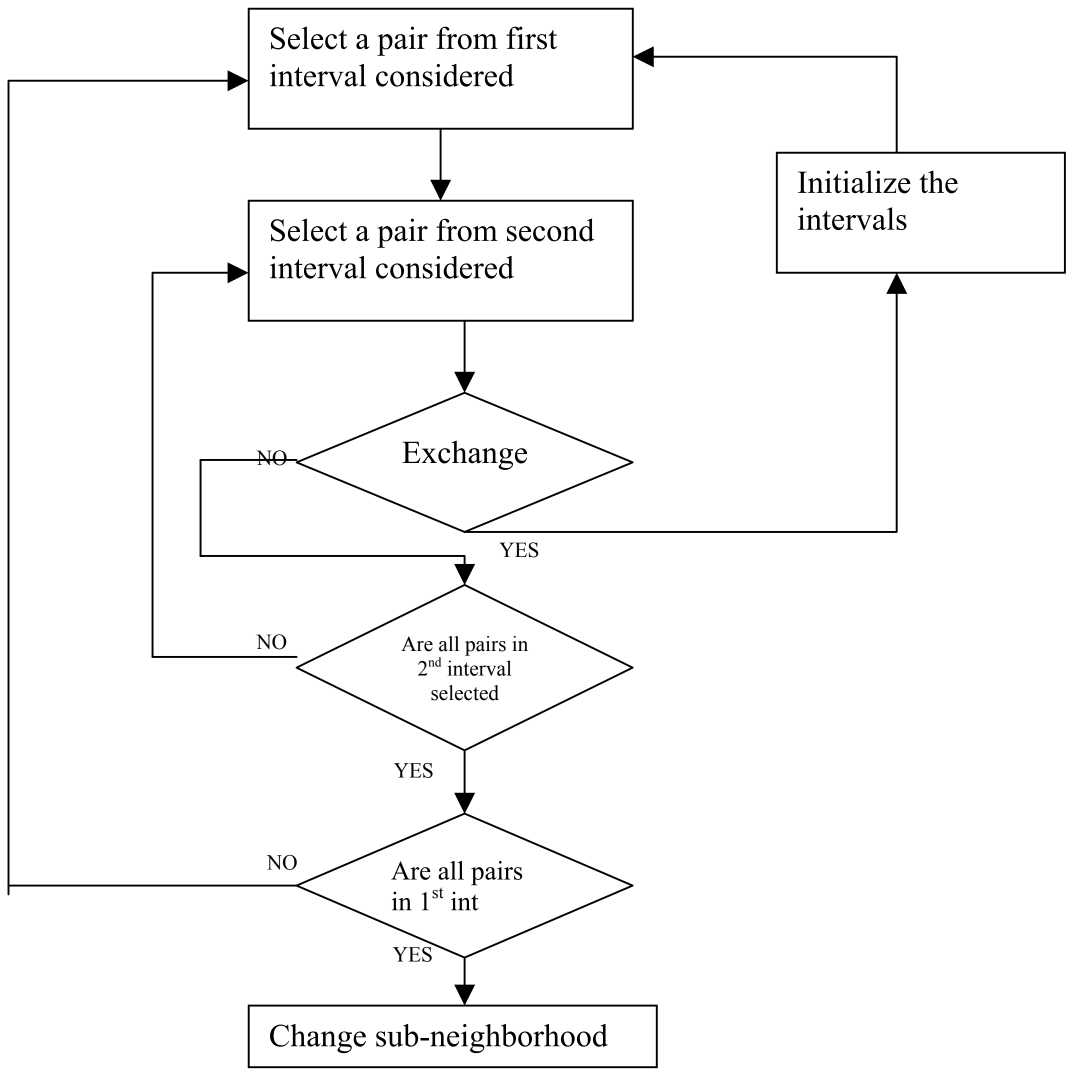
Select a pair from first interval considered
Initialize the intervals
Select a pair from second interval considered
Exchange
NO
YES
Are all pairs in 2nd interval selected
NO
YES
Are all pairs in 1st int
NO
YES
Change sub-neighborhood

Figure 3-8 Representation of NS algorithm

Thus, job exchanges should decrease idle time in intervals 1 and 3. This observation helps accelerate the search process by making the size of the sub-neighborhood and whole neighborhood smaller. As an exception, only job exchanges between interval 1 and interval 3 can violate this rule because, the process should increase at least one of the idle times of these intervals while exchanging jobs between them.

Example 3.1: Consider two-parallel machine case and the following 10-job problem with processing times as shown in Table 3-1, and without loss of generality assume that jobs are indexed in SPT order.

Table 3-1: Processing times of jobs

job *i*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
p_i	16	29	31	34	46	49	54	60	66	92

Starting time of unavailability in the first machine is 50, that in the second machine is 100, and duration of unavailability on both machines is 50. Define sch_i is the assignment of job i that is the interval job i assigned.

The NS algorithm starts with the initial solution shown in Table 3-2.

Table 3-2: Initial solution

job *i*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
sch_i	1	3	3	1	2	2	2	4	2	4

The $\sum C_j$ value for this initial solution is 1572. Once again, since the SPT order is maintained within each interval, the search process consists in reassignment of certain jobs to different intervals. A graphical illustration of the initial schedule given in Table 3-2 is shown in Figure 3-8.

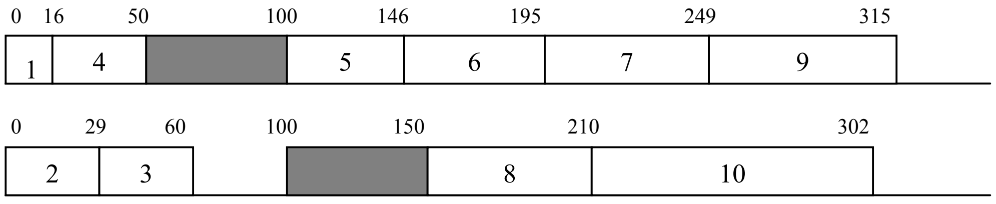

Figure 3-9: Graphical illustration of initial schedule

No idle time is incurred on machine 1, and the total idle time on machine 2 is 40 time units in interval 3.

The Neighborhood Search algorithm looks for a one- or two-job exchange between sub-neighborhoods defined above. Job exchanges are performed by selecting two jobs from each interval and interchanging their assigned intervals. Initially we add a dummy job to each interval with processing time 0 to allow two- and three- way job exchanges. Let d1, d2, d3, and d4 be the dummy jobs numbered with respect to the interval they assigned. The algorithm starts by selecting the sub-neighborhood that considers only the job exchanges between interval 1 and 3. Then it considers all possible job pairs for a potential exchange operation. Since no job exchanges result in a reduction in the $\sum C_j$ value, the algorithm passes on to sub-neighborhood that considers the job exchanges between intervals 2 and 3. Then it selects jobs d2, and 5 from interval 2 and jobs d3 and 2 from interval 3 for an exchange operation. Since this exchange results in both an improvement in the objective function value, and a reduction in the idle time on machine 2, it is accepted and new schedule becomes as shown in Table 3-3.

Table 3-3: Schedule obtained after the first move

job *i*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
sch *i*	1	2	3	1	3	2	2	4	2	4

A graphical illustration of this new schedule is displayed in Figure 3-10.

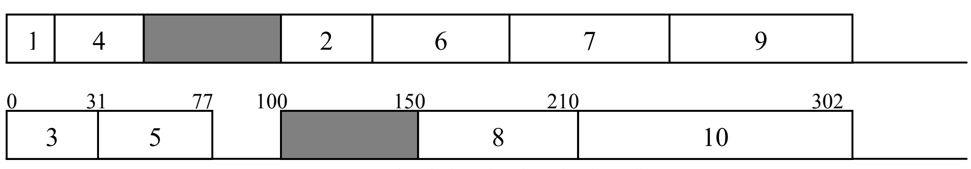

Figure 3-10: Schedule obtained after first move

The $\sum C_j$ value of this new schedule is 1523. The search mechanism is then recentered around this solution and the process restarts by taking the first two jobs from each interval. These jobs are jobs d2 and 2 from interval 2 and jobs d3 and 3 from interval 3. Exchanging these particular jobs does not yield a better solution, and hence this exchange is rejected. Then jobs in the 1st and the 3rd place from interval 2 (d2 and job 6) and first two jobs from interval 3 (d3 and job3) are considered. It is calculated that exchanging these jobs yields a feasible solution and improves the objective function and also this exchange decreases the idle time on machine 2. Then this exchange is accepted and new schedule becomes

Table 3-4: Schedule obtained after the second move

job *i*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
sch_i	1	2	2	1	3	3	2	4	2	4

And the assignments graphically become

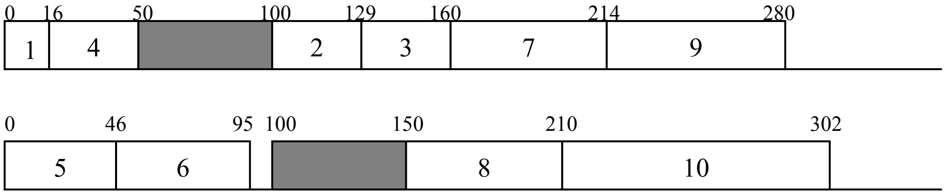

Figure 3-11: Schedule obtained after second move

New objective value is 1502. The algorithm initializes the search and starts with the first two jobs from the interval. When search is completed in sub-neighborhood intervals between 2 and 3, the algorithm continues with sub-neighborhoods between intervals 1-4, 1-2, 3-4, 2-1-4, 4-1-2, 2-3-4, 4-3-2, and 2-4. When no exchange can be found within these sub-neighborhoods the algorithm turns to sub-neighborhood 1-3 and starts searching for improving moves. When there is no improving move in all of the 10 sub-neighborhoods the algorithm stops. In our example NS algorithm gives 1497 as the objective value at the end, that is found in branch-and-bound algorithm as the optimal value.

