OPERATIONAL FIXED JOB SCHEDULING PROBLEM

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

BY

DENİZ TÜRSEL ELİİYİ

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

SEPTEMBER 2004

Approval of the Graduate School of Natural and Applied Sciences

_____

Prof. Dr. Canan Özgen
Director

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

_____

Prof. Dr. Çağlar Güven
Head of Department

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

_____

Prof. Dr. Meral Azizoğlu
Supervisor

Examining Committee Members

Prof. Dr. Ömer Kırca (METU,IE) _____

Prof. Dr. Meral Azizoğlu (METU,IE) _____

Assoc. Prof. Dr. Canan Sepil (METU,IE) _____

Assoc. Prof. Dr. Selim Aktürk (BİLKENT,IE) _____

Asst. Prof. Dr. Arslan Örnek (Dokuz Eylül Ünv.,IE) _____

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

Name, Last Name: Deniz Türsel Eliiyi

Signature:

ABSTRACT

OPERATIONAL FIXED JOB SCHEDULING PROBLEM

Türsel Eliiyi, Deniz
Ph.D., Department of Industrial Engineering
Supervisor : Prof. Dr. Meral Azizoğlu

September 2004, 149 pages

In this study, we consider the Operational Fixed Job Scheduling Problem on identical parallel machines. The problem is to select a subset of jobs for processing among a set of available jobs with fixed arrival times and deadlines, so as to maximize the total weight. We analyze the problem under three environments: Working time constraints, Spread time constraints, and Machine dependent job weights. We show that machine eligibility constraints appear as a special case of the last environment.

We settle the complexity status of all problems, and show that they are NP-hard in the strong sense and have several polynomially solvable special structures.

For all problems, we propose branch and bound algorithms that employ powerful reduction mechanisms and efficient lower and upper bounds. The results of our computational runs reveal that, the algorithms return optimal solutions for problem instances with up to 100 jobs in reasonable solution times.

Keywords: Fixed Job Scheduling, Working Time Constraints, Spread Time Constraints, Machine Dependent Job Weights, Eligibility Constraints.

ÖZ

OPERASYONEL SABİT İŞ ÇİZELGELEMESİ PROBLEMİ

Türsel Eliiyi, Deniz
Doktora, Endüstri Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Meral Azizoğlu

Eylül 2004, 149 sayfa

Bu çalışmada özdeş paralel makinelerde Operasyonel Sabit İş Çizelgelemesi Problemi ele alınmıştır. Problem, geliş ve teslim zamanları sabit işler kümesi içinden, proses edilecek iş kümesinin toplam ağırlığı maksimize edilecek şekilde seçilmesidir. Problem, çalışma zamanı kısıtları, yaygınlık zamanı kısıtları ve makine bağımlı iş ağırlıkları olmak üzere üç ayrı ortamda ele alınmıştır. İşleyebilirlik kısıtlı problemin, belirtilen son ortamın özel bir durumu olduğu gösterilmiştir.

Çalışmada tüm problemlerin karmaşıklık statüleri saptanmış, hepsinin NP-zor olduğu gösterilmiş, ve her bir problem için polinom zamanda çözülebilir özel durumlar belirlenmiştir.

Tüm problemler için, problem boyutunu azaltma mekanizmalarından ve verimli alt ve üst sınır algoritmalarından yararlanan Dal ve Sınır algoritmaları geliştirilmiştir. Deneysel sonuçlarımız algoritmaların 100 işe kadar olan problemleri makul zamanlarda optimal olarak çözebildiğini göstermiştir.

Anahtar Kelimeler: Sabit İş Çizelgelemesi, Çalışma Zamanı Kısıtları, Yaygınlık Zamanı Kısıtları, Makine Bağımlı İş Ağırlıkları, İşleyebilirlik Kısıtları.

ACKNOWLEDGEMENTS

I deem myself extremely blessed to have such a great thesis supervisor as Prof. Dr. Meral Azizoğlu. I am so lucky to have her as a mentor and a friend. She was my constant inspiration in this study with all her devoted guidance, advice, insight, comments and encouragement. I also had the chance to experience her understanding and precious support beyond the academic issues. I hope we will be able to work together in the future.

I would like to express my thanks to Prof. Dr. Ömer Kırca and Assoc. Prof. Dr. Selim Aktürk for their valuable guidance and comments throughout the study. Thanks to them, the follow-up process was very constructive and beneficial.

I owe my mom and dad, Hamide and Ali Türsel, for their everlasting love, empathy, and their faith in me. My love, Uğur Eliiyi was my endless support as he always is. I am grateful for his treasured backing all through my studies.

I have felt the warm support of all my friends in the METU IE department. I would like to thank all of them, especially Burcu Balçık and Selin Bilgin, for their existence and support. I sincerely wish that we will be together again in the same city in the near future.

Last but not the least, thanks to all my professors, my colleagues and the staff in the department for their ever smiling faces.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

Interval Scheduling (IS) problem is an emerging area of scheduling where tasks, each with specified ready times and deadlines, are to be processed on a number of resources. The IS problem is typical for reservation systems and has many real life applications, such as classroom assignment, transportation systems, and shift scheduling.

Reservation systems may arise in a service environment like hotel room reservations and car rental/repair services, where customers represent tasks and hotel rooms, cars or technicians correspond to resources. They may also take place in production environments, where tasks become parts or jobs, and resources become machines. In either environment, tasks give reserve requests to resources during specified time windows of processing, and the problem may involve the decisions as to which orders to accept, or how many resources to accommodate to serve all orders.

A reservation system typically refers to a parallel resource environment. Using conventional scheduling terminology (this terminology will be employed throughout the text), the problem environment can be represented as follows: There are n independent jobs available to be processed in a non-preemptive fashion on m parallel machines. The time window of job j is specified by a release (ready) time r_j and a due time (deadline) d_j. If the time window of a job is larger than its processing time; that is, if it may be started after its ready time, the problem is referred to as the Variable Job Scheduling Problem. On the other hand, if the job cannot be delayed after its ready time, then the IS problem becomes a Fixed Job Scheduling (FJS)

problem. The processing time p_j of job j is equal to (d_j - r_j) in the FJS problem, where it is smaller than or equal to (d_j - r_j) in Variable Job Scheduling problem. If a job is accepted for processing, it will start at r_j and finish at d_j in FJS. However, in variable job scheduling, the start time of the accepted job is a decision variable that takes a value between r_j and (d_j - p_j).

The FJS problem has two variants based on the objective functions. The first variant is the Operational Fixed Job Scheduling (OFJS) problem. In the OFJS problem, each job has a weight w_j that represents its value or relative importance (e.g. profit made by processing the job) for the decision maker, and maximizing weighted number of processed jobs with a given number of processors is of concern. When all jobs have equal weights, i.e. when w_j=w for all j, the objective reduces to maximizing the number of jobs processed. This problem is also known as the Maximal (Weight) Interval Scheduling Problem. The second variant is the Tactical Fixed Job Scheduling (TFJS) problem, where there is a fixed cost c_k associated with machine k, and the objective is the minimization of the total cost of the machines needed to process all available jobs. When machine costs are identical, i.e. when c_k=c for all k, the associated objective reduces to minimizing the number of necessary machines. Note that, the number of machines is a parameter for the OFJS problem, whereas it is a decision variable in the TFJS problem.

There are many practical applications of the TFJS and OFJS problems cited in the literature. In a study by Kroon (1990), the TFJS problem is used as the core model in tactical capacity planning of the aircraft maintenance personnel for an airline company. An aircraft arriving at an airport requires a number of short maintenance inspections, which are specified by the strict maintenance norms. Hence, the timetables of the airline companies and the maintenance norms determine the fixed intervals in which the inspections have to be carried out in order to avoid aircraft delays. The problem may further be complicated by a safety rule that gives license to each engineer to carry out inspections on at most two different aircrafts. A later study by Kroon *et al.* (1995) addresses the OFJS variant of the same case, i.e. for a

given number of maintenance engineers, where the priorities are defined for maintenance inspections.

Another application of the TFJS problem is the Bus Driver Scheduling Problem (see Fischetti *et al.*, 1987). Given a transit company's bus schedule for a particular day, the Bus Driver Scheduling Problem finds a set of driver duties that covers the schedule at minimum cost, while satisfying a set of constraints laid down by the union contract and company regulations.

The OFJS problem may as well be observed in scheduling earth-observing satellites (EOS), as reported by Wolfe and Sorensen (2000). Hundreds of orbiting satellites are used to observe the earth and transmit data to the ground for further processing. A satellite occasionally passes through positions where specific data can be collected, and it needs a link to a relay satellite for transmission to the ground. However, there are very few relay satellites and they do not have enough capacity to guarantee a link. The priority for any observation is set beforehand, and this helps resolve satellite conflicts.

In the most basic setting of the FJS problem, each job needs to be processed on at most one machine. Individual machines are able to perform all operations, i.e. all machines are eligible for processing all jobs. In addition, the processing time requirement of a job does not depend on the machine that it is assigned to, that is, all machines are identical (in speed). Preemption and job splitting are not allowed. There are no restrictions on machine capacities, and all machines are available at all times.

In practice, the cases are often more sophisticated so that additional constraints and parameters that are more general are needed. The variations of the FJS problem that we consider in this study are the following.

General weights: In a generalization of the OFJS problem, the weight of a job j may also depend on the machine to which it is assigned, i.e. the profit depends on

the machine as well as the job. In this case, the weight of job j if processed on machine k becomes w_{jk}. The FJS problem with eligibility constraints appear as a special case of machine dependent job weights, where each machine is eligible to process only a subset of jobs. The special cases of eligibility constraints may arise such as several job and machine classes based on their similarities, and for each job and machine class combination, the feasibility of assigning a job to a machine may be established in advance. Aircraft and Classroom Scheduling are two problems where eligibility constraints find their applications.

Operating time constraints: As a generalization of the FJS problem, there may be some constraints on the operating times of the machines.

- **Working time constraints:** One such generalization includes working time constraints where the actual time that the machine operates is bounded; that is, machine j is not allowed to operate for more than a given total working time of T_j units. Consequently, working time of a machine is the sum of the processing times of the jobs assigned to that machine, and it does not include idle times. In the Bus Driver Scheduling Problem example, a working time limit is imposed on the duty of any driver. For example, when T_j=8 hours for all drivers, a job with deadline 10 p.m. may well be assigned to the driver if there is much idle time in between. Moreover, in production environments, it may not be economical to operate some high precision machines more than their preset allowable time capacity. Working time bound may be the same for all machines, i.e. T_j=T, or may be arbitrary.
- **Spread time constraints:** Another generalization includes spread time constraints, where an upper bound S_j is imposed on the total time between the start and the finish times of the operations on any machine j. In particular, the spread time of a machine is defined as the time between the earliest ready time and the latest deadline among all jobs that are processed on that machine. Note that there may be machine idle times between these two time points, yet these times are included in the spread time as opposed to working time. For example, if S_j=8 hours for all drivers in the Bus Driver Scheduling Problem example, a driver who sets out his duty with a job starting at 10 a.m. should stop working

at 6 p.m., and cannot be assigned any job with deadline later than 6 p.m.. Spread time bound may be the same for all machines, i.e. $S_j=S$, or arbitrary for each machine.

In the literature, most of these variations are studied only for the TFJS problem. Despite its practical importance, the research on the OFJS problem is quite limited, as it will be revealed in the following chapters. Our study is intended to fill the gap in the OFJS literature.

The rest of this thesis is organized as follows. In the next chapter, we present the mathematical formulation for the basic models of the FJS problem and state the existing results. We also present our results that relate the TFJS and OFJS problems, and the OFJS problem with working time and spread time constraints. In Chapter 3, the FJS problem with working time constraints is examined in detail, the related literature is reviewed, and our findings and future work are presented. In Chapter 4, the FJS problem with spread time constraints is covered with the related literature and our findings. In Chapter 5, we cover the FJS problem with general weights, of which special case becomes the problem with eligibility constraints. Computational experimentation and results for all problems are presented in Chapter 6. In Chapter 7, we present our conclusions, and introduce some topics for the OFJS problem that are worth studying further.

CHAPTER 2

PROBLEM DEFINITION

In this chapter, we present the mathematical formulations and complexity results for the two variants of the FJS problem, namely the OFJS problem and the TFJS problem. We state the assumptions and constraints common to both variants, analyze each problem separately, and present the complexity results.

2.1. COMMON ISSUES

In the basic model of the FJS problem, the following conventional scheduling assumptions are made. For the sake of completeness, we restate the assumptions.

- All machines are identical in speed.
- A machine can process at most one job at a time.
- All machines can process all jobs, i.e., there is no concern of eligibility.
- All jobs and machines are available at time zero.
- There are no machine breakdowns, and no intervals of machine unavailability.
- A job should be processed without any interruption once it is started, i.e. no preemption is allowed.
- A job can be processed by at most one machine at a time, i.e. no job splitting is allowed.
- All parameters are known with certainty.

The main parameters of the problem are:
m: number of machines, and
n: number of jobs.

The indices are:

i, j :	Job indices,	$i, j = 1,...,n$
k:	Machine index,	$k = 1,...,m$.

The job-related parameters are:

- r_j: ready time of job j, $j=1,\ldots,n$
- d_j: deadline of job j, $j=1,\ldots,n$
- p_j: processing time of job j, $p_j = d_j - r_j$, $j=1,\ldots,n$
- w_j: weight of job j, $j=1,\ldots,n$.

Without loss of generality, we assume that all numerical data are positive integers. We can assume that the time is divided into periods that are not necessarily equal in length. This is achieved by forming a chronological sequence of all ready times and deadlines. Namely, let $\{t_1, t_2,...\}$ be the sorted sequence of the r_js and d_js in chronological order with duplicates removed. Let P_a be the set of jobs that need to be processed in the interval $[t_a, t_{a+1})$ for $a=1, 2,...$

Our binary decision variable is defined as follows:

$$x_{jk} = \begin{cases} 1, & \text{if job } j \text{ is processed on machine } k, \\ 0, & \text{otherwise.} \end{cases} \qquad \forall j,k.$$

The fundamental constraints that are common for both variants of the FJS problem are the following:

- Each job j is processed on at most one machine:

$$\sum_{k=1}^{m} x_{jk} \le 1 \qquad j = 1,...,n \tag{1}$$

- No machine can process more than one job at a time:

$$\sum_{j \in P_a} x_{jk} \le 1 \qquad k = 1,...,m \qquad \forall a \tag{2}$$

- Preemption and job splitting are not allowed:

$$x_{jk} \in \{0,1\} \qquad k = 1,...,m \qquad j = 1,...,n \tag{3}$$

2.2. THE OFJS MODEL

The constraints stated in the previous section define the feasible region for the OFJS problem. The complete model for the OFJS problem then becomes:

$$\text{Maximize} \sum_{k=1}^{m} \sum_{j=1}^{n} w_j x_{jk} \tag{4}$$

subject to (1), (2), and (3).

The objective function expressed in (4) maximizes the total weight of the processed jobs, in other words it maximizes total profit. When the weights are identical for all jobs and machines, i.e. when $w_j = w$, $\forall j$, the problem reduces to the maximization of the number of processed jobs.

Bouzina and Emmons (1996) provide an $O(n \log n)$ algorithm for solving the OFJS problem with the objective of the maximization of the number of processed jobs. The algorithm is stated below.

Algorithm 2.2.1:

S1. Set $S = \varnothing$.

S2. Index the jobs in chronological order of ready times.

S3. Consider the jobs sequentially. At each ready time, add the arriving job to set S. If no machine is available at the ready time of a job, remove the job with the latest deadline from set S.

Bouzina and Emmons (1996) also formulate the total weight maximization problem as a Minimum Cost Network Flow (MCNF) problem with n+1 nodes and 2n arcs. Hence, this problem is polynomially solvable, since there exist polynomial time algorithms for MCNF (Orlin, 1993). Below is the description of the MCNF problem specified for solving this OFJS problem.

Algorithm 2.2.2:

S1. Index jobs in chronological order of ready times.

S2. Create nodes $s = V_1, V_2, \ldots, V_n$ for each job, and a dummy node $t = V_{n+1}$. Connect V_j to V_{j+1} with arc cost zero and capacity m, j=1,…,n.

S3. Create arc (V_j, V_k), where V_k is the first job not overlapping with job j, j=1,…,n. If no such job exists, create arc (V_j, t). Each of these arcs has cost $-w_j$ and capacity 1.

S4. Require a flow of m from s to t and solve the resulting MCNF problem.

We illustrate the network structure through the following example problem.

Example 1: Consider the jobs in Figure 2.1 to be scheduled on 2 parallel identical machines. Labels are the job numbers. Assume $w_j = p_j$, for j=1,…,8, i.e. the profit is charged per unit time.

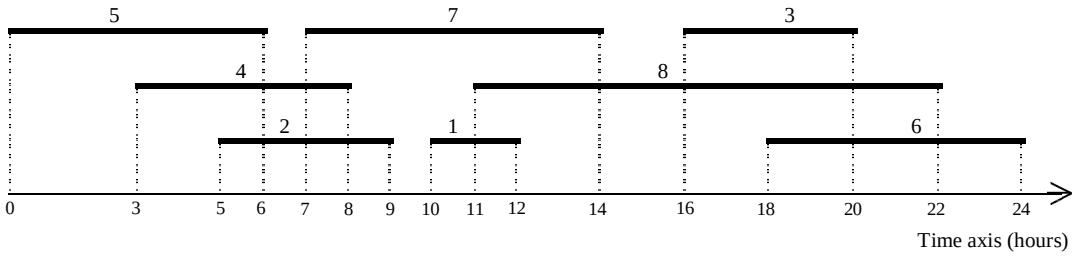


Figure 2.1. Instance of the OFJS problem

Using Algorithm 2.2.1, the maximum number of jobs is found as 6, with the processed job set {1,3,4,5,6,7}.