Computational Complexity

The algorithm generates $C(2m,2) + 2m \times C(m,2)$ sub-neighborhoods in a problem where m is the number of machines. Search of a sub-neighborhood has a complexity of $n^5 m \log m$ where n is the number of jobs. Hence, searching the whole neighborhood once has a complexity of $n^5 m^4 \log m$. The algorithm repeats searching whole neighborhood until no improving move can be found. Then this repeating action has a complexity of n^2, because $\sum C_j$ value cannot exceed $\frac{(n+1)n}{2} p_{\max}$ and every movement results in an integral decrease in $\sum C_j$ value. Therefore, the *NS* algorithm has a computational complexity of $n^7 m^4 \log m$.

3.4.2.3. Simulated Annealing Application

This algorithm uses the same neighborhood structure adopted in algorithm *NS*. It is different, however, in that in addition to always accepting those moves that result in a reduction in the objective value, it also accepts moves that result in an increased $\sum C_j$ value with some probability. In this way, it allows for searching different valleys in an attempt to avoid entrapment at a local optimum.

The algorithm is a straightforward application of the classical approach proposed by Kirkpatrick *et al.* [28]. Recall that in algorithm NS, a job exchange is accepted only if:

1. it decreases the idle time on machines 1 and 2, and
2. it results in a smaller objective value.

The simulated annealing algorithm entirely disregards the first condition while extending the second condition by also allowing moves that increase the objective value with a certain probability. In particular, a move that results in an increase in the $\sum C_j$ value can be accepted with a probability of $e^{-\frac{\Delta t}{t}}$ where Δt is the magnitude of the resulting increase and t is the current temperature of the system.

The parameters of the simulated annealing algorithm such as the initial temperature, cooling function, and maximum number of iterations are set based on preliminary experimentation. The initial temperature is set as 100. A linear cooling function $f(t)=0.99t$ is adopted. The maximum number of iterations to be performed at a given temperature is set at 100 for each neighborhood. Finally, the system is considered frozen when temperature of the system fell below 0.1, that the algorithm is terminated when $t_{fin} \leq 0.1$. A pseudo code of the algorithm is shown below.

Let V be the set of neighborhoods constitutes the whole neighborhood and E_j's be the elements of this set,

```
(1) $W = \varnothing$;
(2) Select a sub-neighborhood $E_j$ s.t. $E_j \notin W$; $W=W \cup E_j$; set $i = 1$;
(3)Consider a job-pair exchange; $\Delta t :=$ change in the objective value;
(4) if $\Delta t < 0$, perform exchange; go to (7);
(5) $R \sim U(0,1)$; if $\Delta t > 0$ and $e^{-(\Delta t/t)} > R$, go to (7);
(6) Reject exchange; if all pairs are not considered go to (3); else go to (9);
(7) Perform exchange; $i = i + 1$;
(8) If $i < iterations$, go to (3);
(9) if $W = V$, $t = 0.99t$; go to (10), else go to (2) ;
(10) if $t \leq 0.1$, exit;
(11) Go to (1);
```

3.5. COMPUTATIONAL EXPERIMENTATION

In this section, we develop an experimental design framework and analyze the performance of the proposed algorithms via computational experimentation on the 2-machine case of the problem.

3.5.1. EXPERIMENTAL DESIGN

Processing times of jobs are generated from a uniform distribution between 1 and 100, and the duration of each period of unavailability is set equal to the average processing time of 50.

Starting times of unavailability periods in each machine is taken as experiment variable. The starting time of the period of unavailability on each machine is set systematically as early, medium, and late as shown in Table 3-5. For the first machine, early start of unavailability is $5n$, medium start of unavailability is $25n/2$, and late start of unavailability is $20n$ where n is the number of jobs. For the second machine, early start of unavailability is immediately after the end of period of unavailability on the first machine. Medium start of unavailability gives a gap of two unavailability durations after the end of the unavailability period on the first machine and late start of unavailability gives a gap of four unavailability durations after the end of the unavailability period in the first machine. The starting times and durations of unavailability periods are:

Table 3-5: Experimental scheme of unavailability periods

First machine	Second Machine	Start in First Machine	Start in Second Machine
Early	Early	$5n$	$6n$
	Medium		$8n$
	Late		$10n$
Medium	Early	$12.5n$	$13.5n$
	Medium		$15.5n$
	Late		$17.5n$
Late	Early	$20n$	$21n$
	Medium		$23n$
	Late		$25n$

3.5.2. COMPUTATIONAL RESULTS

All algorithms are implemented in C programming language and run on an IBM compatible PC with Celeron 1.2 Mhz CPU. Small-, medium-, and large-sized problems having 30, 50, and 70 jobs, respectively, are tested in the experimental scheme. All CPU times for the NS algorithm are smaller than 2 seconds. In the small-sized problems for which the optimum solution could be obtained, the heuristic results are compared with the optimal.

Table 3-6: Branch-and-bound results

Start of unavailability on the first machine	**Start of unavailability on the second machine**	**Number of solved problems**	**% Deviation from the lower bound**
Early	**Early**	5 of 5	0.65
	Medium	5 of 5	0.48
	Late	5 of 5	0.06
Medium	**Early**	3 of 5	0.71
	Medium	5 of 5	0.39
	Late	5 of 5	0.38
Late	**Early**	2 of 5	0.54
	Medium	3 of 5	0.10
	Late	4 of 5	0.04

For medium and large-sized problems, for which the optimum results are unavailable in a 30-minute CPU time, performance evaluation is performed based on the lower bound. The table 3-6 gives the results of Branch-and-bound algorithm in 30-job problems. In small-sized problems with up to 30 jobs, 37 of the 45 instances are solved optimally by the branch-and-bound algorithm. Six of the unsolved instances have late period of unavailability on the first machine. The other two unsolved instances are with medium and early unavailability periods on machines 1 and 2, respectively. Considering the large number of possible combinations of jobs that can together be assigned to the intervals preceding the periods of unavailability on each machine, the poor performance of the branch-and-bound in solving problems with late periods of unavailability should not be surprising. The similar difficulty in solving problems with medium and early periods of unavailability on machines 1 and 2 can be explained by the relatively poor quality of both the lower bound and the heuristic solution fed in to the algorithm as an initial upper bound. Table 3-6 suggests that given the starting time of the unavailability on machine 1, the performance of the lower bound improves as the unavailability of machine 2 is delayed further. This trend is in line with the two observations made in Section 3.3. In addition, the number of unsolved instances shown in Table 3-6 closely follows this trend, that is, the branch-and-bound solves those instances, for which a better lower bound is available, more easily. The average percentage deviation of the lower bound from the optimum in the total 37 solved instances is less than 1%. This small deviation suggests that the lower bound is tight enough to allow for giving insights about performances of heuristic algorithms for medium- and large-sized problems for which the optimum results are unavailable.

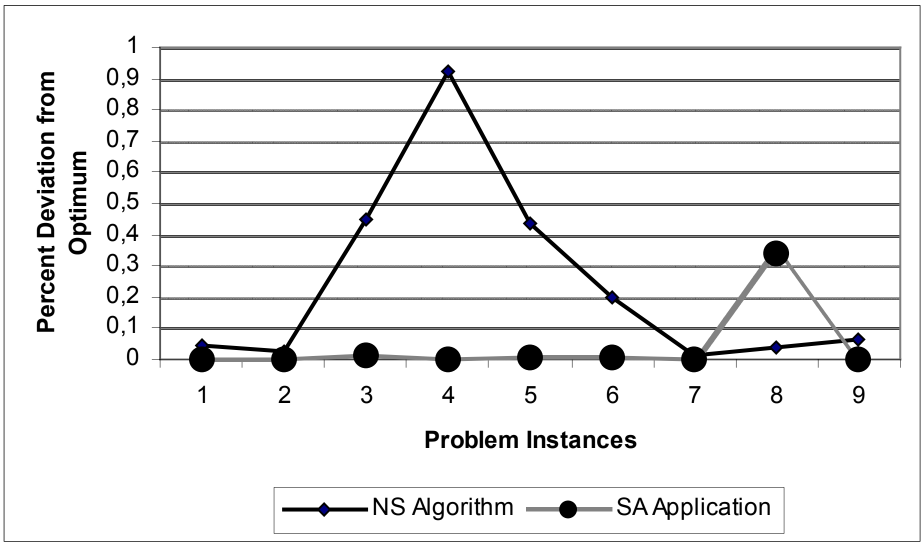

Figure 3-12: Heuristic performance against the optimum in 30-job problems

Table 3-7: Results of heuristic algorithms for 30 jobs

Start of unavailability on the first machine	Start of unavailability on the second machine	Greedy percentage deviation from optimum	NS percentage deviation from optimum	SA percentage deviation from optimum	Average SA time (second)
	Early	1,12	0.04	0	261.7
Early	Medium	0,79	0.02	0	323.2
	Late	2,28	0.45	0.1	323.5
	Early	1,51	0.92	0.003	339
Medium	Medium	1,71	0.43	0.007	313
	Late	1,25	0.20	0.005	304.9
	Early	0,08	0.01	0	260.1
Late	Medium	0,57	0.04	0.3	244.2
	Late	0,41	0.06	0	232.2