The network structure for Algorithm 2.2.2 is given in Figure 2.2, where each arc is labeled with its cost $-w_j$, and capacity 1, except for the arcs generated in Step 2 (those on the straight line from s to t), whose costs and capacities are 0 and 2, respectively.

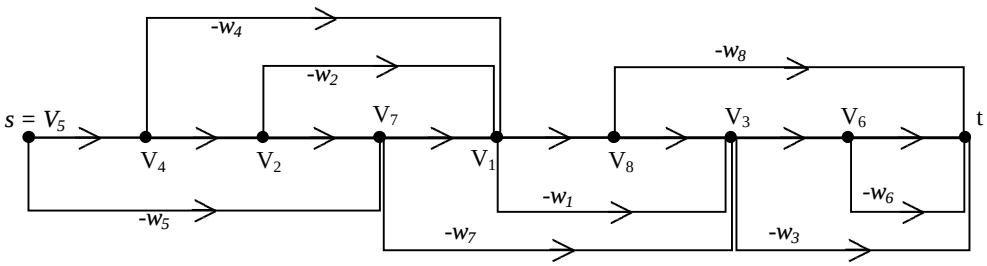


Figure 2.2. Network representation of the instance

The minimal cost becomes –35 and utilizes the arcs corresponding to the job set {4,5,6,7,8}.

2.3. THE TFJS MODEL

The formulation of the TFJS problem additionally necessitates a decision variable for the machines. For this purpose, we let;

$$y_k = \begin{cases} 0, & \text{if no job is assigned to machine } k, \\ 1, & \text{otherwise.} \end{cases} \qquad \forall k.$$

Note that, in the case of the TFJS problem, the parameter m is needed as the upper bound on the number of machines. One such valid upper bound is the number of jobs, n. The following additional constraints are necessary to represent the TFJS problem:

$$x_{jk} \le y_k \qquad k = 1,...,m \qquad j = 1,...,n \tag{5}$$

$$y_k \in \{0,1\} \qquad k = 1,...,m \tag{6}$$

Constraint set (5) prevents the assignment of the jobs to unutilized machines, and constraint set (6) guarantees the integrality of the machine usage variables. Then, the complete model for the TFJS problem becomes:

$$\text{Minimize } \sum_{k=1}^{m} c_k y_k \tag{7}$$

subject to (1'), (2), (3), (5), and (6),

where constraint (1') is the equality form of constraint (1), and c_k is the cost of operating machine k. The objective function expressed in (7) minimizes the total cost of used machines. When $c_k = c$, the problem becomes the minimization of the total number of machines.

The TFJS problem with $c_k = c$ was studied by Dantzig and Fulkerson (1954), and Gertsbakh and Stern (1978) in the context of fleet planning, and by Hashimoto and Stevens (1971), and Gupta *et al.* (1979) in the context of computer wiring. An optimal solution to the problem is described by the following lemma:

Lemma 2.3.1: The minimum number of machines required to carry out all jobs of an instance of the TFJS problem equals the maximum job overlap of the jobs.

This lemma is a direct consequence of Dilworth's theorem on partially ordered sets, stating that in any partially ordered set, the minimum number of chains required for covering all elements is equal to the size of a maximum anti-chain (Dilworth, 1950). An $O(n \log n)$ algorithm for determining the maximum job overlap of the jobs is described by Hashimoto and Stevens (1971) and Gupta *et al.* (1979). Note that according to our notation, the maximum job overlap is $\text{Max}_a\{|P_a|\}$, where $|P_a|$ is the cardinality of set P_a. Consider Figure 2.1 of Example 1. As the maximum job overlap equals 3, the minimum number of machines required to carry out all jobs equals 3 as well.

To the best of our knowledge, there are no reported studies that state the complexity of the case when $c_k \neq c$. We propose Algorithm 2.3.1 below to solve this problem optimally, and settle its complexity in Corollary 2.3.1.

Algorithm 2.3.1:

S1. Find m^* = minimum number of machines required to process all jobs by Hashimoto and Stevens's algorithm.

S2. Select m^* machines that have the lowest costs among m machines. The optimal solution is the total cost of the selected machines.

Theorem 2.3.1: Algorithm 2.3.1 solves the TFJS problem when $c_k \neq c$ optimally.

Proof: The solution can be improved in two ways. The first way is removing a machine m from the solution and adding another machine l which is not in the solution, such that $c_l < c_m$. Since Algorithm 2.3.1 places the machines with minimum costs in the solution, there cannot exist such machine pairs.

The second way is to remove a machine from the solution. As the algorithm finds the minimum number of machines necessary to complete all jobs, it is not feasible to remove a machine from the solution.

Then, a solution by Algorithm 2.3.1 cannot be improved, hence it is optimal. c

Corollary 2.3.1: Algorithm 2.3.1 runs in $O(n \log n)$ time.

Proof: The complexity of Hashimoto and Stevens's algorithm is $O(n \log n)$, which defines the complexity of S1. In S2, we select m^* promising machines out of m machines in $O(m \log m)$ time. Since $n > m$, the overall complexity is defined by S1 as $O(n \log n)$. c

2.4. RELATION BETWEEN COMPLEXITIES OF THE PROBLEMS

In this section, we settle the complexity of any OFJS problem, once the complexity of its tactical version is known.

Consider problem P, a TFJS problem that minimizes the total number of machines to process all jobs, and with the constraint set X, which consists of the basic constraint set of TFJS model and some additional constraints (like working time, spread time, eligibility, etc.). We can represent P as follows:

Minimize $\sum_{k=1}^{m} y_k$

subject to X,

$$\sum_{k=1}^{m} x_{jk} = 1 \qquad j = 1,...,n \qquad (1')$$

$$x_{jk} \le y_k \qquad k = 1,...,m \qquad j = 1,...,n \qquad (5)$$

$$y_k \in \{0,1\} \qquad k = 1,...,m \qquad (6)$$

where X= {(2), (3)}∪{additional constraint set x}.

Consider an OFJS problem P', whose constraint set includes the basic constraint set of the OFJS model and the same additional constraints as that of P. The objective function is maximization of the total weighted number of processed jobs. P' is represented as follows:

Maximize $\sum_{k=1}^{m} \sum_{j=1}^{n} w_j x_{jk}$

subject to X,

$$\sum_{k=1}^{m} x_{jk} \le 1 \qquad j = 1,...,n \qquad (1)$$

where X= {(2), (3) }∪{additional constraint set x}

Through Theorem 2.4.1, we establish a relation between the complexities of P and P'.

Theorem 2.4.1: If P is NP-hard, then P' is NP-hard.

Proof: Suppose that P is NP-hard, but there exists a polynomial algorithm A' that solves P' optimally. Assume p(n) is the complexity function of algorithm A'. Then, algorithm A below gives an optimal solution for P.

Algorithm A:

S0. Set m = 1.

S1. Solve P' by Algorithm A' with m machines. Let the resulting objective function value be $Z^*_{P'}$.

S2. If $Z^*_{P'} < \sum_{j=1}^{n} w_j$ (if every job is not processed), then

Set $m = m+1$,

Go to S1

Else, Stop. $Z^*_P = m$.

Clearly Z^*_P is the optimal solution for P, since we increment m one at a time, solve the operational problem with m, and check whether all jobs are processed. Note that, S1 is executed, i.e. Algorithm A' is run, at most n times since $m \leq n$. Since A' runs in $O(p(n))$ time, A runs in $O(n\ p(n))$ time, i.e. in polynomial time. Hence, there is a contradiction, P' is NP-hard as well. c

2.5. THE PROBLEMS OF OUR CONCERN

In this section, we extend the formulations of the basic models to the problems that we consider in this study.

2.5.1. The FJS Problem with Working Time Constraints

When working time constraints are imposed on the FJS problem, a limit is forced on the actual time that a machine operates. That is, no machine is allowed to operate for more than a given total working time of T units. Consequently, working time of a machine is the sum of the processing times of the jobs assigned to that machine. We assume that the total processing time, $\sum_{j=1}^{n} p_j$, is greater than T, so that the working time constraint is not redundant.

Working time constraints may be imposed in both the tactical or operational problems. Namely, we add the following constraint to the TFJS and OFJS models presented in the second chapter to impose working time limit:

$$\sum_{j=1}^{n} p_j x_{jk} \le T_j \qquad k = 1,\ldots,m \tag{8}$$

We deal with the operational version of the problem in this study. We hereafter denote the OFJS problem with working time constraints as the OFJSW problem, and constraint set (8) as *T*-constraints. We assume that $T_j = T$, i.e. the working time limit is the same for all machines. However, our results also apply to the arbitrary T_j case.

2.5.2. The FJS Problem with Spread Time Constraints

The spread time for a machine is defined as (d_j - r_i) where j is the last job and i is the first job processed by the machine. A limit imposed on the total spread time of the machines necessitates a new set definition and a constraint for both TFJS and OFJS models. We define set I_j as the incompatibility set for job j as:

$I_j = \{\, i > j : r_i < d_j$ or $d_i - r_j > S\}$.

Note that the assignment of jobs j and $i \in I_j$ to the same machine violates the spread time constraint.

Spread time constraints are applicable to both the TFJS and OFJS problems. We add the following constraint to the TFJS and OFJS models to impose spread time:

$$x_{ik} + x_{jk} \le 1 \qquad k = 1,\ldots,m,\ j = 1,\ldots,n-1,\ i \in I_j \tag{9}$$

We deal with the operational version of the problem in this study. We hereafter denote the OFJS problem with spread time constraints as the OFJSS problem. We

assume that $S_j = S$, i.e. the spread time limit is the same for all machines. However, our results also apply to the arbitrary S_j case.

2.5.3. The OFJS Problem with General Weights

In the most general version of the OFJS problem, the weights are job and machine dependent, i.e. the weight of a job j also depends on the machine to which it is assigned. The weight definition then becomes:

w_{jk} : weight of job j if it is processed on machine k.

The model for the OFJS problem with general weights then becomes:

Maximize $\sum_{k=1}^{m}\sum_{j=1}^{n} w_{jk} x_{jk}$

subject to (1), (2), and (3).

We hereafter denote the OFJS problem with general weights as the OFJSG problem. Note that the OFJSG model is a generalization of the OFJS model; when $w_{jk}=w_j$, the problem reduces to the OFJS model defined in Section 2.2.

The OFJS problem with eligibility constraints is a special case of the problem with general weights, where each machine is eligible to process only a subset of jobs. We hereafter denote this problem as the OFJSE problem. The weights for this problem can be defined as follows:

$$w_{jk} = \begin{cases} w_j, & \text{if machine } k \text{ is eligible to process job } j \\ 0, & \text{otherwise.} \end{cases}$$

2.6. RELATION BETWEEN THE OFJSW AND OFJSS PROBLEMS

In this section, we establish a relation between the optimal solutions of the OFJSW and OFJSS problems. Theorem 2.6.1 states this relation formally.

Theorem 2.6.1: An optimal solution of the OFJSW problem with a working time limit T is an upper bound on the optimal solution of the OFJSS problem with spread time limit S, when $T = S$.

Proof: Assume S^* is an optimal schedule to the OFJSS problem, and Z_{S^*} is the optimal objective function value. Note that S^* is feasible for the OFJSW problem when $T = S$. This implies $Z_{S^{**}} \geq Z_{S^*}$, where S^{**} is an optimal schedule to the OFJSW problem. Hence, $Z_{S^{**}}$ is a valid upper bound on the spread time problem. c

CHAPTER 3

THE OFJS PROBLEM WITH WORKING TIME CONSTRAINTS

Recall that, when working time constraints are imposed on the FJS problem, a limit is forced on the sum of the processing times of the jobs assigned to a machine. We first give a brief review of the literature and state the complexity of the problem.

As to the best of our knowledge, there are only two studies in literature regarding the TFJS problem with working time constraints. Both studies are authored by Fischetti, Martello and Toth (1989, 1992), and deal with the Bus Driver Scheduling Problem with the objective of minimizing the number of drivers necessary to perform all duties and a driver cannot work for more than a given working time T in a day. The authors prove that the problem is NP-hard in the strong sense through transformation from Bin Packing Problem, which is shown to be strongly NP-hard by Garey and Johnson (1979). In their first study, Fischetti *et al.* (1989) show that some special cases of the problem, including its preemptive version, can be solved in polynomial time. To solve the general problem, they present a Branch and Bound algorithm, and show that its performance is quite satisfactory for the data sets generated close to real-life situations. In their later study, Fischetti *et al*. (1992) provide some polynomial time approximation algorithms for the problem.

The only study in literature related with the OFJS problem with working time constraints (the OFJSW problem) is that of Bouzina and Emmons' (1996). The authors consider the preemptive version of the problem with the objective of maximizing the number of jobs processed, and they provide a polynomial time algorithm that solves this problem optimally. The authors prove that the preemptive version of the problem with the objective of maximizing total weight is NP-hard in

the ordinary sense through transformation from the knapsack problem, which is shown to be binary NP-hard by Garey and Johnson (1979). The authors also show that the feasibility problem of the nonpreemptive case is NP-complete, through transformation from the Bin Packing Problem.

We now settle the complexity of the OFJSW problem by the following corollary, which is a direct implication of Theorem 2.4.1.

Corollary 3.1: The OFJSW problem is NP-hard in the strong sense.

Proof: The TFJS problem with working time constraints is NP-hard in the strong sense (Fischetti *et al.*, 1989), so is our problem. Replace X with constraint set (8) in Theorem 2.4.1. The proof is immediate. c

The rest of this chapter is organized as follows: In Section 3.1, we analyze some polynomially solvable special cases of the problem. We present the specifics of our optimization procedure in Section 3.2. The results of our computational experiment will be presented in Chapter 6.

3.1. POLYNOMIALLY SOLVABLE SPECIAL CASES

In this section, we examine some special polynomially solvable cases for the OFJSW problem. These special cases may arise from either special structures of job parameters, or some relaxations of the problem. For each special case, we provide a polynomial time algorithm, prove that the algorithm solves this particular case, and settle its time complexity.

Special Case 1: All processing times are unity, i.e. $p_j = 1$, $\forall j$.

The optimization algorithm for this case lists the jobs in nonincreasing order of their weights, assigns jobs to the machines starting with the first job of the list, and continues until no more jobs can be inserted without violating feasibility. Below is the stepwise description of the algorithm.

Algorithm 3.1.1:

S1. Index the jobs in their nonincreasing order of w_j values.

S2. While satisfying working time constraint for k =1,…,m,

For j =1,…,n do:

If no machine is available at time r_j, do not schedule job j.

Else assign job j to the least loaded available machine at time r_j.

Theorem 3.1.1: Algorithm 3.1.1 solves the OFJSW problem when all processing times are unity.

Proof: Any feasible solution to the OFJSW problem can be improved in two ways. One way is to change the status of two jobs i and j such that job i is scheduled but job j is not, and $w_j > w_i$. In this case, an improved solution can be found by removing job i and inserting job j, without violating feasibility. Two cases may arise:

Case 1: $r_i = r_j$, and all machines are busy during $(r_j, r_{j+1}]$. Algorithm 3.1.1 gives priority to the jobs with higher weights. The algorithm assigns job j but not job i in this case, hence no such job pairs can exist in a solution generated by the algorithm.

Case 2: $r_i \neq r_j$, the machine that processes job i is idle during $(r_j, r_{j+1}]$, and is fully loaded. Algorithm 3.1.1 gives priority to the jobs with higher weights; hence no scheduled job on this machine can have a weight smaller than that of job j. Then, no such job pairs can exist in a solution generated by the algorithm.

The second way to improve a solution is to insert a job feasibly without altering the processing of other jobs. Note that Algorithm 3.1.1 terminates when no job can be inserted without violating feasibility. Note that in all possible cases a schedule formed by Algorithm 3.1.1 cannot be improved, hence it is optimal. c

Corollary 3.1.1: Algorithm 3.1.1 runs in $O(n \log n + nm)$ time.

Proof: The sorting operation in S1 takes $O(n \log n)$ time. In S2, we consider n jobs, and examine $O(m)$ machines for each job, hence, the time complexity of this step is $O(nm)$. Therefore, the overall complexity of the algorithm is $O(n \log n + nm)$. c

Special Case 2: All ready times are identical, i.e. r_j =r, "j.

The following $O(m \log n)$ algorithm solves this special case optimally. The algorithm simply selects m jobs with the highest weights, and assigns each to a different machine.

Algorithm 3.1.2:

S1. Select m jobs that have highest w_j values.

S2. Assign the selected jobs each to a different machine.

Theorem 3.1.2: Algorithm 3.1.2 solves the OFJSW problem when all ready times are identical.

Proof: A feasible solution to the OFJSW problem can be improved by changing the status of two jobs i and j such that job i is scheduled but job j is not, and $w_j > w_i$, or by inserting a job without altering the processing of other jobs. An improved solution can be found by exchanging jobs i and j. Algorithm 3.1.2 gives priority to the jobs with higher weights, hence, no such job pairs can exist in a solution generated by the algorithm. Furthermore, since all jobs should start at the same time, no insertion is possible. Therefore, a schedule formed by Algorithm 3.1.2 cannot be improved, i.e. it is optimal. c

Corollary 3.1.2: Algorithm 3.1.2 runs in $O(m \log n)$ time.

Proof: S1 defines the complexity of Algorithm 3.1.2. S1 chooses m jobs with highest w_j values out of n jobs in $O(m \log n)$ time. Hence, the overall complexity of the algorithm is $O(m \log n)$. c

Special Case 3: All weights are identical, i.e. $w_j = w$, $\forall j$, and preemption is allowed.

The objective reduces to the maximization of the number of jobs processed. The algorithm by Bouzina and Emmons (1996) solves this special case optimally.

Special Case 4: Preemption is allowed and the jobs can be processed partially, i.e. $x_{jk} \leq 1$, $\forall i, j$.

We analyze two subcases of this special case: $m = 1$ and $m > 1$.

Special Case 4.1: **Preemption is allowed, the jobs can be processed partially, and there is only one machine, i.e. $m = 1$**.

We develop an algorithm for this case that utilizes the idea of a continuous knapsack solution, as this special case resembles the linear relaxation of the Single Knapsack Problem. The algorithm assigns jobs, starting from the one with the highest weight, to the intervals within their processing window. It divides the time into intervals that have no arrivals and completions in between. The job is assigned if there is an available machine during that interval, and such an assignment does not violate the working time constraint. Below is the stepwise description of the algorithm.

Algorithm 3.1.3:

S1. Index the jobs in their nonincreasing order of w_j / p_j values.

S2. Let $\{t_1, t_2, ...\}$ be the sorted sequence of the r_js and d_js in chronological order with duplicates removed.

S3. While satisfying the working time constraint,

For $j = 1, \ldots, n$ do:

Fix $t_k = r_j$, $t_n = d_j$.

For each interval $[t_k, t_{k+1}), \ldots, [t_{n-1}, t_n)$ do:

If the machine is available in the interval, then schedule job j.

Theorem 3.1.3: Algorithm 3.1.3 solves the OFJSW problem when $m = 1$, $x_{jk} \leq 1$, and preemption is allowed.

Proof: As $x_{jk} \leq 1$, and r_js are integers, we can treat each job i as $v = p_i$ subjobs with unit processing times. If we let r_{ij} and w_{ij} be the ready time and the weight of the j^{th} subjob of job i, then the ready times of these subjobs can be defined as $r_{i1} = r_i$, $r_{i2} = r_i + 1$, …, $r_{iv} = r_i + v - 1$, and weights as $w_{ij} = w_i / p_i$ for all j. Note that, an optimal solution to the problem with subjobs provides an optimal solution to the original problem (OFJSW), when we impose a constraint that requires either all subjobs of the same job are processed consecutively or none are processed. As we relax this constraint, the resulting problem is the unit processing time problem with $n_1 = \sum_{i=1}^{n} p_i$ jobs on a single machine, which can be solved by Algorithm 3.1.1.

Note that, Algorithm 3.1.3 is a more efficient implementation of the procedure defined above, where the intervals of unit length are merged if they have the same set of available jobs. This reduces the number of intervals from $O(n_1)$ to $O(2n)$. Hence, the solution generated by Algorithm 3.1.3 is optimal. c

Corollary 3.1.3: Algorithm 3.1.3 runs in $O(n^2)$ time.

Proof: The complexity of Algorithm 3.1.3 is computed as follows. The sorting operations in S1 and S2 take $O(n \log n)$ and $O(2n \log 2n)$ times, respectively. In S3, there are $O(n)$ jobs, and at most $2n$ intervals are checked for each job, the time complexity of this step is therefore $O(2n^2)$. The overall complexity of the algorithm is then $O(n^2)$. c

<u>Special Case 4.2</u>: Preemption is allowed, the jobs can be processed partially, and the processing times are identical to weights, i.e. $w_j = p_j$, $\forall j$.

The problem reduces to the problem of maximization of the total processing time. The following algorithm solves this special case optimally. After sorting the jobs by their p_j values, the algorithm considers each interval separately in chronological

order, and assigns each of the $O(m)$ available jobs with highest p_j values to one of the machines.

Algorithm 3.1.4:

S1. Index jobs in their nonincreasing order of p_j values.

S2. Let $\{t_1, t_2,...\}$ be the sorted sequence of the r_js and d_js in chronological order with duplicates removed. Let S_a be the sorted set of jobs available in the interval $[t_a, t_{a+1})$ for a=1, 2,...

S3. While satisfying working time constraint for k =1,…,m,

For each interval $[t_a, t_{a+1})$ for a=1, 2,... do:

Assign Min$\{|S_a|, m\}$ jobs of the order to m machines.

Theorem 3.1.4: Algorithm 3.1.4 solves the OFJSW problem when $w_j = p_j$, $x_{jk} \leq 1$, and preemption is allowed.

Proof: As in the previous case, we can treat each job i as $v = p_i$ subjobs with unit processing times. If we let w_{ij} be the weight of the j^{th} subjob of job i, the weights become $w_{ij} = w_i / p_i = 1$ for all j. Hence, there are $n_1 = \sum_{i=1}^{n} p_i$ jobs with equal weights. The problem reduces to the maximization of the number of jobs with unit processing times, which can be optimally solved by Algorithm 3.1.1.

Note that, Algorithm 3.1.4 is a more efficient implementation of the procedure defined above, where the intervals of unit length are merged if they have the same set of available jobs, thereby reducing the number of ready times to $O(n)$. Hence, the solution generated by Algorithm 3.1.4 is optimal. c

Corollary 3.1.4: Algorithm 3.1.4 runs in $O(n \log n + nm)$ time.

Proof: The complexity of Algorithm 3.1.4 is computed as follows. The sorting operations in S1 and S2 take $O(n \log n)$ and $O(2n \log 2n)$ times, respectively. In S3, $2n$ intervals are to be examined for m

machines, leading to a time complexity of $O(2nm)$. Therefore, the overall complexity of the algorithm is $O(n \log n + nm)$. c

3.2. OPTIMIZATION PROCEDURE

Recall that the problem we are considering is NP-hard. This suggests the use of implicit enumeration techniques for finding optimal solutions. We propose a branch and bound algorithm that employs several reduction and bounding mechanisms.

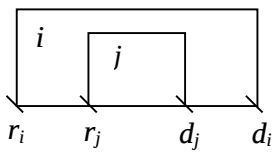
In Section 3.2.1, we provide some properties of an optimal solution. In Sections 3.2.2 and 3.2.3, we present our lower and upper bounding procedures. We provide the implementation details of the branch and bound algorithm in Section 3.2.4.

3.2.1. Properties of an Optimal Solution

In this section, we present some properties that characterize the structure of an optimal solution. The first property is stated through the following theorem:

Theorem 3.2.1: If $w_j > w_i$, $r_i \leq r_j$ and $d_i \geq d_j$, then any optimal solution that includes job i should also include job j.

Proof: Consider the situation represented in the following figure:



Assume that there is an optimal solution that includes job i, but not job j. Let the objective function value of this solution be Z_i. Assume we remove job i, and assign job j to the machine that is freed by removing job i. Note that, such an interchange can be done feasibly without alternating any processing as $r_i \leq r_j$ and $d_i \geq d_j$. The objective function value of this new solution becomes Z_j =

$Z_i - w_i + w_j$. Note that $Z_j > Z_i$, as $w_j > w_i$. Hence, a solution that includes job i, but not job j, cannot be optimal. c

The following theorem establishes another property of an optimal solution, which is a direct implication of Theorem 3.2.1. We hereafter say that job j dominates job i if $w_j > w_i$, $r_i \le r_j$ and $d_i \ge d_j$, and let S_i be the set of jobs that dominate job i.

Theorem 3.2.2: If $|S_i| \ge m$, then job i is not processed in any optimal solution.

Proof: We have already shown that, if job j dominates job i, a solution that includes job j and excludes job i is better than a solution that includes job i and excludes job j. If the number of such dominating jobs is no less than the number of machines, job i cannot be in any optimal solution. c

The jobs that satisfy the conditions of Theorem 3.2.2 can be eliminated, thereby reducing the size of the problem.

We generalize Theorem 3.2.1 by considering the fact that there may exist a job that is dominated by a set of jobs, however not by a single job. We try to detect this job set by using a Longest Path algorithm. Assume $D_i = \{ j \mid r_j \ge r_i$, and $d_j \le d_i \}$, that is the set of jobs that can be replaced by job i without altering any processing sequence. Let LP_i be the longest path formed by the jobs in D_i. The following example illustrates the set. Assume $n = 5$.

j	1	2	3	4	5
r_j	1	2	2	4	5
p_j	5	1	1	2	1
w_j	7	5	4	1	3

For $i = 1$, $D_i = \{2, 3, 4, 5\}$ and $LP_i = \{2, 5\}$.

Note that, when no jobs in set D_i overlap, a longest path solution includes all jobs, hence $LP_i = D_i$.

The following theorem establishes a relation between job i and the jobs in set D_i.

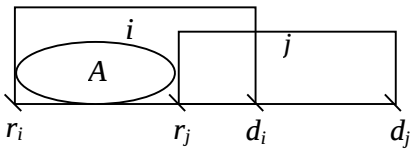
Theorem 3.2.3: In an optimal solution, if job i is processed and $\sum_{j \in LP_i} w_j > w_i$, then at least one job in LP_i should be processed.

Proof: The theorem implies that, there cannot be an optimal solution that includes job i, but not any job in LP_i. Otherwise, job i would be replaced by the jobs in LP_i and the solution would improve by $\sum_{j \in LP_i} w_j - w_i > 0$ units. c

Theorems 3.2.4 and 3.2.5 establish some other properties of an optimal solution.

Theorem 3.2.4: If $r_i < r_j < d_i < d_j$, $w_i > w_j$, $p_i \leq p_j$, and no job is available for processing during $[r_i, r_j]$, then any optimal solution that includes job j should also include job i.

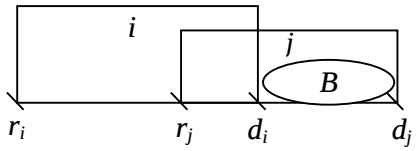
Proof: The condition of the theorem is illustrated by the following figure:



Consider an optimal solution that includes job j, but not job i. Let the resulting objective function value be Z_j. Note that, there is no job in set A. Assume we remove job j from the solution, and assign job i to the machine that is freed by removing job j. The objective function value of this new solution becomes $Z_i = Z_j - w_j + w_i$. Note that $Z_i > Z_j$, as $w_i > w_j$. Hence, a solution that includes job j, but not job i, cannot be optimal. c

Theorem 3.2.5: If $r_i < r_j < d_i < d_j$, $w_j > w_i$, $p_j \leq p_i$, and no job is available for processing during $[d_i, d_j]$, then any optimal solution that includes job i should also include job j.

Proof: The condition of the theorem is illustrated by the following figure:



Consider an optimal solution that includes job *i*, but not job *j*. Let the objective function value of this solution be Z_i. Note that in the figure there is no job in set *B*. Assume we remove job *i* from the solution, and assign job *j* to the machine that is freed by removing job *i*. The objective function value of this new solution becomes $Z_j = Z_i - w_i + w_j$. $Z_j > Z_i$ since $w_j > w_i$. Hence, a solution that includes job *i*, but not job *j*, cannot be optimal. c

3.2.2. Upper Bounds

Consider the surrogate relaxation of our problem where *T*-constraints are represented by:

$$\sum_{k=1}^{m}\sum_{j=1}^{n} p_j x_{jk} \le mT \qquad (8')$$

This representation reduces our problem to the well-known Single Knapsack Problem with capacity *mT* in the absence of overlapping constraints. The Single Knapsack Problem is known to be NP-hard in the ordinary sense (Garey and Johnson, 1979), and so is the surrogate relaxation of our problem. We show through Theorem 3.2.6 that the surrogate relaxation can be solved in polynomial time, when partial processing and preemptions are allowed. Hence, an optimal solution with these relaxations provides an upper bound to our problem.

We first state an optimization algorithm, Algorithm 3.2.1, to this relaxed version, and prove its optimality through Theorem 3.2.6.

Algorithm 3.2.1:

S0. Let $T' = mT$.

Index jobs in their nonincreasing order of w_j / p_j values.

Let $\{t_1, t_2,...\}$ be the sorted sequence of the r_js and d_js in chronological order with duplicates removed for j=1,...,n.

S1. While satisfying constraint $\sum_{k=1}^{m}\sum_{j=1}^{n} p_j x_{jk} \le T'$,

For j = 1,…,n do:

Let $t_k = r_j$, $t_l = d_j$.

For each interval $[t_i, t_{i+1})$ for $i = k,\ldots,l$ do:

If no machine is available in interval i or $T' < [t_i, t_{i+1})$, then do not schedule job j in interval i,

Else, assign job j to one of the machines available through the interval giving priority to the machine it was previously assigned (to reduce preemptions).

If job j is assigned to interval $[t_i, t_{i+1})$, update UB and T' as:

$$UB = UB + \frac{[t_i, t_{i+1})}{p_j} w_j \text{ , } T' = T' - [t_i, t_{i+1}).$$

Theorem 3.2.6: Algorithm 3.2.1 solves the surrogate relaxation of the OFJSW problem, when $x_{jk} \le 1$, and preemption is allowed.

Proof: The relaxed problem reduces to the case where there is a single machine that allows m overlaps at-a-time. If we further relax the constraint that requires full processing of the jobs and allow preemption, then the resulting problem can be treated as a unit processing time problem with the following analogy: There are p_i subjobs of job i where subjob j has a ready time of $r_i + j - 1$, i.e. subjob 1 of subjob i has a ready time of r_i, subjobs 2 and 3 have respective ready times of $r_i + 1$ and $r_i + 2$, and so on. The weight of each subjob of job i is then w_i / p_i. The resulting problem with $n_1 = \sum_{i=1}^{n} p_i$ jobs can be solved by Algorithm 3.1.1.

Algorithm 3.2.1 is a more efficient implementation of the procedure defined above, where the intervals of unit length are

gathered if the set of available jobs are the same. This reduces the number of intervals from $O(n_1)$ to $O(2n)$. Hence, the solution generated by Algorithm 3.2.1 is optimal. c

Corollary 3.2.1: Algorithm 3.2.1 runs in $O(n^2m)$ time.

Proof: The complexity of Algorithm 3.2.1 is computed as follows: The sorting operations in S1 and S2 take $O(n \log n)$ and $O(2n \log 2n)$ times, respectively. In S3, there are $O(n)$ jobs, and for each $O(2n)$ intervals $O(m)$ machines are to be checked for each job. Therefore, this step is executed in $O(2n^2m)$ time. The overall complexity of the algorithm becomes $O(n^2m)$. c

We hereafter let UBW_1 be the optimal solution to the surrogate relaxation when partial processing and preemptions are allowed.

Another upper bound can be found by relaxing T-constraints. Note that this case reduces to the OFJS problem, which can be solved in polynomial time by the Minimum Cost Network Flow (MCNF) algorithm. Hence, when there is a single machine, an optimal solution to the relaxed version can be found by a shortest path problem where costs on arcs are negative job weights. The results of our initial experiments have revealed that employing MCNF algorithm to each partial solution of branch and bound algorithm is very time consuming, i.e. the effort spent to calculate the bound outweighs the savings obtained through partial solution eliminations. However when there is a single feasible machine in the partial solution due to time constraints, we employ the shortest path algorithm and obtain satisfactory results. We hereafter let UBW_2 denote an optimal solution to the shortest path problem with single feasible machine.

3.2.3. Lower Bounds

The lower bounds are used to find quick and complete solutions particularly for the problems of higher sizes. Moreover, in the branch and bound approach, by

providing initial feasible solutions, they can enhance the efficiency of the solutions by eliminating non-promising partial schedules. Our lower bounding procedure that is given in Algorithm 3.2.2 is a simple and greedy construction algorithm.

Algorithm 3.2.2:

S1. a. Index jobs in their nonincreasing order of w_j / p_j values.

b. While satisfying time constraint for k =1,…,m, do:

For j = 1,…,n do:

If no machine is available during [r_j, d_j], do not schedule job j,

Else, assign job j to one of the available machines in the interval.

c. Let LB_1 be the resulting objective function value.

S2. a. Index jobs in their nonincreasing order of w_j values.

b. Same as Step S1.b.

c. LB_2 is the resulting objective function value.

S3. a. Index jobs in their nondecreasing order of p_j values.

b. Same as Step S1.b.

c. LB_3 is the resulting objective function value.

S4. Set LBW = Max {LB_1, LB_2, LB_3}.

Corollary 3.2.2: Algorithm 3.2.2 runs in $O(n \log n) + O(nm)$ time.