Table 3-8 Results of heuristic algorithms for 50 jobs

Start of unavailability in first machine	Start of unavailability in second machine	Greedy percentage deviation from LB	NS percentage deviation from LB	SA percentage deviation from LB	Average SA time (seconds)
Early	Early	1,55	0.51	0.24	2039.8
	Medium	1,75	0.39	0.26	2084.7
	Late	1,41	0.17	0.17	2070.6
Medium	Early	0,7	0.42	0.45	1893,0
	Medium	1,7	0.79	0.82	1786,2
	Late	0,86	0.41	0.27	1788,3
Late	Early	0,53	0.45	0.46	1174,1
	Medium	0,51	0.41	0.43	1043,4
	Late	0,39	0.28	0.29	923,1

Tables 3-7, 3-8 and 3-9 show the performance of the three heuristics for the 30-, 50- and 70-job problems, respectively. The neighborhood search algorithm performs very well in all cases. The algorithm never exceeds a gap of 1% from the optimum in the 30-job case, and from the lower bound in the 50- and 70-job cases. For the 30-job problems, it gives less than 0.5% gap for eight of the nine parameter combinations, and less than 0.1% gap for five of the nine parameter combinations from the optimum solution. The simulated annealing algorithm also produces excellent results in this experimental scheme. It gives less than 1% gap in all 27 problem instances. Simulated Annealing Application produces nearly optimal solutions for 30-job problems, and outperforms Neighborhood Search Algorithm. However, Neighborhood Search Algorithm gives nearly same solutions for 50-job problems and outperforms Simulated Annealing Application in almost every instance for 70-job problems. Extension of the neighborhood of a solution with high number of jobs causes improved results of Neighborhood Search Algorithm.

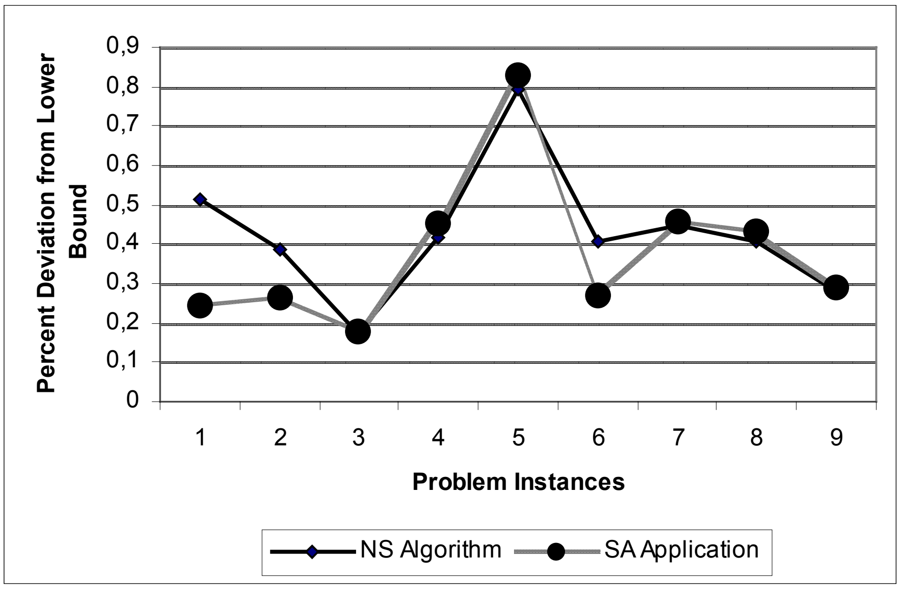

Figure 3-13: Heuristic performance against the lower bound in 50-job problems

Table 3-9: Results of heuristic algorithms for 70 jobs

Start of unavailability in first machine	Start of unavailability in second machine	Greedy percent deviation from LB	NS percentage deviation from LB	SA percentage deviation from LB	Average SA time (seconds)
	Early	0,55	0.14	0.07	6487,2
Early	Medium	0,63	0.18	0.20	6610,4
	Late	0,93	0.16	0.21	6628,0
	Early	0,93	0.48	0.62	5343,2
Medium	Medium	1,04	0.45	0.57	5123,2
	Late	0,36	0.18	0.44	5114,4
	Early	0,49	0.36	0.29	2897,6
Late	Medium	0,48	0.38	0.33	2626,4
	Late	0,43	0.23	0.28	2292,8

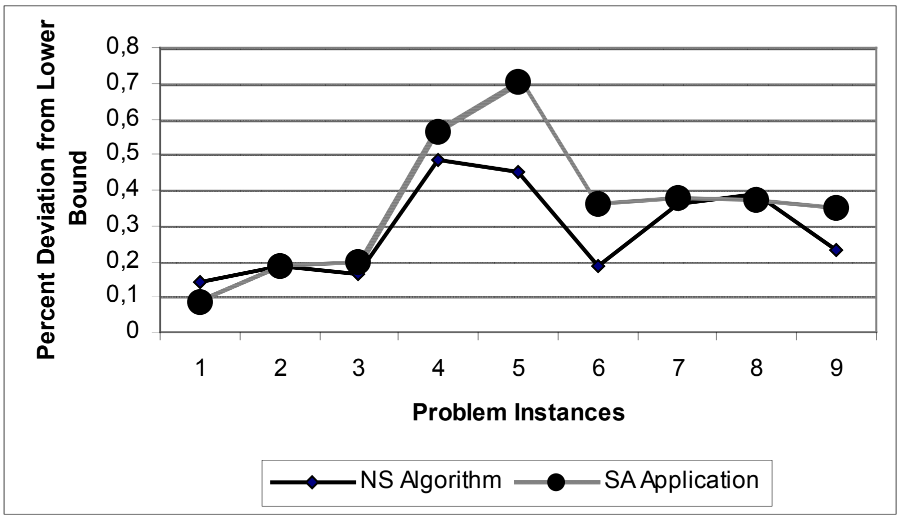

Figure 3-14: Heuristic performance against the lower bound in 70-job problems

Chapter 4

MAXIMUM COMPLETION TIME PROBLEM

In this section, we give the problem definition of minimizing $C_{\max}$ on parallel identical machines with availability constraints, exact and heuristic solution methods for this problem and computational results of proposed methods.

4.1. PROBLEM DEFINITION

This chapter considers parallel machine scheduling problem with the maximum completion time (makespan, $C_{\max}$) objective subject to periods of machine unavailability. Machines are identical. Each job should be processed on exactly one machine. All jobs are available at time zero. Processing times for all jobs are deterministic. Jobs are non-preemptive, that is if a job is interrupted while processing, it should start from the beginning. There is exactly one pre-determined unavailability period on each machine. Starting time and duration for the unavailability periods are known in advance, and machines may become unavailable simultaneously.

MATHEMATICAL FORMULATION

Consider a set $\mathcal{J}$ of n independent and non-preemptive jobs. Every member of this set will be processed on exactly one member of set $\mathcal{M}$ of m identical machines. Processing time of job J_k is denoted by p_k and is the same on each machine. Every machine M_h is subject to a period of unavailability with a length of $t_h - s_h$, where s_h is the starting time, and t_h is the ending time of unavailability period on machine h. In modeling this problem, each interval of availability is treated as a separate machine. In order to model those intervals that follow the periods of unavailability, m dummy jobs $J_{n+1},....,J_{n+m}$ are added to be scheduled on machines 1 through m. The processing times of these dummy jobs are set equal to $t_1,....,t_m$, respectively. Similarly, to model those intervals preceding the period of unavailability, m dummy machines, $M_{m+1},....,M_{2m}$, are defined. These machines are available for processing until times $s_1,....,s_m$, respectively.

An IP formulation for the problem is as follows. The decision variable x_{hk} is a binary integer variable which takes on a value of one if job k is processed on machine h, and zero otherwise.

min C_{max}

$$C_{\max} - \sum_{k=1}^{n+m} p_k x_{hk} \geq 0 \qquad \text{for } h=1,.....,m \qquad (4.1)$$

$$s_h - \sum_{i=1}^{n} p_k x_{hk} \geq 0 \qquad \text{for } h=m+1,....,2m \qquad (4.2)$$

$$\sum_{h=1}^{2m} x_{hk} = 1 \qquad \text{for } k=1,.....,n \qquad (4.3)$$

$$\sum_{h=1}^{m} x_{hk} = 1 \qquad \text{for } k=n+1,\dots,n+m \qquad (4.4)$$

$$\sum_{k=n+1}^{n+m} x_{hk} = 1 \qquad \text{for } h=1,\dots,m \qquad (4.5)$$

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

Once again, dummy jobs, $J_{n+1},\dots,J_{n+m}$, and dummy machines, $M_{m+1},\dots,M_{2m}$ are included to adequately model the respective periods of unavailability on machines 1 through m. Constraint set (4.1) sets the $C_{\max}$ equal to the completion time of the last job. Constraint set (4.2) forces job processing on machines $M_{m+1},\dots,M_{2m}$ to complete before the respective time limits of $s_1,\dots,s_m$. In practice this means jobs assigned before unavailability period should be completed before unavailability period begins. Constraint set (4.3) requires each original job to be processed on exactly one machine. The constraints in (4.4) and (4.5) are associated with periods of unavailability. (4.4) ensures that unavailability periods are completed and (4.5) ensures that only one unavailability period is assigned to each of the machines $M_1,\dots,M_m$.

In order to simplify this problem, we can make a slight modification in constraints (4.5) and (4.4). We know that machines are identical and every machine should have exactly only one unavailability period. Hence we can pre-assign dummy jobs raised from the unavailability periods to the original machines one by one and decrease the total number of jobs to be considered. Then we add new constraint (4.4') in the place of constraints (4.4) and (4.5) in the formulation.

$$x_{n+h,h} = 1 \qquad \text{for } h=1,\dots,m \qquad (4.4')$$

4.2. SOLUTION PROCEDURE

The IP formulation given in Section 4-1 is a modification of the model used by Mokotoff [46] for the parallel machine scheduling problem with continuous machine availability and a $C_{\max}$ objective. The modification arises from the dummy jobs and dummy machines related with unavailability periods. The dummy jobs are not processed on dummy machines. Since the constraints added due to the modification are in the form of strict equalities, the cutting plane scheme described in [46] remains applicable to the problem under consideration. Section 4.2.1 formally states this claim and provides a detailed mathematical proof.

4.2.1 CUTTING PLANE SCHEME

Following Mokotoff's [46] notation, let S be the feasible set corresponding to the LP relaxation of our MIP formulation. Consider $y^0 \in \mathbb{R}^+$ and define $S(y^0)$ as the subset of all points $(x,y^0) \in S$ where y^0 is the objective value which is revealed by x. Let P be the initial set of inequalities defining S, and Φ the set of valid inequalities for the convex hull of $S(y^0)$, conv($S(y^0)$). Finally, let $P(\Phi)$ be the current relaxation of conv($S(y^0)$) defined by the inequalities of P and inequalities of Φ.

Suppose x^0 is a point in $S(y^0)$. We define a valid inequality I such that $(x^0, y^0) \in P(\Phi \cup \{I\})$ if x^0 is integer. For each machine M_h, let A_h be the set of jobs partially or fully assigned to machine M_h including the dummy machines and dummy jobs according to assignment vector x^0. Also let $(x^*, s_h) \in S(y^0)$ and x^* be a binary (0/1) point and let

$B_h = \{j \in A_h \mid x^*_{hj} = 0\}$. Therefore, we can say that from the definition of B_h,

$$\sum_{j \in A_h} p_j x^*_{hj} = \sum_{j \in A_h - B_h} p_j = \sum_{j \in A_h} p_j - \sum_{j \in B_h} p_j \tag{4.6}$$

If either one of the two constraints corresponding to M_h in sets (4.1) and (4.2) is satisfied as a strict equality by (x^0, y^0), then we call that particular constraint active. We define the excess load Δ_h for a machine h as the difference between the summation of processing times of jobs assigned to M_h and available processing time for M_h that is $\Delta_h = \sum_{j \in A_h} p_j - y^0$ for h=1,...,m and $\Delta_h = \sum_{j \in A_h} p_j - s_h$ for h=m+1,.....,2m. From the definition of Δ_h for h=m+1,…,2m it can be easily seen that

$$\sum_{j \in A_h} p_j = s_h + \Delta_h \tag{4.7}$$

if we insert this equation into equation (4.6),

$$\sum_{j \in A_h - B_h} p_j x_{hj} = s_h + \Delta_h - \sum_{j \in B_h} p_j = s_h - \left(\sum_{j \in B_h} p_j - \Delta_h \right) \tag{4.8}$$

Since $\sum_{\forall j} p_j x^*_{hj} \le s_h$,

$\sum_{j \in A_h} p_j x^*_{hj} \le s_h$, then we have that $\sum_{j \in B_h} p_j - \Delta_h \ge 0$. (4.9)

Therefore, we have that

$$\begin{aligned} \sum_{j \in A_h} p_j x^*_{hj} &= s_h - \left(\sum_{j \in B_h} p_j - \Delta_h \right)^+ \\ &\le s_h - \sum_{j \in B_h} \left(p_j - \Delta_h \right)^+ . \end{aligned} \tag{4.10}$$

where $(k)^+ = \max\{0, k\}$ for $k \in \mathbb{R}$.

Moreover $1 - x^*_{hj} = 1$ if $j \in B_h$, and $1 - x^*_{hj} = 0$ if $j \in A_h - B_h$, we get

$$\sum_{j \in B_h} (p_j - \Delta_h)^+ = \sum_{j \in A_h} (p_j - \Delta_h)^+ (1 - x^*_{hj}) \tag{4.11}$$

inserting this equality into equality (4.10) we have

$$\sum_{j \in A_h} p_j x^*_{hj} \le s_h - \sum_{j \in A_h} (p_j - \Delta_h)^+ (1 - x^*_{hj}). \tag{4.12}$$

This inequality leads us a useful result for our problem.

Proposition 4.1: Let (x^0, y^0) be a point of $P(\Phi)$ and assume that the machine constraint for M_h is active for (x^0, y^0). Then the linear inequality

$$\sum_{j \in A_h} p_j x_{hj} \le y^0 - \sum_{j \in A_h} (p_j - \Delta_h)^+ (1 - x_{hj}) \tag{4.13}$$

for h=1,....,m and

$$\sum_{j \in A_h} p_j x_{hj} \le s_h - \sum_{j \in A_h} (p_j - \Delta_h)^+ (1 - x_{hj}) \tag{4.14}$$

for h=m+1,.....,$2m$ are valid inequalities. Moreover if there is a job J_j such that $p_j > \Delta_h$ and $x^0_{hj} < 1$, then the inequality is not satisfied by (x^0, y^0).

Proof: Proof for (4.13) is given in [46]. For inequality (4.14), we have shown above that a binary solution should satisfy this inequality. Hence, this is a valid inequality. Moreover, let $A^{\iota}_h = \{j \in A_h \mid p_j > \Delta_h\}$ be the subset of jobs whose processing times are greater than the excess load on M_h.

Assume now that there is a job $j \in A^{\iota}_h$ such that $p_j > \Delta_h$ and $0 < x^0_{hj} < 1$, then

$$\sum_{j \in A_h} (p_j - \Delta_h)^+ (1 - x^0_{hj}) > 0.$$

Since $\sum_{j \in A_h} p_j x_{hj}^0 = s_h$, the current solution does not satisfy the inequality.

□

4.2.2. EXACT SOLUTION

We develop an exact solution procedure by using the cutting plane scheme obtained in the previous section. This procedure consists in integration of valid cuts to the LP formulation of the problem in an iterative manner until either no more cuts can be added or a binary integer solution is obtained. Initially, a pre-determined objective value is inserted to the formulation. After obtaining the solution for this value, cuts for this solution are added. At the beginning of the algorithm, the initial objective value is found by the wrap-around rule, which is proposed in [43]. In addition, this is a Lower Bound for the optimal value. If no more cuts can be added for this solution value, this lower bound value is updated by increasing by one and new cuts are searched for the new solution value.

4.2.2.1. Lower Bound

Commonly used Wrap-Around rule for preemptive job case is used as the LB for this problem. This can be summarized as:

$$C_{\max} = \max\left\{ \frac{1}{m}\left(\sum_{j=1}^{n+m} p_j - \sum_{h=1}^{m} s_h \right); \max_j p_j \right\}$$

In addition, this lower bound can be improved by:

$$C_{\max} \geq p_{2m} + p_{2m+1}$$

Then our lower bound turns out to be:

$$C_{\max} = \max\left\{\frac{1}{m}\left(\sum_{j=1}^{n+m} p_j - \sum_{h=1}^{m} s_h\right); \max_j p_j; p_{2m} + p_{2m+1}\right\}$$

The lower bound function returning an integer value is denoted by $LB(n,m,p,U)$ for a given problem, where n represents the jobs, m represents the machines, p represents the processing times of jobs and $\mathcal{U}$ represents the unavailability periods in the problem.

4.2.2.2. The Exact Algorithm

An exact cutting plane algorithm ($C_{\max}(\mathcal{J},\mathcal{U})$) that uses the valid inequality defined in the previous sections is given below. Define LP(LB,Φ) is the LP relaxation of the integration of the original problem and the current set of added cuts

```
$C_{\max}(\mathcal{J},\mathcal{U})$ :

LB := $LB(n,m,p,U)$ - 1;
improvelowerbound: $LB = LB+1$ ; $\Phi = \varnothing$;
    truncate: Solve LP(LB,Φ);
    if LP(LB,Φ) has no solution then go to improvelowerbound;
    if solution $x$(LB,Φ) is a binary variable then return LB;
    $I := FindNewCut(\text{LB}, \Phi, x(\text{LB}, \Phi))$ ;
    if $I = \varnothing$, then return ($MIP$(LB));
    $\Phi = \Phi \cup \{I\}$;
    go to truncate;
```

$LB(n,m,p,U)$ function gives the initial lower bound for the problem. The main idea of this exact algorithm is to search whether an integral solution exists for a given objective value. LP(LB,Φ) is the linear programming problem obtained by the linear relaxation of the binary variables x_{ik}, and adding the lower bound value LB and the cut set Φ obtained from valid inequalities defined in section 4.2.1. LB is inserted as the objective value ($C_{\max}$ = LB) to the mathematical formulation. Then, the cutting plane algorithm searches for an integral solution for this objective value.