Proof: The complexity of Algorithm 3.2.2 is computed as follows: The sorting operations in S1.a, S2.a and S3.a take $3*O(n \log n)$ time. In steps S1.b, S2.b and S3.b, there are $O(n)$ jobs, and $O(m)$ machines are to be checked for each job, therefore the time complexity of these steps is $3*O(nm)$. The overall complexity of the algorithm is $O(n \log n) + O(nm)$. c

3.2.4. Branch and Bound Algorithm

In our Branch and Bound (BB) procedure, we assume that the jobs are indexed in their nondecreasing order of ready times. A node at level l corresponds to a partial solution where the decisions about the first l jobs, having the smallest ready times,

are set. For each node emanating from level l, there are $O(m+1)$ decisions: one for rejecting job $l+1$ and one for processing job $l+1$ on machine k, $k=1,\ldots,m$. The assignment to machine k is feasible if the machine does not process any job that overlaps with job $l+1$ and such an assignment does not violate the time constraint. Moreover, we increase the efficiency of branching when there is more than one machine with no processing load. We consider only one machine of such type, to eliminate the duplication of the partial sequences. Specifically, we consider only the first machine for the first job, thereby only two decisions. If job 1 is rejected, then there are two decisions for job 2: rejection or assignment to machine 1 as all machines are empty. If job 1 is assigned there are three decisions for job 2: rejection, assignment to machine 1 and assignment to machine 2. Similarly, for job l there are $O(\text{Min}\{m,l\}+1)$ alternatives, and if only R ($<m$) machines are in use, then there are $R+2$ decisions for job l: one for rejecting, and one for assignment to machine k, $k=1,\ldots,R+1$. We illustrate our BB tree in Figure 3.1.

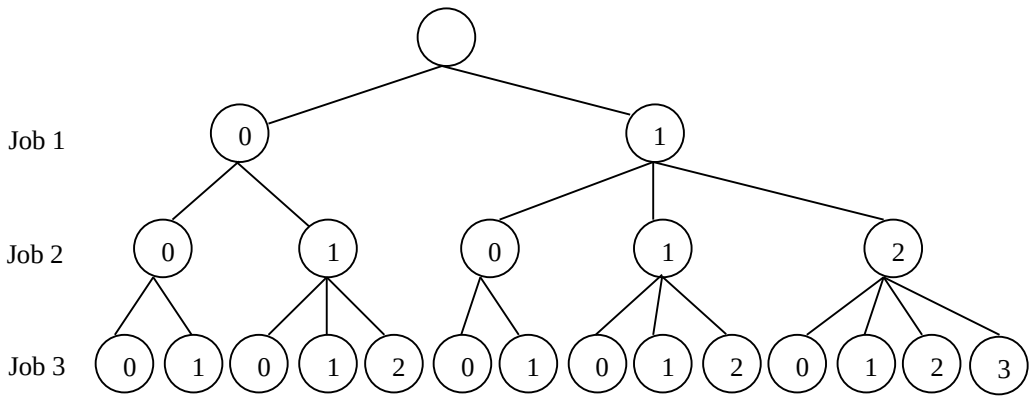


Figure 3.1. Branch and Bound tree for the OFJSW problem.

Our algorithm uses a depth-first strategy that introduces one job at a time to the partial schedule. The strategy has relatively low memory requirements and complete solutions are likely to be reached earlier, which enables us to get high quality lower bounds earlier, thereby increasing the power of fathoming. We

generate all subproblems of a node and select the one having the maximum upper bound for branching. The remaining subproblems are sorted in nonincreasing order of their upper bounds and stored in a candidate list. Each time the current node is revisited, the next subproblem is removed from the list and selected for branching.

We remove a job from consideration if its assignment to any machine leads to an infeasible solution. We also remove a machine from consideration if no job can be assigned on it without violating overlapping or *T*-constraints.

We first reduce the problem size by employing the result of Theorem 3.2.2. We remove job i if $|S_i| \geq m$, where S_i is the set of jobs that dominate job i.

During the implementation of the BB procedure, we employ Theorems 3.2.1 and 3.2.4 as follows: We fathom the node representing the rejection of job i if there exists a scheduled job j such that $w_i > w_j$, $r_j \leq r_i$ and $d_j \geq d_i$ (otherwise Theorem 3.2.1 is violated). If there exists an unscheduled job i, and no job k that satisfies $r_i \leq r_k \leq d_k \leq r_j$ is scheduled, then job j such that $r_j \leq d_i < d_j$ and $w_i > w_j$ cannot be assigned to any machine (otherwise Theorem 3.2.4 is violated).

We also calculate a lower bound for a partial schedule with set U of assigned jobs, when the total processing time of unassigned jobs is no more than the slack of our surrogate constraint, i.e. when:

$$\sum_{j \in U} p_j \leq mT - \sum_{k=1}^{m} L_k \,,$$

where L_k is the load of machine k in the partial schedule. If the resulting solution contains all jobs in U, it means that the solution is optimal for the given partial schedule, and we can fathom the node and update the best lower bound if a better solution is reached.

We calculate an upper bound, for each node that cannot be eliminated by the above conditions. If there is a single machine to which further assignments can be done, we use Max{UBW_1, UBW_2}. Otherwise we merely use UBW_1. We discard the node

if its upper bound is no greater than the best-known lower bound. We update the best-known lower bound when a complete solution with a better objective function value is found.

The computational experience of our optimization procedure for the OFJSW problem will be presented in Chapter 6.

CHAPTER 4

THE OFJS PROBLEM WITH SPREAD TIME CONSTRAINTS

Recall that the spread time for a machine is defined as (d_j - r_i) where j is the last job and i is the first job processed by the machine. We first give a brief review of the literature for the problem.

To the best of our knowledge, Fischetti, Martello and Toth (1987) are the unique authors who report on FJS with spread time constraints. In their two studies, the authors deal with the Bus Driver Scheduling Problem with an additional constraint on the spread time of the drivers.

In their first study, the authors prove that TFJS with spread time constraints is NP-hard in the strong sense through transformation from Circular Arc Coloring Problem, which is known to be NP-hard in the strong sense (Garey and Johnson, 1979). They provide the mathematical representation of the problem, and present a graph theoretical model based on vertex coloring problem. They provide an $O(n \log n)$ optimization algorithm for the preemptive version of the problem. Some lower bounds with worst-case performance ratios are developed, and some dominance criteria that help to reduce the problem size are defined. They also describe a Branch and Bound algorithm for solving the problem.

In their later study, Fischetti *et al.* (1992) provide some polynomial time approximation algorithms. One such algorithm is a greedy one that runs in $O(n \log n)$ time. Two other $O(n \log n)$ time algorithms that are based on the preemptive relaxation of the model are presented. The authors also present two other greedy algorithms and an algorithm that is based on a longest path solution. Worst case

performance analyses of algorithms are also presented and all algorithms are compared by their running times and the quality of solutions.

To the best of our knowledge, there is no study in literature related with the OFJS problem with spread time constraints (the OFJSS problem), as in the case with working time constraints. Now, we settle the complexity of the OFJSS problem by the following corollary, which is a direct implication of Theorem 2.4.1.

Corollary 4.1: The OFJSS problem is NP-hard in the strong sense.

Proof: The TFJS problem with spread time constraints is NP-hard in the strong sense, so is our problem. Plug X = {constraint set (9)} in Theorem 2.4.1. The proof is immediate. c

We also show that the preemptive version of the OFJSS is NP-hard in the ordinary sense, and prove through transformation from Knapsack problem, which is known to be binary NP-hard (Garey and Johnson, 1979). The following theorem states this result formally:

Theorem 4.1: The preemptive version of OFJSS problem is NP-hard in the ordinary sense.

Proof: Assume that our concern is to find whether there exists a feasible subset of jobs F such that $\sum_{i \in F} w_i \geq w$, where w is a positive integer.

To prove the result, we reduce the Knapsack problem to the preemptive OFJSS problem. In an instance of a knapsack problem, we are given a set $\{1,...,p\}$ and with each one of its elements j is associated two positive integers z_j and v_j, reflecting the size and the value of j, respectively for $j = 1,..., p$. Moreover, let Z and V be two positive integers. We are asked to find a subset $U \subseteq \{1,...,p\}$ such that $\sum_{i \in U} v_i \geq V$ and $\sum_{i \in U} z_i \leq Z$.

Given this instance of Knapsack problem, we construct the following instance of the preemptive OFJSS problem:

- $n = p$ jobs
- $r_j = 0$, $d_j = z_j$, $w_j = v_j$ for j =1,..., n
- $m = p$ machines
- $S = Z / p$
- $w = V$

Note that, solving this instance will solve the knapsack instance as well. The Knapsack Problem is NP-hard in the ordinary sense, so is the preemptive version of the OFJSS problem. c

The rest of this chapter is organized as follows: Some polynomially solvable cases of the OFJSS problem are presented in Section 4.1. We describe our optimization procedure for the OFJSS problem in Section 4.2. The results of our computational experiment for the OFJSS problem will be presented in Chapter 6, together with other problems.

4.1. POLYNOMIALLY SOLVABLE SPECIAL CASES

In this section, we determine some polynomially solvable special cases for the OFJSS problem. These cases arise from special forms of job parameters, and some relaxations of the problem. For each special case, we provide a polynomial time algorithm, and settle its time complexity.

Special Case 1: There is only one machine, i.e. m =1.

Once the first job j in the schedule is fixed, $[r_j, r_j + S]$ becomes the time span of the machine. Hence, the set of jobs that can be processed without violating the spread time constraint are the ones that arrive no earlier than $(r_j + p_j)$ and complete no later than $(r_j + S)$. Then, given the first job, the optimal solution can be found in $O(n^2)$ time by the Longest Path algorithm using weights as arc costs (equivalently the Shortest Path (SP) algorithm using negative weights). As there are n candidates for the first job, LP algorithm is employed $O(n)$ times. The following algorithm solves this special case optimally.

Algorithm 4.1.1:

S1. Solve the following Shortest Path problem, $SP_j, \forall j$, by fixing job j to the first position:

Determine the set of jobs $A_j = \{k: r_k > r_j + p_j,\ d_k \leq r_j + S\}$ that can be processed in the interval $[r_j + p_j,\ r_j + S]$. Create a source node, s, and a sink node, t. Create arcs of length zero between all consecutive nodes. Create an arc (i,j) of length $-w_i$ between two non-overlapping jobs i and j (with $r_j > r_i$). If there is no such job j, connect i to t with length $-w_i$.

Z_j = Objective function value of SP_j.

S2. Optimal objective function value is Z =(-1)* $\underset{j}{Min}\{SP_j\}$.

Consider the problem instance of Example 1, given by Figure 2.1. Assume that S=15, and job 4 is fixed to the first position on the machine in an iteration of the above algorithm. Figure 4.1 illustrates the shortest path network to be solved when job 4 is the first job.

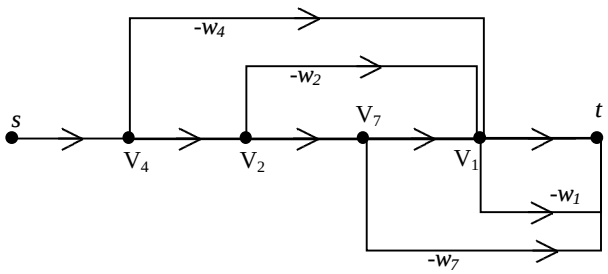


Figure 4.1. Example network representation for Algorithm 4.1.1

Theorem 4.1.1: Algorithm 4.1.1 solves the OFJSS problem when there is only one machine.

Proof: Note that, if the first job of the optimal solution was known, then the optimal schedule could be found by using SP algorithm considering the set of jobs defined in S1. Hence, the optimal

solution can be found by considering all jobs for the first position and selecting the one yielding maximum total weight. c

Corollary 4.1.1: Algorithm 4.1.1 runs in $O(n^3)$ time.

Proof: The complexity of Algorithm 4.1.1 is computed as follows. In S1, a Shortest Path problem is solved for each of the n jobs in $O(n^2)$ time. The time complexity of this step is therefore $O(n^3)$. In S3, minimum value among all objective function values is found in $O(\log n)$ time. Hence, the overall complexity of the algorithm hence is $O(n^3)$. c

Special Case 2: Preemption is allowed and the jobs can be processed partially, i.e. $x_{jk} \leq 1$, $\forall i, j$.

Once the first m jobs (first job on each machine) are fixed, then the set of jobs that can be processed without violating the spread time constraint, A_T, are automatically determined with a procedure similar to Algorithm 4.1.1. Next, the algorithm considers each idle interval in chronological order for the jobs in A_T, and assigns the jobs with maximum w_j / p_j values to the machines without violating the spread time constraint.

Algorithm 4.1.2:

S1. For each of the m-tuples out of n jobs:

Assign m jobs to the first positions of m machines,

Index the machines in nondecreasing order of their start times.

Let $t = \underset{j \in F}{Min}\{d_j\}$, where F is the set of first jobs, and

$T = r_{j_m} + S$, where j_m is the first job on machine m (last machine).

Let $A_T = \{k : r_k \geq t, d_k \leq T, k \text{ is not a first job}\}$.

Index jobs in A_T in nonincreasing order of their w_j / p_j values,

Let $\{t_1, t_2, ...\}$ be the sorted sequence of the r_js and d_js of the jobs in A_T in chronological order with duplicates removed,

Let J_a be the set of jobs in A_T available during $[t_a, t_{a+1})$ for a=1, 2,...

Let K_a be the set of machines available during $[t_a, t_{a+1})$ for a=1, 2,...

While satisfying the capacity constraint for k =1,…,m,

For each interval $[t_a, t_{a+1})$ for a=1, 2,... do:

Assign first Min{$|J_a|$, $|K_a|$} jobs in J_a to the available machines with minimum indices.

S2. Determine the maximum objective function value among all $\binom{n}{m}$ candidates.

Theorem 4.1.2: Algorithm 4.1.2 solves the OFJSS problem when $x_{jk} \leq 1$, and preemption is allowed.

Proof: If first jobs of the schedule were known, we could determine the set of jobs that can be processed without violating the spread time constraint on each machine and on each interval $[t_a, t_{a+1})$. Note that, the algorithm considers each interval separately, and assigns the most-weighted jobs to the machines with earliest start times. As we generate all possible alternatives, i.e. all m-tuples of jobs, and select the alternative yielding the best objective function value, the solution generated by Algorithm 4.1.2 is optimal. c

Corollary 4.1.2: Algorithm 4.1.2 runs in $O(n^m)$ time.

Proof: The complexity of Algorithm 4.1.2 is solely determined by generating the m-tuples out of n jobs, which can be done in $\binom{n}{m}$ ways. Note that,

$$\binom{n}{m} = \frac{n!}{m!(n-m)!} = n(n-1)...(n-m+1).$$

Hence, $O\binom{n}{m} = O\left(n^m\right)$, and Algorithm 4.1.2 runs in $O(n^m)$ time. c

The complexity result implies that the problem is polynomially solvable when m is fixed, and NP-hard in the ordinary sense when m is variable.

We illustrate the execution of Algorithm 4.1.2 through an example problem:

Example 2: Ten jobs are to be scheduled on 3 parallel identical machines. Let S = 11 for all machines. Table 4.1 shows the values for job parameters, where jobs are indexed in nonincreasing order of their w_j / p_j values.

Table 4.1. Data for Example 2

j	1	2	3	4	5	6	7	8	9	10
r_j	10	5	7	12	3	4	18	16	0	6
d_j	12	9	14	20	8	11	24	20	6	13
w_j	3	5	8	9	6	7	6	4	5	4

The first step of Algorithm 4.1.2 proceeds as follows:
Let the first jobs be jobs 9, 6, and 8, for machines 1, 2, and 3, respectively.
The machines are already indexed by their operation start times.

Start time of machine 1 = 0,

Completion time of machine 1 = 0 + 11 = 11.

Start time of machine 2 = 4,

Completion time of machine 2 = 4 + 11 = 15.

Start time of machine 3 = 16,

Completion time of machine 3 = 16 + 11 = 27.

Then, t = 6, T = 27. A_T = {1,3,4,7,10}.

The sorted sequence of the r_js and d_js of the jobs in A_T is {6, 7, 10, 12, 13, 14, 18, 20, 24}. The assignments by Algorithm 4.1.2 are as follows:

Interval	J_a	K_a	**Assignments**
[6,7)	10	1	job 10→machine 1
[7,10)	3,10	1	job 3→machine 1, job 10 not assigned
[10,11)	1,3,10	1	job 1→machine 1, jobs 3 and 10 not assigned, machine 1 full

[11,12)	1,3,10	2	job 1→machine 2, jobs 3 and 10 not assigned
[12,13)	3,4,10	2	job 3→machine 2, jobs 4 and 10 not assigned
[13,14)	3,4	2	job 3→machine 2, job 4 not assigned
[14,15)	4	2	job 4→machine 2, machine 2 full
[18,20)	7	∅	job 7 not assigned
[20,24)	7	3	job 7→machine 3

Total weight = $w_9 + w_6 + w_8 + w_1 + (5/7)* w_3 + (1/8)*w_4 + (4/6)*w_7 + (1/7)* w_{10}$

= 30.41.

All $\binom{n}{m}$ job combinations are evaluated in S1 of Algorithm 4.1.2. Note that, a job may be processed partially when a spread time limit is reached. For example, when the algorithm assigns job 4 in interval [14,18), machine 2 reaches its spread time limit at time 15. Hence, job 4 is assigned to machine 2 only during [14,15).

When the first jobs on machines are already set, the following algorithm, which is a more efficient implementation of Algorithm 4.1.2, yields an optimal solution.

<u>Special Case 2.1</u>: Preemption is allowed, the jobs can be processed partially, and the first jobs on machines are determined.