Set Φ is dynamic in the sense that, it is updated at the end of each iteration (i.e., each time a solution is obtained for the LP relaxation). Inserting the LB as the objective value to the mathematical formulation and solving the LP relaxation of the problem may result in two possible outcomes: a feasible solution is found or the problem classified as infeasible. If a feasible solution is obtained, it is checked whether it is an integral solution. If it is an integral solution, the algorithm reports this solution as the optimum and stops. If not, the algorithm searches for new cuts (*I*). If new cuts are found, the cut set Φ is updated by adding these new cuts ($\Phi = \Phi \cup I$). If no new cuts are found for the current LP relaxation of the problem, then the *MIP* (Mixed Integer Programming) solver is invoked for the original problem where the LB value is inserted as a lower bound (i.e., $C_{\max} \geq \text{LB}$). In the second possible outcome that is if LP relaxation of the problem is classified as infeasible, the LB value is increased by 1 unit (LB = LB + 1), the cut set is emptied ($\Phi = \emptyset$), and the above steps are repeated until either an integral solution is found or the MIP formulation is classified as infeasible. Since, when the inserted $C_{\max}$ value become infeasible, the $C_{\max}$ value is increased by 1 starting from the lower bound we found, the algorithm finds the optimal value by adding cuts to the solution in the optimal value or if it cannot add any valid cuts after some point it finds the optimal by sending the problem to a MIP solver. Schematic view of the algorithm can be seen from Figure 4-1 below:

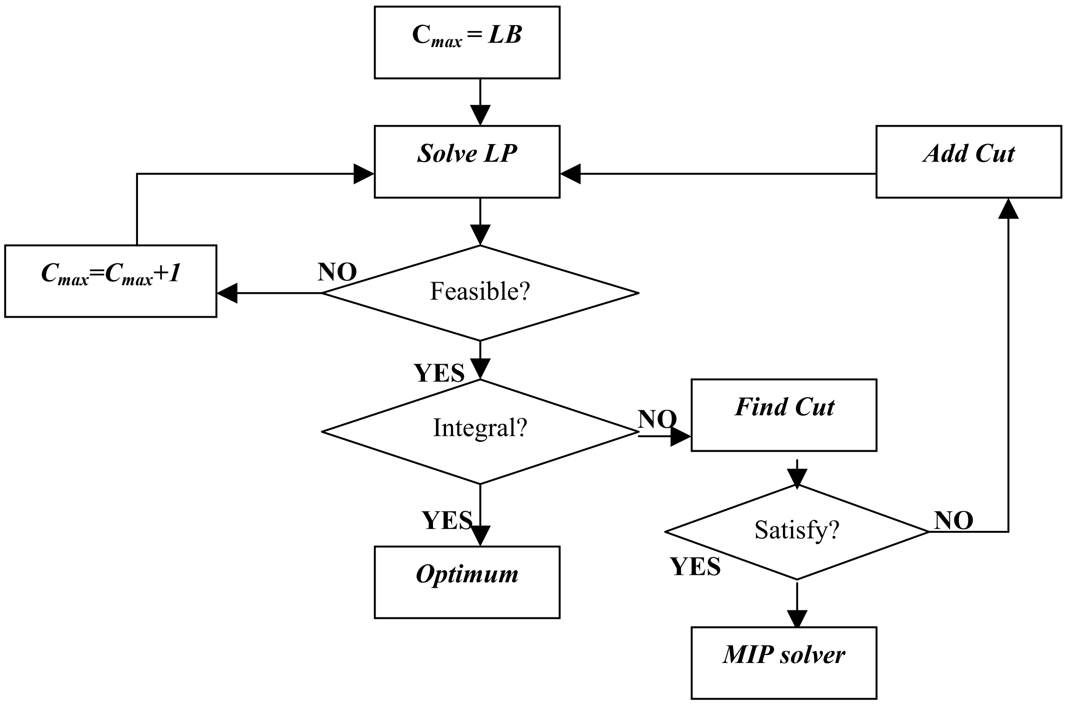

Figure 4-1: Schematic view of exact algorithm

Example 4.1:

Consider jobs 1, 2 and 3 with processing times of 3, 7 and 8, respectively. These jobs should be processed in a two-parallel machine setting in which each machine becomes unavailable for 5 time units. The unavailability period starts with machine 1 at time 3 and proceeds with machine 2 without interruption. In our formulation, this problem is modeled as a 5-job, 4-parallel machine problem where the processing times of the three original and two dummy jobs are 3, 7, 8, 8 and 13, and machines 4 and 5 are available only until time 3 and 8, respectively. Constraint (4.4') in the mathematical formulation given in Section 4.1, immediately forces job 4 to be assigned to machine 1, and job 5 to machine 2 resulting in the following partial schedule.

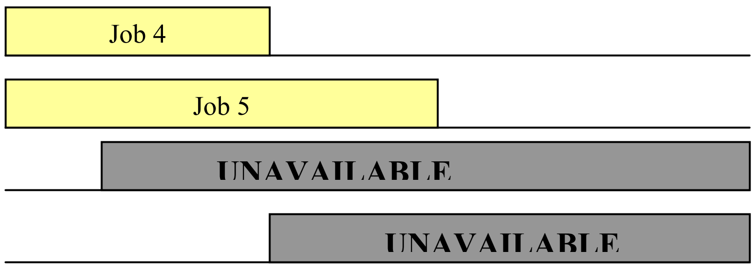

Figure 4-2: Schematic representation of the partial schedule

The LP relaxation of this problem yields the following solution with an objective value of $C_{\max} = 14$.

$i \backslash j$	M_1	M_2	M_3	M_4
J_1	0	0	1	0
J_2	0.857	0.143	0	0
J_3	0	0	0	1
J_4	1	0	0	0
J_5	0	1	0	0

$A_1 = \{2,4\}$ and $A_2 = \{2,5\}$,

$A_1' = \{2,4\}$ and $A_2' = \{2,5\}$. Then we derive the cuts

$7\,x_{12} + 8\,x_{14} \leq 14 - [6(1 - x_{12}) + 7(1 - x_{14})] \quad \Rightarrow \quad x_{12} + x_{14} \leq 1$

and,

$7\,x_{22} + 13\,x_{25} \leq 14 - [(1 - x_{22}) + 7(1 - x_{25})] \quad \Rightarrow \quad 6\,x_{22} + 6\,x_{25} \leq 6.$

With these two cuts added to the constraint set of the LP relaxation, the following solution is obtained.

$i \setminus j$	M_1	M_2	M_3	M_4
J_1	0	0	1	0
J_2	0	0	0	1
J_3	0.75	0.125	0	0.125
J_4	1	0	0	0
J_5	0	1	0	0

Based on this solution we derive the following three cuts.

$$8\,x_{13} + 8\,x_{14} \leq 14 - [6(1 - x_{13}) + 6(1 - x_{14})] \quad \Rightarrow \quad 2\,x_{13} + 2\,x_{14} \leq 2$$

$$8\,x_{23} + 13\,x_{25} \leq 14 - [(1 - x_{23}) + 6(1 - x_{25})] \quad \Rightarrow \quad 7\,x_{23} + 7\,x_{25} \leq 7$$

$$7\,x_{42} + 8\,x_{43} \leq 8 - [0(1 - x_{42}) + (1 - x_{43})] \quad \Rightarrow \quad 7\,x_{42} + 7\,x_{43} \leq 7$$

Adding these three cuts to the LP relaxation renders the problem infeasible. Thus, the solution value $C_{\max}$ is incremented by one ($C_{\max}$ = 14 +1) and the cut set is emptied. Solving the LP relaxation with the incremented $C_{\max}$ value provides the following solution.

$i \setminus j$	M_1	M_2	M_3	M_4
J_1	0	0	1	0
J_2	1	0	0	0
J_3	0	0	0	1
J_4	1	0	0	0
J_5	0	1	0	0

Since this is a binary solution, the algorithm reports it as the optimum and stops.

A Special Case

When problem instances for the 2-machine case with only one unavailability period are carefully examined, it can be seen that the algorithm can be simplified by doing some operations as done in the [40].

The exact algorithm defined above is designed to deal with more complex problem structures that every machine in the problem has exactly one unavailability period in scheduled time horizon.

In view of this fact, the exact algorithm is restructured for this particular two-machine experimental scheme with unavailability period only on one machine. Problems are classified into four cases based on the total processing time and the starting (s) and ending (t) times of the unavailability period. An alternative version of the algorithm is characterized for each case, and the appropriate version is applied to solve a given instance.

Case 1: $s = 0$;
In that case, the problem instance becomes a 2-parallel machine scheduling problem with continuous availability and with a dummy job with processing time t ($p_{n+1} = t$). The resulting IP formulation is as follows:

min $C_{\max}$

$$C_{\max} - \sum_{j=1}^{n+1} p_j x_{hj} \geq 0 \qquad \text{for } h = 1, 2 \qquad (4.15)$$

$$\sum_{h=1}^{2} x_{hj} = 1 \qquad \text{for } j = 1,2,\ldots,n+1 \qquad (4.16)$$

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

Case 2: $\sum_{j=1}^{n} p_j \le 2s$

That is, the last job on at least one of the two machines completes processing before time *s*. Hence, the problem reduces to a 2-parallel machine scheduling problem with continuous availability. We exploit this property, set both the starting (*s*) and the ending (*t*) times of the period of unavailability equal to 0 (i.e., *s* = 0, *t* = 0), and apply to the problem. The resulting IP formulation is as follows:

min $C_{\max}$

$$C_{\max} - \sum_{j=1}^{n} p_j x_{hj} \ge 0 \qquad \text{for } h = 1, 2 \qquad (4.17)$$

$$\sum_{h=1}^{2} x_{jh} = 1 \qquad \text{for } j = 1,2,\ldots,n+1 \qquad (4.18)$$

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

Case 3: $2s < \sum_{j=1}^{n} p_j \le s + t$

In this case, completion time of the last job on the continuously available machine is greater than *s*, and no jobs will be assigned to the other machine after the period of unavailability. This observation leads to the following proposition.

Define $\overline{C}$ as the completion time of the last job assigned to the interval preceding the period of unavailability.

Proposition 4.2*:* Problem reduces to maximizing $\overline{C}$.

Proof: Obviously, it must be that $\overline{C} \leq t$. Since no jobs will be processed on the partially available machine after the period of unavailability, we also have that

$$\overline{C} + C_{\max} = \sum_{j=1}^{n} p_j$$

Therefore, $C_{\max}$ assumes its minimum value when $\overline{C}$ is at its maximum. Thus, the problem reduces to a knapsack problem. □

It is important to note that the Heuristic Algorithm, *Heur*($\mathcal{J},\mathcal{V}$), proposed in section 4.2.3 provides the optimum solution under Case 3.

Case 4: $s + t < \sum_{j=1}^{n} p_j$

Problems in this class do not render themselves to any particular simplification and hence Algorithm 1 should be applied as it to obtain the optimum solution.

Four cases defined above constitute a 2-machine problem with an availability constraint. To solve this type of problem, appropriate solution method is applied according to these cases.

4.2.3. HEURISTIC APPROACH

A heuristic approach is developed based on a divide-and-conquer philosophy to obtain good solutions for the problem in reasonable CPU times. In particular, the problem is divided into *m* sub-problems each having only one interval for processing, each one of these sub-problems is solved in isolation, jobs assigned in this initial phase are eliminated from the list of jobs to be scheduled, and an a *m*-parallel machine scheduling problem is solved for the remaining jobs. Computational results given in Chapter 5 provide evidence for an acceptable tradeoff between the gain in computational time and deviation from optimality.

Figure 4-2 depicts *m* intervals of machine availability preceding each period of unavailability on an *m*-machine problem. Let U_h be the length of unavailability on machine *h* and D_h the idle time in interval *j* after all jobs are assigned to the intervals. We can conclude that

$$C_{\max} \geq \frac{1}{m}\left(\sum_{j=1}^{n} p_j + \sum_{h=1}^{m}(D_h + U_h)\right).$$

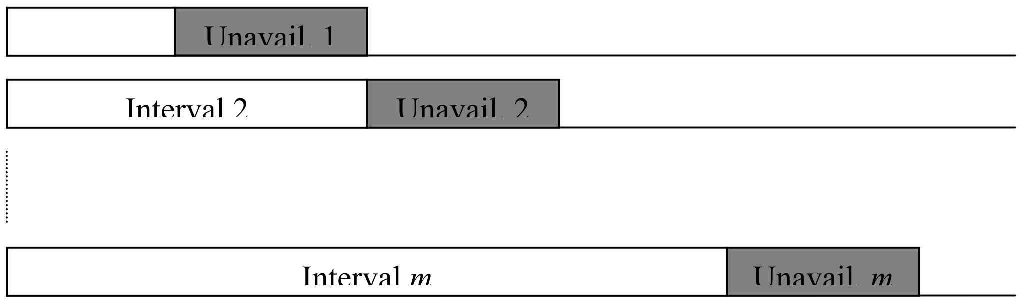

Figure 4-3 Availability intervals preceding unavailability periods

Processing times "p_j" and lengths of unavailability "U_h" on each machine are deterministic and known in advance. Hence, the only solution dependent components in the lower bound formulation are the idle times in intervals 1 through m. Consequently, the quality of this lower bound dependent upon an accurate estimation of the minimum total unavoidable idle time to be incurred in these intervals.

This general idea triggers a natural heuristic approach to the solution of the problem. In this approach, a knapsack problem is solved optimally for each one of the intervals $1,...,m$, to decide on the jobs to be assigned to these intervals. At each step, the jobs assigned to an interval are excluded from job list J, and finally after all knapsack problems are solved and jobs assigned to intervals 1 through m are removed from list J, a $C_{\max}$ problem is solved to schedule all unassigned jobs and m dummy jobs with processing times $t_1,...,t_m$ (to account for the point in time at which each machine becomes available after unavailability period) on m parallel identical machines using the exact algorithm $C_{\max}(\mathcal{J},\mathcal{U})$ defined in previous sections. By doing this, m knapsack problems and an ordinary parallel machine $C_{\max}$ problem with reduced job number are solved iteratively by the algorithm. Knapsack problems are solved in the increasing order of size of the knapsack to allow small sized jobs assigned to the small sized intervals. If larger sized knapsacks are solved formerly, small sized jobs may be assigned to these intervals and small sized intervals cannot be filled properly.

4.2.3.1. Knapsack Algorithm

Consider a simple knapsack problem,

max $p_1x_1 + \ldots\ldots + p_nx_n$

subject to

$p_1 x_1 + \ldots\ldots + p_n x_n \leq C$ (4.19)

where p_i's are the sizes of items to be inserted and C is the size of knapsack. We can see that inequality (4.2) in the mathematical formulation phase is very similar to the inequality (4.19). We can conclude from this similarity that $\sum_{j \in A_h} p_j x_j \leq s_h - \sum_{j \in A_h} (p_j - \Delta_h)^+ (1 - x_j)$ is a valid inequality for the knapsack problem. The proof of the idea above is the same as the proof of Proposition 4.1. We follow the same approach as in $C_{\max}(\mathcal{J}, \mathcal{V})$**,** and search for an integral solution for the *LP* relaxation of the knapsack problem before using an *MIP* solver. Let *Knap*(j,$\mathcal{E}$) be an exact algorithm to solve the knapsack problem corresponding to availability interval j and the set $\mathcal{E}$ of jobs assigned to any interval. Also let *LPKnap*(T_j,$\mathcal{E}$,Φ) be the solution to the LP relaxation of the problem and *MIPKnap*(T_j,$\mathcal{E}$,Φ) be the optimum integer solution for the problem. The algorithm "*Knap*(j,$\mathcal{E}$)" can be summarized as follows.

```
set $\Phi = \emptyset$;
**truncate:** Solve LPKnap($s_h$,$\mathcal{E}$,$\Phi$);
If solution $x(s_h,\mathcal{E},\Phi)$ is a binary integer, then return LB;
$I$ := *findNewCut*($s_h$, $\mathcal{E}$, $\Phi$, $x$(LB, $\Phi$) );
 If $I = 0$, then return (*MIPKnap*($s_h$,$\mathcal{E}$,$\Phi$));
$\Phi = \Phi \cup \{I\}$;
 Go to **truncate**;
```

Since the cuts are still valid for the value s_h, they are added to the problem formulation before invoking the MIP solver.

4.2.3.2. Heuristic Algorithm

In this section, we present a heuristic algorithm composed of two stages. In the first stage (Knapsack Stage), jobs are inserted to the availability intervals consecutively by using *Knap*(j,$\mathcal{E}$) algorithm and in the second stage ($C_{\max}$ Stage), $C_{\max}$ with remaining jobs and m dummy jobs is computed. Let E_j be the set of jobs assigned to interval j by *Knap*(j,$\mathcal{E}$), and D be the set of dummy jobs arising from the unavailability periods.
The heuristic algorithm "*Heur*($\mathcal{J},\mathcal{U}$)" is summarized below.

```
Set $j = 1$; $\mathcal{E} = \varnothing$;
**Knapsack:** *Knap*($j$,$\mathcal{E}$); $\mathcal{E} = \mathcal{E} \cup \mathcal{E}_j$;
If $j = m$; go to **ExactAlgorithm**;
$j = j + 1$; go to **Knapsack;**
**ExactAlgorithm 1:** $\mathcal{J} = \mathcal{J} \setminus \mathcal{E} \cup \mathcal{D}$; $C_{\max}(\mathcal{J})$;
**end**
```

It is important to note that the exact algorithm "$C_{\max}(\mathcal{J})$" here runs without the unavailability periods on m parallel identical machines. The problem has a special structure however in that each one of the m machines is assigned exactly one dummy job.

Example 4.2:

Consider the set $\mathcal{J}$ of 3 jobs with processing times {3,7,8} respectively. These jobs should be processed on a two-parallel machine setting in which both of the two machines become unavailable for 5 time units iteratively starting in time 3. There are two knapsacks in this problem with sizes 3, and 8 respectively. Algorithm starts with the smallest sized knapsack and generates the problem

max $3x_1 + 7x_2 + 8x_3$

subject to

$3x_1 + 7x_2 + 8x_3 \leq 3$

$x_{ik} \in \{0,1\}$.

Solving the LP relaxation of this problem, we obtain the solution $\{x_1 = 1, x_2 = 0, x_3 = 0\}$. Because of this solution is integral, the heuristic algorithm excludes job 1 from the job list and passes to the second knapsack. After the following knapsack problem is generated:

max $7x_2 + 8x_3$

subject to

$7x_2 + 8x_3 \leq 8$

x_i's are binary .

Solving the LP relaxation of this problem, we obtain the solution $\{x_2 = 1, x_3 = 0.125\}$. This is not an integral solution, hence the heuristic algorithm adds the cut :

$$7x_2 + 8x_3 \leq 8 - [0\times(1 - x_2) + (1 - x_3)] \quad \Rightarrow \quad 7x_2 + 7x_3 \leq 7$$

Added this cut, the LP relaxation of the algorithm gives the solution $\{x_2 = 0, x_3 = 1\}$. Because of this solution is integral, the heuristic algorithm excludes job 3 from the job list and passes the $C_{\max}$ phase with dummy jobs and remaining job in the job list, hence all knapsack problems are solved. New problem generated by heuristic algorithm is

min y

subject to

$y - 7x_{12} - 8x_{14} - 13x_{15} \geq 0$

$y - 7x_{22} - 8x_{24} - 13x_{25} \geq 0$

$x_{12} + x_{22} = 1$

$x_{14} = 1$

$x_{25} = 1$

Solving the LP relaxation of this problem, we obtain the solution:
$C_{\max} = 14$

$i \backslash j$	M_1	M_2
J_2	0.857	0.143
J_4	1	0
J_5	0	1

The cuts are found that

$7x_{12} + 8x_{14} \leq 14 - [6\times(1 - x_{12}) + 7\times(1 - x_{14})] \quad \Rightarrow \quad x_{12} + x_{14} \leq 1$

$7x_{22} + 13x_{25} \leq 14 - [(1 - x_{22}) + 7\times(1 - x_{25})] \quad \Rightarrow \quad 6x_{22} + 6x_{25} \leq 6.$

Adding these cuts, we found that the LP relaxation of the problem becomes infeasible. Then The $C_{\max}$ value is increased by one ($C_{\max} = 14 + 1$), and cut set is emptied. Then, the LP relaxation is solved with this new $C_{\max}$ value. The following solution is obtained:
$C_{\max} = 15$

$i \backslash j$	M_1	M_2
J_2	1	0
J_4	1	0
J_5	0	1

This is an integral solution and this solution is returned as the solution for the problem.