Algorithm 4.1.3, which solves this special case optimally, proceeds as follows: Once the first jobs on the machines are fixed, their spread times are already determined. The algorithm considers each interval in chronological order, and assigns each of the $O(m)$ jobs with highest w_j / p_j values to one of the machines without violating the spread time constraint.

Algorithm 4.1.3:

S1. For a given set of m first jobs on each of the machines:

Let $t = \underset{j \in F}{Min}\{d_j\}$ and $T = \underset{j \in F}{Max}\{r_j + S\}$, where F is the set of first jobs.

Let $A_T = \{k : r_k \geq t, d_k \leq T, k \text{ is not a first job}\}$. Set $r_{j_k} = 0$ for all $j_k \in F$ to block processing during [0, r_{j_k}], where $j_k \in F$ is the first job on machine k.

S2. Index jobs in A_T in nonincreasing order of their w_j / p_j values.

S3. Let $\{t_1, t_2,...\}$ be the sorted sequence of the r_js and d_js of the jobs in A_T in chronological order with duplicates removed,

Let J_a be the set of jobs in A_T available during $[t_a, t_{a+1})$ for a=1, 2,...

Let K_a be the set of machines available during $[t_a, t_{a+1})$ for a=1, 2,...

S4. While satisfying the spread time constraint for k =1,…,m,

For each interval $[t_a, t_{a+1})$ for a=1, 2,... do:

Assign first Min{$|J_a|$, $|K_a|$} smallest-indexed jobs in J_a to the available machines.

Theorem 4.1.3: Algorithm 4.1.3 solves the OFJSS problem when $x_{jk} \leq 1$, preemption is allowed, and the first jobs on machines are set.

Proof: Once the first jobs of the schedule are set, the set of jobs that can be processed without violating the spread time constraint on each machine and on each interval $[t_a, t_{a+1})$ can be found. The algorithm considers each interval separately, and assigns the most-weighted jobs to the machines with earliest start times. c

Corollary 4.1.3: Algorithm 4.1.3 runs in $O(n \log n + nm)$ time.

Proof: The complexity of Algorithm 4.1.3 is computed as follows. The sorting operations in S1, S2 and S3 take $2O(\log n)$, $O(n \log n)$ and $O(2n \log 2n)$ times, respectively. The time complexity of S4 is $O(2nm)$ as $2n$ intervals are to be examined for m machines. Therefore, the overall complexity of the algorithm is $O(n \log n + nm)$. c

<u>Special Case 4</u>: All machines have identical start times.

In this case, all machines are allowed to work during the same interval of length S. The problem reduces to finding the start time of the interval. The ready time of a

particular job j identifies the start time of all machines, and $[r_j, r_j + S]$ becomes the time span of operations. The set of jobs that can be processed without violating the spread time constraint are the ones that arrive no earlier than r_j, and complete no later than $(r_j + S)$. Given job j, the optimal solution can be found in polynomial time by solving a MCNF problem. Note that there are n candidates for job j. The following algorithm solves this special case optimally.

Algorithm 4.1.4:

S1. Solve the following MCNF problem, $MCNF_j, \forall j$, by fixing the start time to r_j:

Determine the set of jobs $A_j = \{k: r_k \geq r_j, d_k \leq r_j + S\}$. Solve a MCNF problem for the jobs in A_j using Algorithm 2.2.2. Z_j = Objective function value of $MCNF_j$.

S2. The maximum weight is Z =(-1)* $\underset{j}{Min}\{Z_j\}$.

Theorem 4.1.4: Algorithm 4.1.4 solves the OFJSS problem when all machines have identical start times.

Proof: If we know the start time for the machines, we can determine the set of jobs that can be processed within the spread time, and consider only this set. Note that, an optimal solution for this set can be found by Algorithm 2.2.2. As we generate such a problem for each job and select the one yielding the maximum weight, the solution found by Algorithm 4.1.4 is optimal. c

Corollary 4.1.4: Algorithm 4.1.4 runs in $O(n^4)$ time.

Proof: The complexity of Algorithm 4.1.4 is computed as follows. The time complexity of S1 is $O(n^4)$, as MCNF problem is solved for each of n ready times in $O(n^3)$ time. In S3, the minimum value among all objective function values is found in $O(\log n)$ time. Hence, the overall complexity of the algorithm is $O(n^4)$. c

4.2. OPTIMIZATION PROCEDURE

Recall that the OFJSS problem is NP-hard. This suggests the use of implicit enumeration techniques for finding optimal solutions. We propose a branch and bound algorithm that first fixes the first jobs on all machines (Phase 1), then finds the optimal assignments given the first jobs (Phase 2). In our branch and bound algorithm, we employ some properties that help to characterize the structure of an optimal solution of the OFJSS problem, and use these properties to reduce the problem size.

We first state three theorems that characterize the structure of an optimal solution of the OFJSS problem. These theorems are identical to the ones for the OFJSW problem. We omit the proofs, as they are trivial.

Theorem 3.2.1: If $w_j > w_i$, $r_i \leq r_j$ and $d_i \geq d_j$, then any optimal solution that includes job i should also include job j.

Theorem 3.2.2: If $|S_i| \geq m$, then job i is not processed in any optimal solution.

We reduce the problem size by eliminating all jobs that satisfy the condition of Theorem 3.2.2.

Theorem 3.2.3: In an optimal solution, if job i is processed and $\sum_{j \in LP_i} w_j > w_i$, then at least one job in LP_i should be processed.

We assume that the jobs are indexed in nondecreasing order of their ready times. In Phase 1 of our branch and bound procedure, a node at level l corresponds to a partial solution where the first jobs on the first l machines are set. For each node emanating from level l, there are $O(n)$ decisions for scheduling each of the $O(n)$ jobs as a first job on machine l+1, l=1,…,m. To eliminate the duplication of the schedules, we always branch to a node having a higher index than the parent node. We illustrate our branching tree for Phase 1 in Figure 4.2. Note that, at the leaf nodes (nodes at level m) of the partial tree, the first job on each machine is

determined. We generate all feasible combinations of the first jobs on machines by this branching phase, and find upper bounds. We sort the resulting partial solutions in nonincreasing order of their upper bounds and store them in a candidate list. As all solutions are kept active, we proceed in line with the breadth-first search strategy.

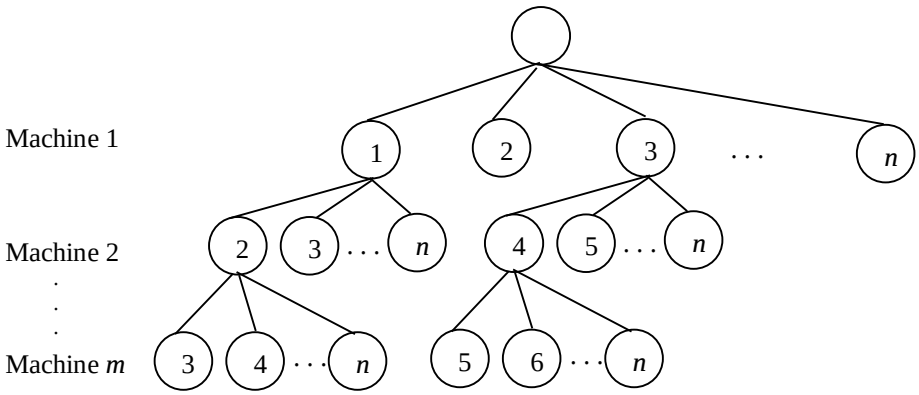
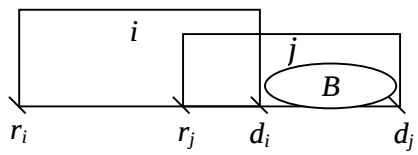


Figure 4.2. Branching tree for Phase 1

We employ the properties of the optimal solutions associated with the first jobs to eliminate non-promising candidates in Phase 1. Theorems 4.2.1 through 4.2.3 state these properties formally.

Theorem 4.2.1: Let $r_i \le r_j < d_i < d_j$, and $B = \{l \mid d_i \le r_l < d_j\}$. Then, any optimal solution that includes job i as a first job on a machine should also include job j, if $\sum_{l \in B} w_l + w_i < w_j$.

Proof: The situation is illustrated in the following figure:



Consider an optimal solution that includes job i as the first job on any machine (say machine k), but job j is not scheduled. Note that, the finishing time of machine k is $r_i + S$. Assume we remove job i from the solution, and assign job j to the machine that is freed by removing job i. The assignment of job j increases the objective function value by $w_j - \sum_{l \in B} w_l - w_i$ units when all jobs in B are also on machine i. Hence, $w_j - \sum_{l \in B} w_l - w_i > 0$ is a lower bound on the total increase. We can then conclude that, a solution that includes job i as a first job, but not job j at all, cannot be optimal. The objective function value for the new solution improves by at least $w_j - \sum_{l \in B} w_l - w_i$.

c

Theorem 4.2.2: Let $r_j < r_i$, and $C = \{l \mid r_j + S < d_l \leq r_i + S\}$. Then, the following are true:

i. If $r_i < d_j$, any optimal solution that includes job i as a first job on a machine should also include job j, if $\sum_{l \in C} w_l + w_i < w_j$.

ii. If $r_i \geq d_j$, any optimal solution that includes job i as a first job on a machine should also include job j, if $\sum_{l \in C} w_l < w_j$.

Proof: Consider an optimal solution that includes job i as the first job on any machine (say machine k), but job j is not scheduled.

i. Assume we remove job i from the solution, and assign job j to the machine that is freed by removing job i. The assignment of job j increases the objective function value by $w_j - \sum_{l \in C} w_l - w_i$ units when all jobs in C are also on machine k. Hence, $w_j - \sum_{l \in C} w_l - w_i > 0$ is a lower bound on the total increase. We can then conclude that, a solution that includes job i as a first job, but not job j at all, cannot be optimal. The

objective function value for the new solution improves by at least $w_j - \sum_{l \in C} w_l - w_i$.

ii. Assume we remove job *i* from the solution, and assign job *j* to the machine that is freed by removing job *i*. The assignment of job *j* increases the objective function value by $w_j - \sum_{l \in C} w_l$ units when all jobs in *C* are also on machine *k*. Then, $w_j - \sum_{l \in C} w_l > 0$ is a lower bound on the total increase. We can then conclude that, a solution that includes job *i* as a first job, but not job *j* at all, cannot be optimal. The objective function value for the new solution improves by at least $w_j - \sum_{l \in C} w_l$. c

The following theorem is a direct implication of Theorems 4.2.1 and 4.2.2. Let, $R_i = \{ j : r_i \le r_j < d_i < d_j$, and $\sum_{l \in B} w_l + w_i \le w_j \}$, where $B = \{ l \mid d_i \le r_l < d_j \}$, and $Q_i = \{ j : r_j < r_i$, and $\sum_{l \in C} w_l + w_i Y \le w_j \}$, where $C = \{ l \mid r_j + S < d_l \le r_i + S \}$ and

$$Y = \begin{cases} 1 & \text{if jobs } i \text{ and } j \text{ overlap } (r_i < d_j) \\ 0 & \text{otherwise} \end{cases} .$$

Theorem 4.2.3: If $|R_i \cup Q_i| \ge m$, then job *i* cannot be a first job in the optimal solution.

Proof: We have already shown that, if job $j \in R_i$ or $j \in Q_i$, we can find a better solution that includes job *j* and excludes job *i*, than a solution that includes job *i* as a first job and excludes job *j*. If the number of such dominating jobs is more than or equal to the number of machines, job *i* cannot be in the optimal solution. c

At the beginning of Phase 1, all jobs that satisfy the condition of Theorem 4.2.3 are eliminated from being candidates of the first jobs. Theorems 4.2.1 and 4.2.2 are

implemented as follows: At a final node of Phase 1 where first jobs on all machines are determined, the following routine is executed to determine if this partial schedule may lead to an optimal solution or not. The algorithm solves the maximum number problem with the set of first jobs F, and $QR = \bigcup_{i \in F} (R_i \cup Q_i)$.

Algorithm 4.2.1:

S0. Determine Q_i and R_i for all $i \in F$. Let $QR = \bigcup_{i \in F} (R_i \cup Q_i)$.

Set $r_i = 0$ for all $i \in F$ to block processing during $[0, r_i]$.

S1. Index the jobs in $QR \cup F$ in chronological order of their ready times.

S2. Consider the jobs in $QR \cup F$ sequentially. At each ready time, add the arriving job to the schedule (Note that we schedule the jobs in set F immediately by setting their ready times to zero). If no machine is available at the ready time of a job, then stop. In such a case, there is an unsequenced job $j \in Q_i$ or R_i for any $i \in F$, which satisfies the condition of Theorem 4.2.1 or Theorem 4.2.2. Hence, F cannot be optimal since not all jobs in $QR \cup F$ are in the solution.

The following is an example illustrating the reduction through Algorithm 4.2.1. Let m=2 and S=8. The ready times, processing times and weights are given as follows:

j	1	2	3	4	5
r_j	1	3	2	4	2
p_j	3	2	1	2	5
w_j	3	5	4	1	6

For F ={1,3}, $Q_1 = Q_3 = \varnothing$, R_1 = {2,5} and R_3 = {5}. Algorithm 4.2.1 for $QR \cup F$ yields A={1,2,3}. Since not all jobs in $QR \cup F$ appear in the solution, F cannot lead to an optimal solution.

Theorems 4.2.1 and 4.2.2 can be implemented more efficiently when there are only two machines. In this case, we know that $|R_i \cup Q_i| \leq 1$ for the two first jobs in set F, due to the earlier elimination by Theorem 4.2.3. Then, to implement the theorems we only need to check for $i \in F$, whether the only job in set $R_i \cup Q_i$ (if not empty)

can be scheduled feasibly on the other machine without violating its spread time constraint. Hence, we get a stronger condition for the two-machine case due to the inclusion of spread time limit, and possibly eliminate more nodes.

For each node, not eliminated by the above conditions, we calculate upper bounds through three procedures. We start with the simplest upper bound UBS_1, and then we calculate UBS_2 for the nodes that cannot be eliminated by UBS_1. If the node cannot be eliminated by UBS_2, we finally calculate UBS_3. All three procedures take the first jobs and calculate the upper bounds accordingly, as explained below.

We hereafter let A_k denote the set of jobs that can be scheduled on machine k without violating the spread time constraint, i.e.:

$A_k = \{ i \mid r_i \geq d_{j_k}$ and $d_i \leq r_{j_k} + S, i \notin F \}$ where $j_k \in F$ is the first job on machine k.

Note that an optimal solution for any relaxation of the constraints provides an upper bound. In Section 4.1, we show that when preemption and partial processing are allowed, the resulting problem is polynomially solvable. This polynomial solution, which can be found by Algorithm 4.1.3, is an upper bound on the optimal weight. We adapt the solution by assuming $\left[d_{j_k} , r_{j_k} + S \right]$ as the interval of availability of machine k. We hereafter refer to this upper bound as UBS_1.

UBS_2 is found by solving m independent Longest Path problems, one for each machine. Since the first job on each machine is already set, the longest path on a machine can be found with a single iteration of Algorithm 4.1.1. Let LP_k be the maximum weight of the jobs on the longest path among the jobs in A_k. UBS_2 may provide an infeasible solution as a job may appear in more than one longest path. Such a case may occur if a particular job appears in more than one A_k set. Clearly, if A_k sets are disjoint, then the resulting solution will be feasible, hence optimal. Theorem 4.2.4 states the validity of UBS_2.

Theorem 4.2.4: UBS_2 is a valid upper bound on the OFJSS problem.

Proof: Assume that W_k is the collection of the weights on machine k in the optimal solution. Note that $W_k \leq LP_k$ as LP_k is the maximum weight that can be assigned to machine k feasibly. Hence, total weight $\sum_k W_k \leq \sum_k LP_k$ and UBS_2 is a valid upper bound. c

Corollary 4.2.1: UBS_2 runs in $O(mn^2)$ time.

Proof: A Longest Path problem is solved in $O(n^2)$ time for each machine. UBS_2 considers all machines, hence runs in $O(mn^2)$ time. c

UBS_3 solves MCNF problem, taking the first job set F, and considering the jobs in $A = \bigcup_k A_k$. The resulting network problem for the jobs in $A \cup F$ can be stated as follows:

- Create a source node s, nodes 1, 2,…, $|A \cup F|$ for each job, and a dummy node $t=|A \cup F|+1$.
- Connect node s to each node $j_k \in F$ with arc cost zero and capacity 1.
- Connect node j to $j+1$ with arc cost zero and capacity m, $j=1,\ldots, |A \cup F|$, unless $j \in F$.
- Create arc (j, k), where k is the first job not overlapping with job j and $k \notin F$, $j=1,\ldots, |A \cup F|$. If no such job exists, create arc (j, t). Each of these arcs has cost $-w_j$ and capacity 1.
- Require a flow of m from s to t.

An example network representation for 7 jobs and 2 machines, where $F=\{4,2\}$ and $A=\{1,3,5,6,7\}$, is given in Figure 4.3. Note that the source node s is connected only to the jobs in set F.

UBS_3 results in a feasible solution once all jobs that are scheduled on machine k are in set A_k. Otherwise, the resulting solution is infeasible, however provides an upper bound on the optimal solution as stated in the following theorem.