4.3. COMPUTATIONAL STUDY

As a part of the experimental scheme, experiment carried out by Liao *et al.* [40] is taken into consideration. This experiment contains a 2-machine scheduling problem with an unavailability period only on one machine.

Table 4-1 shows the results for the experimental scheme is given [40]. Results are obtained for small-, medium-, and large-sized problems with n =10, 50, 100. Starting times of the unavailability period are set as 0, $a \times n$, and $(a+b) \times n/4$ where a and b are the lower and upper limits of the uniform distribution from which job processing times generated, respectively. Ending times are selected to cover all three cases discussed in Section 4.2.2.2. Processing times of jobs are generated from $U(20, 50)$ and $U(20, 100)$ for all parameter combinations. Duration of the unavailability period is not constant in this scheme. The proposed algorithm in Section 4.2.2.2 solves all cases in the average 0.185 seconds and in the worst case 1.404 seconds.

In addition to the experimental scheme of [40], the proposed technique is tested also on 2-, 3-, and 5-parallel machine problems with a period of unavailability on each machine. Similar to the problems lends itself to the same reductions and simplifications. The periods of unavailability are consecutive in the sense that a period of unavailability starts on one machine as soon as one ends on a different machine.

Starting and ending times of the periods of unavailability are selected to allow the algorithm to insert jobs to each interval of availability on all machines. Duration of an unavailability period is taken as the average processing time of the jobs. Processing times of jobs are generated from

U(1,70). Table 4-2 shows the results of the exact and heuristic algorithms. The results indicate that for a given number of machines, the CPU time fluctuate randomly as the number of jobs is increased, and do not seem to display any systematic behavior. However, when the number of machines is increased for the same number of jobs per machine, the CPU time increases. Although the algorithm may require unreasonably long CPU times for large instances, it is seemingly able to solve problems with m = 5 machines, and n = 250 jobs within a reasonable time limit. Results shown in Table 4-3 also indicate that the heuristic algorithm provides very good solutions for the instances for which an optimum solution could be obtained. The modest computational times with the heuristic algorithm illustrate its efficiency, and hence suggest that it may provide good solutions for those problems whose size may be prohibitive for the exact procedure.

Experiments with processing time generated from *U*(21,50) are also performed to observe the behavior of the exact and heuristic algorithms when the processing times display little dispersion. Table 4.3 shows the results of the exact and heuristic algorithms for this last scheme in the experimental design. The results indicate that for this class of experiments with a narrow range of processing times, the CPU time for both the exact and the heuristic algorithms increase rapidly as the problem size increases, and after a certain point neither algorithm can provide a solution.

Table 4-1 Exact algorithm CPU times (in seconds) for one unavailability period

t	***P_i* ~ (20,50)**			***t***	***P_i* ~ (20,100)**		
	s				***s***		
***n*= 10**							
	0	**200**	**175**		**0**	**200**	**300**
250	0.074	0.071	0.068	**350**	0.075	0.143	0.069
300	0.13	0.064	0.062	**400**	0.077	0.073	0.07
350	0.091	0.07	0.069	**450**	0.061	0.064	0.068
***n*=50**							
	0	**1000**	**875**		**0**	**1000**	**1500**
1250	0.149	0.125	0.124	**1750**	0.136	0.375	0.12
1500	0.144	0.123	0.122	**2000**	0.141	0.5	0.118
1750	0.164	0.119	0.119	**2250**	0.129	0.094	0.12
***n* = 100**							
	0	**2000**	**1750**		**0**	**2000**	**3000**
2500	0.191	0.186	0.177	**3500**	0.273	1.404	0.182
3000	0.197	0.202	0.175	**4000**	0.248	1.184	0.182
3500	0.163	0.184	0.172	**4450**	0.261	0.138	0.18

Table 4-2: Exact and heuristic algorithm results for cases $p_i \sim U(1,70)$

m	n	s_1	t_1	**Average CPU Time of EA (in seconds)**	**Average CPU Time of Heuristic (in seconds)**	**Percent Deviation from Optimum**	**Average CPU Time of MIP**
$p_i \sim U(1, 70)$							
2	4	35	70	0.164	0.178	0	0.036
2	6	50	85	0.138	0.162	0.73	0.036
2	8	70	105	0.094	0.158	3.547	0.058
2	10	85	120	0.164	0.194	4.835	0.116
2	14	100	135	0.384	0.21	2.252	0.44
2	20	160	195	0.248	0.356	0.287	0.218
2	40	335	370	0.418	0.794	0	0.364
2	100	860	895	1.384	1.684	0	1.402
3	9	35	70	0.22	0.108	0	0.058
3	12	50	85	0.422	0.208	3.828	0.294
3	15	70	105	8.286	0.17	3.366	7.91
3	21	90	125	34.098	0.18	0.727	7.318
3	30	165	200	4.854	0.21	0.174	4.632
3	60	315	350	1.306	0.41	0	2.32
3	150	805	840	7.88	0.694	0	6.374
5	25	0	35	29.612	0.858	1.748	116.65
5	35	55	90	165.258	0.844	1.469	158.338
5	50	90	125	47.466	5.652	0	138.992

5	100	265	300	40.046	1.87	0	50.292
5	250	785	820	67.18	3.764	0	44.438

Table 4-3 Exact and heuristic algorithm results for cases $p_i \sim U(21,50)$

m	n	s_1	t_1	**Average CPU Time of EA (in seconds)**	**Average CPU Time of Heuristic (in seconds)**	**Percent Deviation From Optimum**
$p_i \sim U(21, 50)$						
2	10	85	120	0.226	0.116	0.297
2	40	335	370	19.466	0.166	0
2	100	860	895	2.38	0.298	0
3	15	70	105	7.406	0.186	3.378
3	60	315	350	70.148	0.792	0

Chapter 5
CONCLUSION

This thesis addressed the problem of parallel machine scheduling with availability constraints on each machine. The objectives of minimizing the total completion time ($\sum C_j$), and minimizing the makespan ($C_{\max}$) are studied.

We developed an exact branch-and-bound procedure, and proposed three heuristic algorithms to find approximate solutions for the $\sum C_j$ problem. A pre-emptive lower bound is used within the branch-and-bound procedure. The three heuristic algorithms are a constructive algorithm, a neighborhood improvement algorithm and a simulated annealing procedure. The constructive algorithm relies on assignments of jobs based on a projection for the best promising solution. The algorithm has a worst-case error bound of 2. The neighborhood improvement algorithm takes the schedule produced by the constructive procedure and attempts to improve it through a local search within a given neighborhood. The particular neighborhood structure can be thought of as a combination of smaller neighborhoods. The search process exploits some optimality properties of the problem. Computational results show that the improvement algorithm gives very good solutions (with less than 1% deviation from lower bound in all cases), very efficiently with less than 2 seconds of CPU times in all cases.

The simulated annealing procedure is proposed as a random search technique using the same neighborhood structure in the hopes to avoid entrapment at local optima. Although it gives excellent solutions (optimal solutions for almost all 30-job problems and less than 1% deviation from the lower bound for the 50- and 70-job problems), its CPU times overly exceed those with the improvement algorithm.

The second part of the thesis addressed the problem of minimizing $C_{\max}$. We developed an exact algorithm based on a branch-and-cut method, and a heuristic algorithm that uses a divide-and-conquer strategy. We modified a cutting plane scheme previously proposed in the literature for a parallel machine problem under continuous availability, and proposed a transformation of our problem to make it suitable for the application of the modified cutting plane scheme. This same cutting plane scheme is also utilized within the context of a new heuristic algorithm that divides the problem into sub-problems, solves these sub-problems to optimality, and reports the combined solution. Computational experimentation shows that the exact algorithm solves problems within a 10-minute CPU time only when the range of processing times is large. The heuristic algorithm provides very good solutions within reasonable CPU times even when the processing times are sampled from a smaller range.

An immediate extension of this study is the consideration of the total flextime objective with simultaneous periods of unavailability on different machines. Problems with total weighted completion time objective and the due date related objectives (e.g., minimizing maximum lateness ($L_{\max}$), number of tardy jobs) are other natural extensions. In addition, stochastic periods of unavailability, for example due to unexpected breakdowns, may be another interesting and challenging avenue for future research.

References:

[1] R. Aggoune. Minimizing the Makespan for the Flowshop Scheduling Problem with Availability Constraints. *European Journal of operational Research*, 153:534-543, 2004.

[2] M.S. Akturk, J.B. Ghosh, E.D. Gunes. Scheduling with Tool Changes to Minimize Total Completion Time: A Study of Heuristics and Their Performance. *Naval Research Logistics*, 50:15-30, 2003.

[3] M.S. Akturk, J.B. Ghosh, E.D. Gunes. Scheduling with Tool Changes to Minimize Total Completion Time: Basic Results and SPT Performance. *European Journal of Operational Research*, 157:784-790, 2004.

[4] M. Azizoglu, and O. Kirca. On the Minimization of Total Weighted Flowtime with Identical and Uniform Parallel Machines. *European Journal of Operational Research*, 113:91-100, 1999.

[5] J. Blazewicz, and M. Drozdowski, P. Formanowicz, W. Kubiak, G. Schmidt, Scheduling Preemptable Tasks on Parallel Processors with Limited Availability. *Parallel Computing*, 26:1195-1211, 2000.

[6] J. Blazewicz, J. Breit, P. Formanowicz, W. Kubiak, and G. Schmidt. Heuristic Algorithms for the Two-Machine Flowshop with Limited Machine Availability. *Omega*, 29:599-608, 2001.

[7] J. Breit. An Improved Approximation Algorithm for Two-Machine Flowshop Scheduling with an Availability Constraint. *Information Processing Letters*, 90(6): 273-278, 2004.

[8] L.J. Bruno, E.G. Coffman, and R. Sethi. Scheduling Independent Tasks to Reduce Mean Finishing Time. *AIIE Transactions*, 17:382-387, 1974.

[9] P.C. Chang, and Y.S. Jiang. A State-Space Search Approach for Parallel Processor Scheduling Problems with Arbitrary Precedence Relations. *European Journal of Operational Research*, 77(2):208-223, 1994.

[10] T.C.E. Cheng. A State-of-the Art: A Review of the Parallel Machine Scheduling. *European Journal of Operational Research*, 18(2): 193-242, 2001

[11] T.C.E. Cheng, and G. Wang. Two-Machine Flowshop Scheduling with Consecutive Availability Constraints. *Information Processing Letters*, 71:49-54, 1999.

[12] T.C.E. Cheng, and G. Wang. An Improved Heuristic for Two-Machine Flowshop Scheduling with Availability Constraints. *Operations Research Letters*, 26:223-229, 2000.

[13] R.W. Conway, W.L. Maxwell, and L.W. Miller. *Theory of Scheduling*, Addison-Wesley, 1967.

[14] E. Davis, and J.M. Jaffe. Algorithms for Scheduling Tasks on Unrelated Processors. *Journal of Assoc. Computing Mach.*,28:721-736, 1981.

[15] P. De, and T.E. Morton. Scheduling to Minimize Makespan on Unequal Parallel Processors. *Decision Science*, 11:586-603, 1980.

[16] M. Dell'Amico, and S. Martello. Optimal Scheduling of Tasks on Identical Parallel Processors. *ORSA Journal of Computing*, 7:191-200, 1995.

[17] M. Dell'Amico, and S. Martello. A Note on Exact Algorithms for the Identical Parallel Machine Scheduling Problem. *European Journal of Operational Research*, 160: 576-578, 2005.

[18] M.R. Garey, and D.S. Johnson. Computers and Intractability: A Guide to the Theory of NP-completeness, Freeman, San Francisco, 1979.

[19] M.R. Garey, and D.S. Johnson. Strong NP-completeness Results: Motivation, Examples and Implications. *Journal of the ACM*, 25: 499-508, 1978.

[20] A. Gharbi, and M. Haouari. Optimal Parallel Machines Scheduling with Availability Constraints. *Discrete Applied Mathematics*, 148:63-87, 2005.

[21] M. Ghirardi, and C.N. Potts. Makespan Minimization for Scheduling Unrelated Parallel Machines: A Recovering Beam Search

Approach. *European Journal of Operational Research*, 165(2): 457-467, 2005

[22] G.H. Graves, and C.Y. Lee. Scheduling Maintenance and Semiresumable Jobs on a Single Machine. *Naval Research Logistics*, 46:845-863, 1999.

[23] A.M.A. Hariri, and C.N. Potts. Heuristics for Scheduling Unrelated Parallel Machines. *Comput. Operations Research*, 18: 313-321, 1991.

[24] J.C. Ho, and J.S. Wong. Makespan Minimization for *m*-parallel Identical Processors. *Naval Research Logistics*, 42:935-942, 1995.

[25] E. Horowitz, and S. Sahni. Exact and Approximate Algorithms for Scheduling Non-identical Processors. *Journal of Assoc. Computing Mach.*, 23: 317-327, 1976.

[26] O.H. Ibarra, and C.E. Kim. Heuristic Algorithms for Scheduling Independent Tasks on Non-identical Processors. *Journal of Assoc. Computing Mach.*,24: 280-289, 1978.

[27] M. Kaspi, and B. Montreuil. On the Scheduling of Identical Parallel Processes with Arbitrary Initial Processor Available Time. *Research Report, 88-12, School of Industrial Engineering, Purdue University, West Lafayette, IN*, 1988.

[28] S. Kirkpatrick, C.D. Gelatt Jr., M.P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671-680, 1983.

[29] C.Y. Lee. Parallel Machines Scheduling with Non-simultaneous Machine Available Time. *Discrete Applied Mathematics*, 30: 53-61, 1991.

[30] C.Y. Lee. Two-Machine Flowshop Scheduling with Availability Constraints. *European Journal of Operational Research*, 114:420-429, 1999.

[31] C.Y. Lee. Minimizing the Makespan in the Two-Machine Flowshop Scheduling Problem with an Availability Constraint. *OR Letters*, 20:129-139, 1997.

[32] C.Y. Lee, and Z.L. Chen. Scheduling Jobs and Maintenance Activities on Parallel Machines. *Naval Research Logistics*, 47:145-165, 2000.

[33] C.Y. Lee, and V.J. Leon. Macihne Scheduling with Rate-Modifying Activity. *European Journal of Operational Research*, 128:119-128, 2001.

[34] C.Y. Lee, and S.D. Liman. Single Macihne Flow-time Scheduling with Scheduled Maintenance. *Acta Informatica*, 29: 375-382, 1992.

[35] C.Y. Lee, and S.D. Liman. Capacitated Two-Parallel Machine Scheduling to Minimize Sum of Job Completions. *Discrete Applied Mathematics*, 41:211-222, 1993.

[36] C.Y. Lee, and C.S. Lin. Single Machine Scheduling with Maintenance and Repair Rate Modifying Activities. *European Journal of Operational Research*, 135:493-513, 2001.

[37] J.K. Lenstra, and D.B. Shmoys, E. Tardos. Approximation Algorithms for Scheduling Unrelated Parallel Machines. *Math. Programming*, 46: 259-271, 1990.

[38] J.Y.T. Leung, and M. Pinedo. A Note on Scheduling Parallel Machines Subject to Breakdown and Repair. *Naval Research Logistics,* 2004.

[39] C.J. Liao, and W.J. Chen. Single Machine scheduling with Periodic Maintenance and Non-resumable Jobs. *Computers & Operations Research*, 30: 1335-1347, 2003.

[40] C.L. Liao, D.L. Shyur, and C.H. Lin. Makespan Minimization for Two Parallel Machines with an Availability Constraint. *European Journal of Operational Research*, 160:445-456, 2005.

[41] S.D. Liman. Scheduling Capacities and Due Dates. *PhD. Thesis*, *University of Florida*, *Gainesville*, *FL*, 1991.

[42] S. Martello, F. Soumis, and P. Toth. Exact and Approximation Algorithms for Makespan Minimization on Unrelated Parallel Machines. *Discrete Applied Mathematics*, 75:169-188, 1997.

[43] R. McNaughton. Scheduling with Deadlines and Loss Functions. *Management Science*, 6:1-12, 1959.

[44] E. Mokotoff. Parallel Macihne Scheduling Problems: A Survey. *Asia-Pacific Journal of Operational Research*, 18(2):193-242, 2001.

[45] E. Mokotoff, and P. Chretienne. A Cutting Plane Algorithm for the Unrelated Parallel Machine Scheduling Problem. *European Journal of Operational Research*, 141:515-525, 2002.

[46] E. Mokotoff. An Exact Algorithm for the Identical Parallel Machine Scheduling Problem. *European Journal of Operational Research*, 152:758-769, 2004.

[47] R. Muntz, and E.G. Coffman, Preemptive Scheduling of Real-time Tasks on Multiprocessor Systems. *Journal of ACM*, 17:324-338, 1970.

[48] C.N. Potts. Analysis of a Linear Programming Heuristic for Scheduling Unrelated Parallel Machines. *Discrete Applied Mathematics*, 10:155-164, 1985.

[49] C. Sadfi, B. Penz, C. Rapine, J. Blazewicz, .and P. Formanowicz. An Improved Approximation Algorithm for the Single Machine Total Completion Time Scheduling Problem with Availability Constraints. *European Journal of Operational Research*, 161:3-10, 2005.

[50] A. Salem, G.C. Anagnostopoulos, and G. Rabadi. A Branch-and-Bound Algorithm for Parallel Machine Scheduling Problems. *Proceedings of the International Workshop on Harbour, Maritime &Multimodel Logistics Modeling &Simulation* (*HMS 2000*) *Society for Computers & Simulation International*, 88-93, 2000

[51] E. Sanlaville, and G. Schmidt. Machine Scheduling with Availability Constraints. *Acta Informatica*, 35:795-811, 1998.

[52] G. Schmidt. Scheduling with Limited Machine Availability. *European Journal of Operational Research*, 121:1-15, 2000.

[53] S.L. van de Velde. Duality-based Algorithms for Scheduling Unrelated Parallel Machines. *ORSA Journal of Computing*, 5: 192-205, 1993.