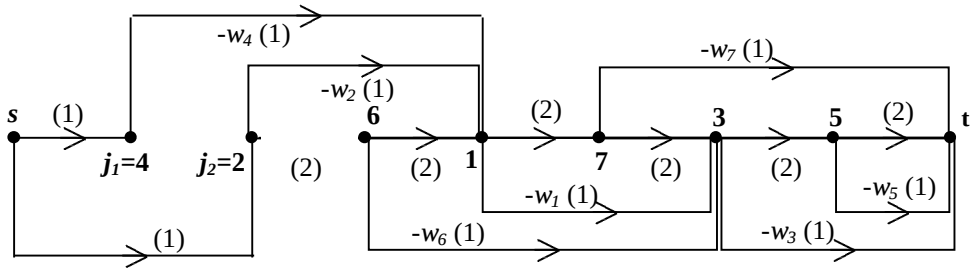


Figure 4.3. Example network representation for UBS_3

Theorem 4.2.5: UBS_3 is a valid upper bound on the OFJSS problem.

Proof: Note that, the jobs that do not appear in set A cannot appear in any optimal solution that satisfies the spread time constraints. The optimal way of scheduling the jobs is found by the MCNF algorithm, as this algorithm solves the problem optimally in the absence of spread time constraints. c

The complexity of the MCNF problem, i.e. $O(n^3)$, defines the complexity of UBS_3.

We now discuss the procedures developed to find initial feasible solutions to our problem, which is done in Phase 1. The following procedure formally states our initial lower bounding procedure, which is used at node zero of the branch and bound tree.

Algorithm 4.2.3:

S1. Solve a MCNF problem with all available n jobs for the OFJS problem that results when the spread time constraint is completely ignored.

Let Z_k be the resulting optimal set of jobs scheduled on machine k.

S2. For each machine k =1,…,m, do:

For each job $j \in Z_k$ do:

Set job j as the first job on machine k. Set $T = r_j + S$.

Determine $Y_{jk} = \{i \in Z_k : r_i \geq r_j, d_i \leq T\}$. Compute $W_{jk} = \sum_{i \in Y_{jk}} w_i$.

Set $W_k = \underset{j}{Max}\{W_{jk}\}$.

Set $LBS_1 = \sum_{k=1}^{m} W_k$.

Corollary 4.2.2: Algorithm 4.2.3 runs in $O(n^3)$ time.

Proof: The complexity of the MCNF problem identifies the complexity of the first step as $O(n^3)$. In S2, there are $O(m)$ machines, and $O(n)$ jobs are to be checked for each machine, leading to a time complexity of $O(nm)$ steps. The overall complexity of the algorithm is $O(n^3)$. c

We utilize a second lower bounding procedure at the leaf nodes of the Phase 1 of the branch and bound tree. This lower bound is based on decomposition idea, and solves m longest path problems in a sequential manner one for each machine, starting from machine 1. Once the longest path is determined on a machine, the jobs in the longest path are eliminated from further consideration. Algorithm 4.2.4 states the procedure formally.

Algorithm 4.2.4:

S1. Set P=∅. Note that, $A_k = \{i \mid r_i \geq d_{j_k}$ and $d_i \leq r_{j_k} + S, i \notin F\}$ where $j_k \in F$ is the first job on machine k.

S2. For each machine k =1,…,m :

Set $A_k = A_k \setminus$ P.

Solve a longest path problem for the jobs in $\{A_k \cup j_k\}$. Let the resulting objective function value be Z_k, and the jobs in the longest path be LP_k.

Set P = P $\cup$ LP_k.

Set $LBS_2 = \sum_{k=1}^{m} Z_k$.

Corollary 4.2.3: Algorithm 4.2.4 runs in $O(mn^2)$ time.

Proof: A Longest Path problem is solved in $O(n^2)$ time for each machine. Algorithm 4.2.4 considers all machines, hence runs in $O(mn^2)$ time. c

We illustrate the execution of Phase 1 through the following example.

Example 3: Let m=2, S=6. Job data are as follows:

j	1	2	3	4	5
r_j	7	3	2	5	2
d_j	10	7	5	6	7
w_j	3	4	4	1	5

At the beginning of Phase 1, we try to reduce the problem size by Theorem 3.2.2. For this instance, the condition of the theorem does not hold. Initial lower bounding procedure that uses the MCNF solution at the root node yields a total weight of 10, where jobs 3 and 4 are scheduled on machine 1 and job 5 on machine 2.

Note that there are a total of 10 possible first job combinations (child nodes of Phase 1) for the above 5-job and 2-machine instance. Below, all combinations are listed with reduction information, and corresponding upper bound values.

First Job Set	*Reduction (Theorem #)*	*Upper Bound*
{3, 5}	-	10
{3, 2}	-	9
{3, 4}	-	8
{3, 1}	Theorem 4.2.3 (job 1)	-
{5, 2}	-	9
{5, 4}	Theorem 4.2.2 (job 4)	-
{5, 1}	Theorem 4.2.3 (job 1)	-
{2, 4}	Theorem 4.2.2 (job 4)	-
{2, 1}	Theorem 4.2.3 (job 1)	-
{4, 1}	Theorem 4.2.3 (job 1)	-

Note that, six of the nodes are eliminated. The nodes having job 1 as one of the first jobs are immediately eliminated due to Theorem 4.2.3, since $|Q_1| = 4 > m = 2$. Other

eliminations are due to Theorem 4.2.2 for job 4: $Q_4 = \{3\}$, and job 3 overlaps with jobs 5 and 2. Hence, Algorithm 4.2.1 cannot schedule job 3 when one of these overlapping jobs is a first job.

Upper bounds are computed for the remaining nodes using UBS_2. Since no upper bound is greater than the initial lower bound, a lower bound is not calculated for any of the leaf nodes. The optimal solution is obtained as the initial feasible solution with an objective function value of 10. Hence, the problem is optimally solved in Phase 1.

Phase 2 takes the partial solutions in the order of the candidate list, and computes an exact solution for the OFJSS problem by implicit enumeration. A node at level l corresponds to a partial solution where the decisions about the first $l + m$ jobs are set, m of which are the first jobs coming from Phase 1. For each node emanating from level l, there are $O(m+1)$ decisions: one for rejecting job $l+1$ (assigning job $l+1$ to machine zero) and one for processing job $l+1$ on machine k, $k=1,\ldots,m$. The assignment to machine k is feasible if the machine does not process any job that overlaps with job $l+1$ and such an assignment does not lead to a violation of spread time constraint.

The branch and bound algorithm in Phase 2 uses a depth-first strategy that introduces one job at a time to the partial schedule. The strategy has relatively low memory requirements and complete solutions are likely to be reached earlier, which enables us to get high quality lower bounds earlier, thereby increasing the power of fathoming. We generate all subproblems of a node and select the one having the maximum upper bound for branching. We remove a job j from consideration if it cannot be feasibly assigned to any machine k such that $j \in A_k$. We remove a machine k from consideration if no job $j \in A_k$ can be assigned to it without overlapping.

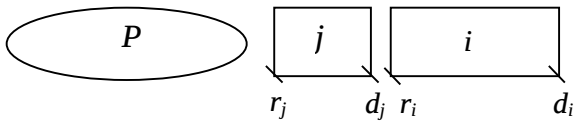
During the implementation of Phase 2, we fathom the node representing the rejection of job j if there exists a scheduled job i such that $w_j > w_i$, $r_i \leq r_j$ and $d_i \geq d_j$ (otherwise Theorem 3.2.1 is violated).

We employ Theorem 3.2.3 as follows: At a node, say δ, of the branch and bound tree where the decision for job i is given, we fathom the node if $\sum_{j \in LP_i, j \in d} w_j \geq w_i$ (otherwise Theorem 3.2.3 is violated).

We let S_k be the start time of j_k, the first job on machine k. Therefore, $S_k + S$ is the upper bound on the completion time of the last job on machine k.

Theorem 4.2.6: If job i is processed on machine k, and there exists a job j such that $S_k \leq r_j \leq r_i$, $d_j \leq r_i$, and no job is processed during $[r_j, d_j]$ on machine k, then job j is also processed in any optimal solution.

Proof: The situation is depicted in the following figure:



Consider an optimal solution that processes job i just after schedule P, but not job j. Assume we add job j to the machine that processes job i. Such an insertion is feasible as the machine is idle during $[r_j, d_j]$. The objective function improves by $w_j > 0$ units. Hence, a solution that does not process job j cannot be optimal. c

If an unscheduled job can be inserted into the partial schedule without causing any overlaps or violation of the spread time constraint, then the node that represents the assignment of the job to machine zero is fathomed (otherwise Theorem 4.2.6 is violated).

Theorem 4.2.7: If job j overlaps with less than l jobs, and $S_k \leq r_j \leq d_j \leq S_k + S$ for l machines, then job j is processed in an optimal schedule.

Proof: According to the conditions of the theorem, at least one machine is idle in the interval $[r_j, d_j]$. When we assign job j to one of these idle machines, the objective function will improve by $w_j > 0$ units. Hence, the solution that excludes job j cannot be optimal. c

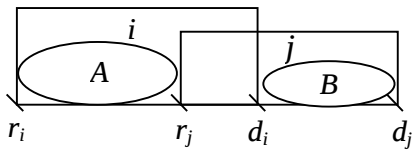
For employing Theorem 4.2.7, at a node of the tree, if there exists an unscheduled job j such that $S_k \le r_j \le d_j \le S_k + S$ for l machines, and job j overlaps with less than l jobs, then we fathom that node (otherwise Theorem 4.2.7 is violated).

Theorem 4.2.8: If $S_k < r_i < r_j < d_i < d_j < S_k + S$, the followings are true:

i. If $w_i > w_j$ and no job is available in the interval $[r_i, r_j]$, then any optimal solution that includes job j should also include job i.

ii. If $w_j > w_i$ and no job is available in the interval $[d_i, d_j]$, then any optimal solution that includes job i should also include job j.

Proof: The situation is illustrated in the following figure:



i. Consider an optimal solution that includes job j, but not job i, and set A is empty. Assume we remove job j, and assign job i to the machine that is freed by removing job j. The objective function improves by $w_i - w_j > 0$ units after this interchange. Hence, a solution that includes job j, but not job i, cannot be optimal.

ii. The proof follows from part (i), and is therefore omitted. c

When branching for assigning part j to machine k at a node of the branch and bound tree, if there exists an unscheduled job i (whose ready time is greater than the deadline of the last job scheduled on machine k) such that $r_j < d_i < d_j$ and $w_i > w_j$, and there exist no scheduled job l on any machine such that $r_i \le r_l \le d_l \le r_j$, then job j cannot be assigned to any machine (otherwise Theorem 4.2.8 is violated).

We calculate an upper bound, for each node that cannot be eliminated by the above conditions. We use UBS_1 adapted for partial schedules in our upper bound

computations in Phase 2. The following algorithm formally states the upper bounding procedure in Phase 2.

Algorithm 4.2.2:

S1. For a given partial schedule *P*, set $UB = \sum_{j \in P} w_j$.

Let $t = \underset{j \in F}{Min}\{d_j\}$ and $T = \underset{j \in F}{Max}\{r_j + S\}$, where *F* is the set of first jobs in the partial schedule.

Let $A_T = \{i : r_i \geq t, d_i \leq T, i \notin P\}$. Set r_{j_k} = 0 for all $j_k \in F$ to block processing during [0, r_{j_k}], where $j_k \in F$ is the first job on machine *k*.

S2. Index jobs in A_T in nonincreasing order of their w_j / p_j values.

S3. Let $\{t_1, t_2, ...\}$ be the sorted sequence of the r_js and d_js of the jobs in A_T in chronological order with duplicates removed,

Let J_a be the set of jobs in A_T available during $[t_a, t_{a+1})$ for a=1, 2,...

Let K_a be the set of machines available during $[t_a, t_{a+1})$ for a=1, 2,...

S4. While satisfying the spread time constraint for k =1,…,m,

For each interval $[t_a, t_{a+1})$ for a=1, 2,... do:

Assign first Min{$|J_a|$, $|K_a|$} smallest-indexed jobs in J_a to the available machines. Let B_a be the set of assigned jobs in interval $[t_a, t_{a+1})$.

Update upper bound as:

$$UB = UB + \sum_{j \in B_a} w_j \left(\frac{t_{a+1} - t_a}{p_j} \right).$$

If an upper bound yields a feasible solution, then we fathom the node and update the current best lower bound. We discard the node if its upper bound is no greater than the best-known lower bound. We update the best-known lower bound when a complete solution with a better objective function value is found.

The computational experience of our optimization procedure for the OFJSS problem will be presented in Chapter 6.

CHAPTER 5

THE OFJS PROBLEM WITH GENERAL WEIGHTS

Recall that, there are machine dependent job weights in the most general version of the OFJS problem (the OFJSG problem). The OFJS problem with eligibility constraints (the OFJSE problem) is a special case of the OFJSG problem. To assure the satisfaction of eligibility constraints for the operational version of the problem, we redefine the machine dependent job weights as follows:

$$w_{jk} = \begin{cases} w_j, & \text{if machine } k \text{ is eligible to process job } j \\ 0, & \text{otherwise.} \end{cases}$$

Machine eligibility constraints are present in the environments where each machine can perform a subset of jobs due to the technological restrictions. We present this special case in Section 5.1, together with related literature and some polynomially solvable cases. In Section 5.2, we present the OFJSG problem with unit processing times as another special case. An optimization procedure is presented for the OFJSG problem in Section 5.3.

5.1. THE FJS PROBLEM WITH ELIGIBILITY CONSTRAINTS

When each machine is eligible to process only a subset of jobs in FJS, several job and machine classes may be formed based on similarities. For each combination of a job class and a machine class, the feasibility of assigning a job to a machine is established in advance, by either using zero/one matrices or some set definitions.

In addition to the parameters defined in Section 2.1, class parameters for jobs and machines are defined as follows:

- e_j: the job class of job j $\quad j=1,\ldots,n$
- f_k: the machine class of machine k $\quad k=1,\ldots,m.$

The number of different job classes and machine classes are denoted by E and F, respectively. For each combination of a job and a machine class, an (E x F) zero/one matrix, L, represents the feasibility of assigning a job in a job class to a machine in a specific machine class. Rows of matrix L denote job classes, while columns denote machine classes. L_{ef} takes the value of one if and only if it is allowed to assign jobs in job class e to machines in machine class f. An example L matrix, L_1 is given below:

$$L_1 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}.$$

Note that L_1 represents a special case with two machine classes having a hierarchical eligibility structure. Machine class 2 can process only the jobs in job class 2, while machine class 1 can process all jobs. We may illustrate this eligibility structure schematically by the following figure. In the figure, while any machine can process the jobs in job class 2, job class 1 can only be processed by a subset of the machines, namely machine class 1.

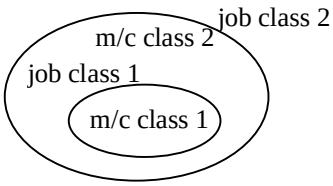


Figure 5.1. The hierarchical eligibility structure imposed by L_1

With this matrix definition, the additional condition L_{ef} =1 should be added to constraint sets (1), (2), and (3) to complete the formulation of the FJS problem with eligibility constraints. The assignment variable x_{jk} will assume zero value if the corresponding matrix entry is zero.

5.1.1. Related Literature

There are a number of studies in literature on FJS with eligibility constraints. The problem takes different names based on the inherent eligibility structure. Each of these problems will be discussed separately.

The TFJS problem with eligibility constraints

The TFJS problem with identical machine classes and one job class is considered by Hashimoto and Stevens (1971), Gertsbakh and Stern (1978), and Gupta *et al.* (1979). This problem is equivalent to the basic TFJS problem, which deals with one job and machine class, and can be solved in $O(n \log n)$ time. Arkin and Silverberg (1987) make an alternative representation of eligibility by letting a V_j to represent the subset of machines by which job j can be processed. Note that, the set V_j corresponds to the set of machines belonging to those machine classes with matrix entries of 1. The authors show that the corresponding feasibility problem that aims to find a feasible schedule that processes all jobs is NP-complete.

Similar problems were studied by several authors. Dondeti and Emmons (1992) prove that the TFJS problem with minimization of cost objective with L matrices L_1 and L_2 (L_2 given below) can be solved in polynomial time by repeatedly solving a Maximum Network Flow Problem, changing upper bounds on arcs iteratively.

$$L_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix}.$$

Figure 5.2 schematically illustrates the eligibility structure imposed by L_2. The machines in machine class i can process the jobs in job classes i and 3.

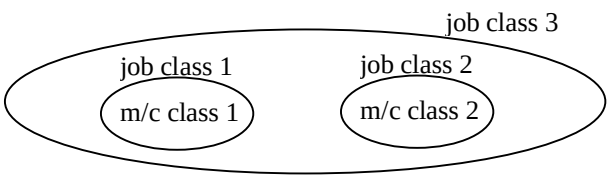


Figure 5.2. The hierarchical eligibility structure imposed by L_2

Dondeti and Emmons (1992) also show that, for these special matrices, the tactical version of the problem that finds the required number of machines in each machine class can be solved in polynomial time.

Kolen and Kroon (1992) study the TFJS problem that minimizes the total cost of machines with the five specific L matrices, including L_1 and L_2. They show that the problem with L_2 can be solved in polynomial time by a combination of Linear Programming and Network Flow algorithms. They prove that the feasibility problem is NP-complete if L has at least 3 columns, i.e. when the number of machine classes (F) is greater than 2. They also show that the optimality problem with $F > 2$ machine classes is NP-hard.

Dondeti and Emmons (1993) refer to the TFJS problem with eligibility and minimization of cost objective as the Exclusive Class Scheduling (ECS) problem. They study the preemptive Exclusive Class Scheduling (ECS) problem. They formulate the preemptive feasibility problem as a set of transportation problems that are solved in polynomial time. However, they conjecture that the preemptive optimization problem is NP-hard.

Kroon *et al.* (1997) present a complete classification for the TFJS problem with eligibility constraints in terms of complexity. When the number of machine classes, F, is variable, the preemptive and nonpreemptive versions of the problem become NP-hard in the strong sense. On the other hand, when F is fixed and smaller than 3, both versions are polynomially solvable. When F is fixed and greater than or equal to 3, only the nonpreemptive version becomes NP-hard. The authors develop some lower and upper bounds for the problem, incorporate these into Branch and Bound algorithm and Lagrangean Relaxation based procedures. Their computations show that the approaches are effective for medium-sized problems.

Jansen (1994) provides an approximation algorithm for the TFJS problem with eligibility and availability constraints, with a specific 3x3 square L matrix, where machine class 1 can process all jobs, machine class 2 can process job classes 1 and

2, and machine class 3 can process job classes 1 and 3. His algorithm has a complexity of $O(F.Z.n^2)$, where Z is the number of shifts. A genetic algorithm using a reinforcement learning system is developed by Santos and Zhong (2001) for the TFJS problem with eligibility constraints and minimum total cost objective. Their computational experiments reveal that their approach is robust in terms of problem structure, and promising with respect to catching a fast local minimum.

The OFJS problem with eligibility constraints

Arkin and Silverberg (1987) show that the feasibility problem for the OFJSE problem is NP-complete, when $F > 2$. They also present an $O(n^{m+1})$ time Dynamic Programming based algorithm for the OFJSE problem. The existence of the algorithm implies that, the OFJS problem can be solved in polynomial time if the number of machines is fixed. Kolen and Kroon (1991) study five specific L matrices for the OFJSE problem with different eligibility structures and at most 3 job and machine classes. They prove that the OFJSE problem with at least 2 machine classes ($F > 1$) is NP-hard in the strong sense, through transformation from the Numerical Three Dimensional Matching Problem, which is known to be NP-hard in the strong sense (Garey and Johnson, 1979). They also show that the associated feasibility problem is NP-complete if and only if L has at least 3 columns.

Kroon *et al.* (1995) provide a Lagrangean relaxation based approximation algorithm for the OFJSE problem. Their algorithm is based on the observation that the OFJS problem can be modeled as a MCNF problem when all machines are identical. They relax the restriction that each job should be processed at most once. For each machine class, they solve a MCNF problem having some side constraints. Their computational results based on hypothetical and real data reveal that their algorithm is satisfactory for practical purposes.

The Hierarchical Class Scheduling (HCS) problem

The Hierarchical Class Scheduling (HCS) problem is defined as a problem with a upper triangular (or lower triangular) square L matrix, which implies that a machine

in class f can process jobs in classes e such that $e \leq f$ (or $e \geq f$). Note that the matrix L_1 represents such an eligibility structure. Kroon *et al.* (1997) show that the feasibility problem for the tactical HCS problem is NP-complete when $F \geq 4$. Dondeti and Emmons (1992) show that the tactical HCS problem can be solved in polynomial time if $F \leq 2$. Dondeti and Emmons (1993) propose a greedy algorithm for the optimal preemptive schedule for the tactical HCS problem with $F > 2$ and with cost minimization objective. They also show that the feasibility problem associated with the nonpreemptive version is NP-complete.

For the operational version of the problem, Kolen and Kroon (1991) prove that the problem is NP-hard in the strong sense, when $F > 1$. They also show that the associated feasibility problem is NP-complete when $F \geq 3$. Bouzina and Emmons (1995) study the operational HCS problem with L_1, and conjecture that the problem of maximizing the number of jobs is NP-hard. They provide a polynomial time algorithm for the preemptive version of this problem, and a polynomial time algorithm for the case having additional machine availability constraints.

The One-or-all Class Scheduling (OCS) problem

The One-or-all Class Scheduling (OCS) problem is defined as follows: There are E machine classes and E+1 job classes, and a job in job class e can only be processed by a machine of class e, e=1,2,…,E, and jobs in class *zero* can be processed by all machines. Note that the matrix L_2 represents such an eligibility structure. Dondeti and Emmons (1993) prove that the feasibility problem for the tactical OCS problem is NP-complete. The authors provide an algorithm for the preemptive tactical OCS problem with the minimization of total cost objective. The algorithm provides an optimal preemptive solution, where only class-zero jobs are preempted. They also show that the feasibility problem associated with the non-preemptive version is NP-complete.

Table 5.1 combines the complexity results in literature on the FJS problem with eligibility constraints. Some polynomially solvable special cases of the OFJSE problem are presented in Section 5.1.2.

Table 5.1. Results for the FJS problem with eligibility constraints

Objective Functions	
Minimize Total Cost (Minimize number of machines)	**Maximize Total Weight (Maximize number of jobs)**
The feasibility problem is NP-complete. (Arkin & Silverberg, '87)	The feasibility problem is NP-complete for $F > 2$. (Arkin & Silverberg, '87)
$F = 2 \rightarrow$ Polynomially solvable. (Dondeti, Emmons '92) $F > 2 \rightarrow$ NP-hard. (Kolen, Kroon '92)	$F > 1 \rightarrow$ NP-hard. (Kolen, Kroon '91)
Preemptive version: Feasibility problem is polynomially solvable. (Dondeti, Emmons '93) Optimality problem: F fixed $\rightarrow$ polynomially solvable. F not fixed $\rightarrow$ Strongly NP-hard. (Dondeti, Emmons '93)	
HCS: Feasibility problem is NP-complete for $F \geq 3$. (Kroon et. al. '97) $F = 2 \rightarrow$ Polynomially solvable. (Dondeti, Emmons '92)	**HCS:** HCS with $F = 2$ is NP-hard. (Kolen, Kroon '91) Feasibility problem with $F \geq 3$ is NP-complete. (Kroon et. al. '97)
Preemptive HCS: Polynomially solvable. (Dondeti, Emmons '93)	**Preemptive HCS:** $F = 2 \rightarrow$ Polynomially solvable (Bouzina, Emmons '95)
OCS: Feasibility problem is NP-complete. Preemptive OCS is polynomially solvable. (Dondeti, Emmons '93)	

5.1.2. Special Cases of the OFJSE Problem

In this section, we examine some variations of the OFJS problem with eligibility constraints that can be solved in polynomial time. For these cases, we provide polynomial time algorithms, and settle their time complexities.

We classify the special cases according to the weight structures as machine specific, job specific, or job and machine specific. Job weights are further categorized as weight per job, or weight per unit processing time of the job. We analyze these cases below.

Special Cases with Machine Specific Weights

When weights are defined on machines, any job that is processed on machine type f brings a profit of w_f. Machine dependent weights may apply to the cases where the profits brought by the resources vary, but not according to the customer served. One example case may be encountered in assigning holiday bungalows to vacationers (Kroon et. al., 1995). Usually, holiday bungalows are booked a long time in advance for a period of one or more weeks. The holiday bungalows may differ in several aspects like size, location, accommodation, quality, as well as price (w_f). Each season, the booking office faces a problem of finding an assignment of holiday bungalows to vacationers to maximize profit, such that there is a matching between the desires of the vacationers and available accommodation. Below is a special case with machine specific weights, and two types of machines.

Special Case 1: The preemptive HIS Problem with E=2, F=2.

There are two types of jobs and two types of machines, where one machine type is more capable than the other. Without loss of generality we may assume that machines of type 2 (small machines) can only process job type 2, where machines of type 1 (large machines) can process both type 1 and type 2 jobs. Recall that the matrix L_1 and Figure 5.1 represents this problem.

Any job that is processed on machine type 1 brings a profit of w_1, and similarly w_2 for machine type 2. More formally, we can define weights consistent with our problem formulation. Let us remember our weight definition:

w_{jk}: weight of job j if it is processed on machine k j=1,…,n k=1,...,m.

Also recall the following definitions:

- e_j: the job class of job j j=1,…,n

- f_k: the machine class of machine k k=1,…,m.

The job weights for this problem become:

$$w_{jk} = \begin{cases} w_1 & \text{if } f_k = 1 \\ w_2 & \text{if } e_j = 2,\ f_k = 2 \\ 0 & \text{if } e_j = 1,\ f_k = 2 \end{cases}$$

Our objective is the maximization of total weight of jobs, i.e. $\sum_k \sum_j w_{jk} p_{jk}$, where p_{jk} is the percentage of job j processed on machine k, $\forall j,k$ (Note that $\sum_k p_{jk} = 0$ or $1, \forall j$).

The solution approach differs based on the relative magnitudes of the machine weights. When $w_1 = w_2$, the problem objective becomes the maximization of the total number of jobs. This case is shown to be polynomially solvable by Bouzina and Emmons (1995). We deal with the remaining cases, in particular, either $w_1 > w_2$, or $w_2 > w_1$. We examine them below:

Case 1.1: $w_1 > w_2$. Machine with greater capability (which can process any type of job) brings more profit per job. In this case, we try to load machine 1 with many jobs as possible, and process the jobs on machine 2 only when machine 1 is not available. Note that, this problem is very similar to the number maximization problem on machine 1. The algorithm below is parallel to that of Bouzina and Emmons' (1995), and solves the problem optimally:

Algorithm 5.1.1:

S1. Index jobs in their nondecreasing order of r_j values.

S2. Consider jobs sequentially:

Case A: Arriving job j is a type 2 job:

a. A machine is free **à** Assign j to that machine, giving priority to type 1 machines

b. All machines are busy **à** Remove the job with latest deadline among all jobs in process (including job *j*)

Case B: Arriving job *j* is a type 1 job:

a. A type 1 machine is free **à** Assign *j* to that machine

b. All type 1 machines are busy with type 1 jobs **à** Remove the job with latest deadline among all jobs in process on type 1 machines (including job *j*)

c. All type 1 machines are busy, a type 2 job is processed on a type 1 machine, and a type 2 machine is free at the time **à** Switch the type 2 job to the idle type 2 machine, assign *j* to the type 1 machine freed by the switched job.

d. All machines are busy, a type 2 job is processed on a type 1 machine **à** Remove the job with latest deadline among all jobs in process (including job *j*), call this job as job *h*.

If job *h* was on a type 1 machine **à** Assign *j* to that machine

If job *h* was on a type 2 machine **à** Switch the type 2 job to that machine, assign *j* to the type 1 machine freed by the switched job.

S3. After the schedule is determined, try to switch the scheduled jobs from the less-weight machine to the more-weight machine whenever possible (whenever type 1 machine is idle and type 2 machine is busy).

S4. Compute p_{jk}, $\forall j,k$. Objective function value, $Z = \sum_k \sum_j w_{jk} p_{jk}$.

Theorem 5.1.1: Algorithm 5.1.1 solves the preemptive HIS Problem with two job and machine classes, where the weights are machine specific and $w_1 > w_2$.

Proof: Note that, when machine type 1 has more weight, the algorithm loads machine type 1 as much as possible, and machines of type 2 are loaded whenever type 1 machines are full. Moreover, no job remains unscheduled if there is space on machines. Hence, the solution generated by Algorithm 5.1.1 is optimal. c

Corollary 5.1.1: Algorithm 5.1.1 runs in $O(nm + n \log n)$ time.

Proof: Complexity of Algorithm 5.1.1 is computed as follows. The sorting operation in S1 takes $O(n \log n)$ time. In S2, for each job, each machine is examined in $O(m)$ time, and finding the job with latest deadline takes $O(\log n)$ time. The time complexity of this step is therefore $O(nm + n \log n)$. Finally, in S3, $O(2n)$ intervals are checked for improvement. The overall complexity of the algorithm hence becomes $O(n \log n + nm + n \log n + 2n)$, equivalently $O(nm + n \log n)$. c

Case 1.2: $w_2 > w_1$. Machine with limited capability (which can process only type 2 jobs) brings more profit per job. In this case, we try to load machine 2 with type 2 jobs as many as possible, and process type 2 jobs on machine 1 only when machine 2 is not available. Note that, the problem resembles the number maximization problem on machine 2. Algorithm 5.1.2 solves this case optimally:

Algorithm 5.1.2: Implement Algorithm 5.1.1, replacing "type 1 machines" with "type 2 machines" only in S2.Case A.a.

Theorem 5.1.2: Algorithm 5.1.2 solves the preemptive HIS Problem with two job and machine classes, where the weights are machine specific and $w_2 > w_1$.

Proof: The proof of Theorem 5.1.2 is similar to the proof of Theorem 5.1.1, and is therefore omitted. c

Corollary 5.1.2: Algorithm 5.1.2 runs in $O(nm + n \log n)$ time.

Proof: The proof of Corollary 5.1.2 is similar to the proof of Corollary 5.1.1, and is therefore omitted. c

Special Cases with Job Specific Weights

Assume the weights are defined on jobs instead of machines, and any job of type e brings a profit of w_e. This problem has many practical applications. For example, it

may be used in scheduling operating rooms in a hospital. In most hospitals, a limited number of operating rooms is available. Some of these rooms may be general purpose, but others may be suitable for only a subset of various types of operations. In general, the time slot of an operation is fixed some time ahead. The problem to be solved in this context is to find a feasible schedule for as many as possible of the planned operations having different priorities, taking into account the restricted suitability of the operating rooms. Special Cases 2 and 3 have job specific weights.

Special Case 2: The preemptive HIS Problem with *E*=2, *F*=2.

The problem is represented by L_1. This time, any type 1 job brings a profit of w_1, and similarly any type 2 job brings w_2. We can define these weights as follows:

$$w_{jk} = \begin{cases} w_1 & \text{if } e_j = 1,\ f_k = 1 \\ w_2 & \text{if } e_j = 2 \\ 0 & \text{if } e_j = 1,\ f_k = 2 \end{cases}$$

Our objective is the maximization of the sum, $\sum_k \sum_j w_{jk} p_{jk}$, where p_{jk} is the percentage of job j processed on machine k, $\forall j,k$. The solution approach differs based on the relative magnitude of weights of the jobs. We examine two possible cases below:

Case 2.1: $w_1 \geq w_2$. Job type 1, i.e. jobs that can be processed on a subset of machines (Refer to Figure 5.1) is more valuable. The algorithm below solves this case:

Algorithm 5.1.3:

S1. Index jobs in nondecreasing order of their r_j values.

S2. Consider jobs sequentially:

Case A: Arriving job j is a type 2 job:

a. A machine is free **à** Assign *j* to that machine, giving priority to type 2 machines

b. All machines are busy **à** Find the job with latest deadline among all jobs in process (including job *j*), call this job as job *h*.

1. If job *h* is a type 2 job **à** Remove job *h* from schedule.
2. If job *h* is a type 1 job **à** Find the job with latest deadline among all type 2 jobs in process (including job *j*), call this job as job *i*.

If no job arrives in $[d_i, d_h)$, remove job *i* from schedule

Else, we should remove either *h* or *i*: To decide, either;

-Solve optimally by enumeration, or

-Solve approximately by bounding.

Case B: Arriving job *j* is a type 1 job:

a. A type 1 machine is free **à** Assign *j* to that machine

b. All type 1 machines are busy with type 1 jobs **à** Remove the job with latest deadline among all jobs in process on type 1 machines (including job *j*)

c. All type 1 machines are busy, a type 2 job is processed on a type 1 machine, and a type 2 machine is free at the time **à** Switch the type 2 job to the idle type 2 machine, assign *j* to the type 1 machine freed by the switched job.

d. All machines are busy, a type 2 job is processed on a type 1 machine **à** Find the job with latest deadline among all jobs in process (including job *j*), call this job as job *h*.

1. If job *h* is a type 2 job **à** Remove job *h* from schedule.
2. If job *h* is a type 1 job **à** Find the job with latest deadline among all type 2 jobs in process, call this job as job *i*.

If no job arrives in $[d_i, d_h)$, remove job *i* from schedule

Else, we should remove either *h* or *i*: To decide, either;

- Solve optimally by enumeration, or

- Solve approximately by bounding.

S3. Compute p_{jk}, $\forall j,k$. Objective function value, $Z = \sum_k \sum_j w_{jk} p_{jk}$.

In steps S2.CaseA.b.2 and S2.CaseB.d.2, the decision as to which of the candidate jobs to remove from schedule can be taken optimally by an enumeration algorithm. Note that, two decisions; removing job *h*, and removing job *i*, should be investigated for each arriving job, thereby leading to $O(2^n)$ alternatives. Enumeration algorithms for making this decision may be a research issue. On the other hand, if we use heuristics to make this decision, the algorithm will yield an upper or lower bound. In particular, if an upper bound is required for this special case, the following adaptation may be used.

Upper Bound: We remove the job with the latest deadline, that is, job *h*. However, we make the assumption that job *i* (which is the job with latest deadline among all type 2 jobs in process) is of type 1 (has higher weight), thereby creating an infeasible solution. The resulting solution is an upper bound, since we are overestimating the weight of job *i*, and remove the job having largest deadline.

If a feasible solution, i.e. a lower bound is required, one of the following adaptations is required for steps S2.CaseA.b.2, and S2.CaseB.d.2:

Lower Bound (1): Remove *h*, i.e. the job with the largest deadline.

Lower Bound (2): Consider the maximal-weight set (*M*) of all nonoverlapping jobs arriving in the interval $[d_j, d_h]$.

If $w_j + \sum_{i \in M} w_i < w_h$, keep *h*.

Else, remove *h*, assign *j* and jobs in *M* to the machine freed by *h*.

Note that feasible solutions are generated as a result; however, the solutions are not guaranteed to be optimal.

Theorem 5.1.3: Algorithm 5.1.3 solves the preemptive HIS Problem with two job and machine classes, where the weights are job specific and $w_1 > w_2$.

Proof: Algorithm 5.1.3 gives priority to the job having largest weight in all steps except steps S2.CaseA.b.2, and S2.CaseB.d.2. Such a priority should be given, unless there is a job that has smaller priority and earlier completion. Preferring low priority job can be justified if there are arrivals between the completion time of the low priority and high priority jobs. Steps S2.CaseA.b.2, and S2.CaseB.d.2 check the existence of such cases. If such a case exists, the decision of selecting a job cannot be based on a simple rule, unlike the other cases, as mentioned earlier. In all other cases, we prefer the job having higher priority. When there are two jobs having same priority, the one that completes earlier has to be selected to give room for future arrivals. c

We do not report the complexity of this case, as we have already shown that there are $O(2^n)$, i.e. exponential number of alternatives, to be searched for reaching an optimum solution.

We illustrate the execution of Algorithm 5.1.3 through Example 4:

Example 4: Assume 13 jobs are to be scheduled on 3 parallel machines. Machines 1.1 and 1.2 are of type 1, and machine 2 is of type 2. Table 5.2 shows the values for job parameters. Jobs of type 1 have weights of 6, while weights of type 2 jobs are 2.

Table 5.2 Job parameter values for Example 4

j	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
r_j	1	1	2	4	5	7	8	10	11	11	14	19	19
d_j	7	5	9	11	11	15	14	18	19	17	20	23	22
w_j	6	2	2	6	6	2	6	2	6	2	2	6	6

Suppose that we are looking for an upper bound for the weight maximization problem. Algorithm 5.1.3 proceeds as follows:

S1. Jobs are already in their nondecreasing order of r_j values.

S2. Job 1 arrives: type 1 job. Machine 1.1 is free. Assign job 1 to machine 1.1.

Job 2 arrives: type 2 job. Machine 2 is free. Assign job 2 to machine 2.

Job 3 arrives: type 2 job. Machine 1.2 is free. Assign job 3 to machine 1.2.

Job 4 arrives: type 1 job. All machines are busy. A type 2 job is on a type 1 machine. Job with latest deadline: Job 4 (type 1) with $d_4 = 11$

Job of type 2 with latest deadline: Job 3 with $d_3 = 9$

Is there an arrival in [9,11)? Yes, job 8 arrives. Compute upper bound:

Do not schedule job 4. Assume that job 3 is of type 1, i.e. $w_3 = 6$.

Job 5 arrives: type 1 job. All type 1 machines are busy with type 1 jobs.

Job of type 1 with latest deadline: Job 5 with $d_5 = 11$. Do not schedule.

Job 6 arrives: type 2 job. Machine 2 is free. Assign job 6 to machine 2.

Job 7 arrives: type 1 job. Machine 1.1 is free. Assign job 7 to machine 1.1.

Job 8 arrives: type 2 job. Machine 1.2 is free. Assign job 8 to machine 1.2.

Job 9 arrives: type 1 job. All machines are busy. A type 2 job is on a type 1 machine. Job with latest deadline: Job 9 (type 1) with $d_9 = 19$

Job of type 2 with latest deadline: Job 8 with $d_8 = 18$

No arrival is possible in [18,19). Remove job 8 from schedule, assign job 9 to machine 1.2 freed by job 8.

Job 10 arrives: type 2 job. All machines are busy.

Job with latest deadline: Job 9 (type 1) with $d_9 = 19$

Job of type 2 with latest deadline: Job 10 with $d_{10} = 17$

Is there an arrival in [17,19)? No. Do not schedule job 10. Keep job 9 in schedule.

Job 11 arrives: type 2 job. Machine 1.1 is free. Assign job 11 to machine 1.1.

Job 12 arrives: type 1 job. Machine 1.2 is free. Assign job 12 to machine 1.2.

Job 13 arrives: type 1 job. All machines are busy. A type 2 job is on a type 1 machine, and machine 2 is free.

Switch job 11 to machine 2. Assign job 13 to machine 1.1 freed by job 11.

Objective function value = $w_1 + w_2 + w_3 + w_6 + w_7 + w_9 + w_{11} + w_{12} + w_{13}$

$= 6 + 2 + 6 + 2 + 6 + 6 + 2 + 6 + 6 = 42.$

Only job 11 is preempted in the schedule. Note that, the weight of job 3 is overestimated while computing the objective function value of 42. Hence, 42 is an upper bound on the objective function value of the original problem. Figure 5.3 depicts the schedule computed by the algorithm.

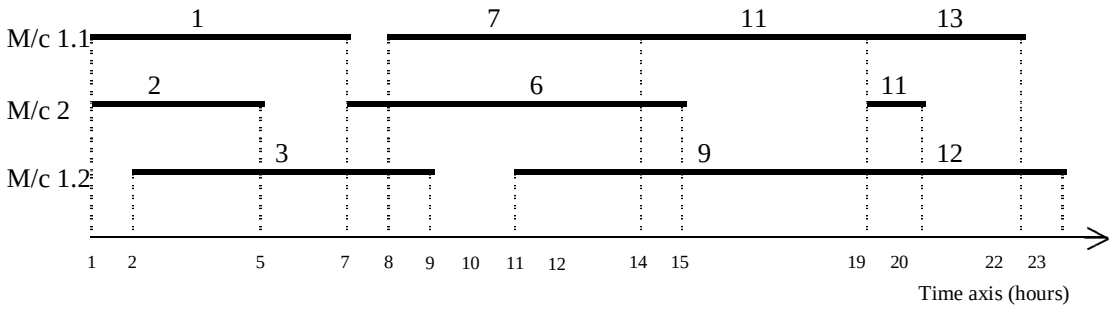


Figure 5.3. Solution of Example 4 by Algorithm 5.1.3

Case 2.2: $w_2 > w_1$**.** Job type 2, which can be processed on any machine (Refer to Figure 5.1) is more valuable. The algorithm below, which is merely an adaptation of Algorithm 5.1.3, solves this case:

Algorithm 5.1.4:

S1. Index jobs in nondecreasing order of their r_j values.

S2. Consider jobs sequentially:

Case A: Arriving job *j* is a type 2 job:

- **a.** A machine is free **à** Assign *j* to that machine, giving priority to type 2 machines
- **b.** All machines are busy **à** Find the job with latest deadline among all jobs in process (including job *j*), call this job as job *h*.
    1. If job *h* is a type 1 job **à** Remove job *h* from schedule

2. If job *h* is a type 2 job **à** Find the job with latest deadline among all type 1 jobs in process, call this job as job *i*.

If no job arrives in $[d_i, d_h)$, remove job *i* from schedule

Else, we should remove either *h* or *i*: To decide, either;

-Solve optimally by enumeration, or

-Solve approximately by bounding.

Case B: Arriving job *j* is a type 1 job:

a. A type 1 machine is free **à** Assign *j* to that machine

b. All type 1 machines are busy with type 1 jobs **à** Remove the job with latest deadline among all jobs in process on type 1 machines (including job *j*)

c. All type 1 machines are busy, a type 2 job is processed on a type 1 machine, and a type 2 machine is free at the time **à** Switch the type 2 job to the idle type 2 machine, assign *j* to the type 1 machine freed by job the switched job

d. All machines are busy, a type 2 job is processed on a type 1 machine **à** Find the job with latest deadline among all jobs in process (including job *j*), call this job as job *h*.

1. If job *h* is a type 1 job **à** Remove job *h* from schedule
2. If job *h* is a type 2 job **à** Find the job with latest deadline among all type 1 jobs in process (including job *j*), call this job as job *i*.

If no job arrives in $[d_i, d_h)$, remove job *i* from schedule

Else, we should remove either *h* or *i*: To decide, either;

- Solve optimally by enumeration, or

- Solve approximately by bounding.

S3. Compute p_{jk}, $\forall j,k$. Objective function value, $Z = \sum_k \sum_j w_{jk} p_{jk}$.

Theorem 5.1.4: Algorithm 5.1.4 solves the preemptive HIS Problem with two job and machine classes, where the weights are job specific and $w_2 > w_1$.

Proof: The proof of Theorem 5.1.4 is similar to the proof of Theorem 5.1.3, and is therefore omitted. c

We do not report the complexity of this case, since there is exponential number of alternatives to be searched, as in Algorithm 5.1.3.

Job specific weights may also be defined per unit time. In this case, any job of type e brings a profit of w_e per unit time of processing. In this case, the objective function of the problem becomes $\sum_k \sum_j w_{jk} p_{jk}$, where p_{jk} is the total time that job j is processed on machine k. Note that, $\sum_k p_{jk} = 0$ or $p_j, \forall j$.

Below is a special case with job specific weights per unit time.

<u>Special Case 3</u>: The HIS Problem, weights are job specific and per unit time, p_j =1, "j.

Recall that this problem may be represented by an upper triangular square L matrix. There are E types of jobs and E types of machines, where a machine type e is capable of processing job types e, e-1,…,1. Job weights are given below:

$$w_{jk} = \begin{cases} w_e & \text{if } f_k \geq e_j, \\ 0 & \text{if } f_k < e_j. \end{cases} \quad \text{where } w_e > w_{e-1} > ... > w_1 \ .$$

In this special case, the objective is the maximization of the total profit, expressed by $\sum_k \sum_j w_{jk} x_{jk}$.

This problem is common in hotel reservation systems, where customers bring profits per night-of-stay. In this special case, when a customer requires a certain type of room (e.g. single) that is not available at his/her arrival, he/she may accept to stay in a superior/more expensive (e.g. double) room, but at the same price (e.g. single room price). However, a customer requiring a superior room cannot be served by a less luxurious one. The following algorithm solves this special case optimally:

Algorithm 5.1.5:

S1. Let $\{t_1, t_2,...\}$ be the sorted sequence of the r_js in chronological order with duplicates removed. Let S_a be the set of jobs available in the interval $[t_a, t_{a+1})$ for a=1, 2,...

S2. For each interval $[t_a, t_a+1)$ for a=1, 2,... do:

Consider jobs in S_a sequentially, beginning with the largest type index.

Let the type index of the current job j be e,

If any machine of type $f \geq e$ is available in the interval, assign j to that machine,

Else, skip job j.

Theorem 5.1.5: Algorithm 5.1.5 solves the HIS problem, where the weights are job specific and per unit time, and p_j =1,$\forall j$.

Proof: The procedure defined in S2 finds the maximum-weight subset of the jobs that are completely processed in the interval $[t_a, t_a + 1)$, and assigns them to available eligible machines. As S_as are disjoint due to unit processing times, the collection of optimal assignment solutions gives the overall optimum solution. c

Corollary 5.1.5: Algorithm 5.1.5 runs in $O(n^2m)$ time.

Proof: The complexity of Algorithm 5.1.5 is computed as follows. In S1, the sorting operation takes $O(n \log n)$ time. In S2, for each interval, $O(n)$ jobs and $O(m)$ machines are examined in $O(nm)$ steps. There are $O(n)$ intervals, therefore S2 is executed in $O(n^2m)$ steps. Hence, the overall complexity of the algorithm is $O(n^2m)$. c

This algorithm can also be generalized for the case with p_j =1, and there is no relation between machine capabilities. In this case, instead of considering the jobs in S_a sequentially in each interval, we should choose the most-weighted subset of jobs in S_a, by solving a linear assignment problem. Special Case 4 below defines this case formally.

5.2. THE OFJSG PROBLEM WITH UNIT PROCESSING TIMES

In this section, we show that when all processing times are unity in the OFJSG problem, then the problem can be solved in $O(n^3)$ time when $m = 2$, and $O(n^4)$ time for arbitrary m.

Algorithm 5.2.1 solves the OFJSG problem optimally, when $p_j = 1, \forall j$

Algorithm 5.2.1:

S1. Let $\{t_1, t_2,...\}$ be the sorted sequence of the r_js in chronological order with duplicates removed. Let S_a be the set of jobs available in the interval $[t_a, t_a + 1)$ for a=1, 2,...

S2. For each interval $[t_a, t_a + 1)$ for a=1, 2,... do:

Choose the most-weighted subset of Min$\{|S_a|, m\}$ jobs among the jobs in S_a by solving the following assignment problem:

$$\text{Maximize} \quad \sum_k \sum_j w_{jk} x_{jk}$$

$$\text{s.t.} \quad \sum_j x_{jk} \le 1 \qquad \forall k$$

$$\sum_k x_{jk} \le 1 \qquad \forall j \in S_a$$

$$x_{jk} \in \{0,1\} \qquad \forall j,k$$

Theorem 5.2.1: Algorithm 5.2.1 solves the OFJSE problem, where the weights are job and machine specific and per unit time, and $p_j = 1, \forall j$.

Proof: The assignment problem defined in S2 finds the maximum-weight subset of the jobs that are completely processed in the interval $[t_a, t_a + 1)$. As S_as are disjoint due to unit processing times, the collection of optimal assignment solutions gives the overall optimum solution. c

Corollary 5.2.1: Algorithm 5.2.1 runs in $O(n^4)$ time.

Proof: The complexity of Algorithm 5.2.1 is computed as follows. In S1, the sorting operation takes $O(n \log n)$ time. In S2, for each of the $O(n)$ intervals, an assignment algorithm is solved for $O(n)$ jobs in $O(n^3)$ steps. Hence, the overall complexity of the algorithm is $O(n^4)$. c

When m=2, i.e. there are two machines, a more efficient procedure than an assignment problem may be used. The following algorithm solves the OFJSG problem, when $m = 2$, and $p_j = 1, \forall j$.

Algorithm 5.2.2:

S1. Let $\{t_1, t_2,...\}$ be the sorted sequence of the r_js in chronological order with duplicates removed. Let S_a be the set of jobs available in the interval $[t_a, t_a + 1)$ for a=1, 2,...

S2. For each interval $[t_a, t_a + 1)$ for a=1, 2,... do:

Choose the most-weighted two jobs among jobs in S_a, i.e. choose jobs r and s such that;

$$w_{r1} + w_{s2} = \underset{i,j}{Max}\left\{Max\left\{w_{i1} + w_{j2}, w_{j1} + w_{i2}\right\}\right\}.$$

Assign jobs r and s to machines 1 and 2, respectively.

Theorem 5.2.2: Algorithm 5.2.2 solves the OFJSE problem, where the weights are job and machine specific and per unit time, $p_j = 1, \forall j$ and F=m=2.

Proof: Note that, for each interval, at most two jobs yielding maximum total weight should be selected. The job pair (r, s) that maximizes total weight is clearly computed as:

$$w_{r1} + w_{s2} = \underset{i,j}{Max}\left\{Max\left\{w_{i1} + w_{j2}, w_{j1} + w_{i2}\right\}\right\}.$$ c

Corollary 5.2.2: Algorithm 5.2.2 runs in $O(n^3)$ time.

Proof: The complexity of Algorithm 5.2.2 is computed as follows. In S1, the sorting operation takes $O(n \log n)$ time. In S2, for each of the

$O(n)$ intervals, the most-weighted two jobs are selected in $O(n^2)$ time, as $O(n^2)$ pairs are evaluated. Hence, the overall complexity of the algorithm is $O(n^3)$. c

5.3. OPTIMIZATION PROCEDURE

Recall that, the OFJSE problem with at least 2 machine classes ($F > 1$), which is a special case of the OFJSG problem, is NP-hard in the strong sense. This suggests the use of implicit enumeration techniques for finding optimal solutions. Hence, we propose a branch and bound algorithm to find the optimal assignments for the OFJSG problem with arbitrary w_{jk} values.

In our Branch and Bound (BB) procedure, we assume that the jobs are indexed in nondecreasing order of their ready times. A node at level l corresponds to a partial solution where the decisions about the first l jobs, having the smallest ready times, are set. For each node emanating from level l, there are $O(m+1)$ decisions: one for rejecting job $l+1$ and one for processing job $l+1$ on machine k, $k=1,\ldots,m$. The assignment to machine k is feasible if the machine does not process any job that overlaps with job $l+1$. We illustrate our BB tree in Figure 5.4.

In our branch and bound algorithm, we employ some properties that help to characterize the structure of an optimal solution of the OFJSE problem, and use these properties to reduce the problem size. Theorem 5.2.1 states one such property.

Theorem 5.3.1: If $r_i \leq r_j$, $d_i \geq d_j$, and $\exists k \ni w_{jk} > w_{ik}$, then any optimal solution that includes job i on machine k should also include job j.

Proof: Assume that there is an optimal solution that includes job i on machine k, but not job j. Assume we remove job i, and assign job j instead. Note that, such an interchange can be done feasibly without alternating any processing as $r_i \leq r_j$ and $d_i \geq d_j$. The objective function improves by $w_{jk} - w_{ik} > 0$ units. Hence, a

solution that includes job *i* on machine *k*, but not job *j*, cannot be optimal. c

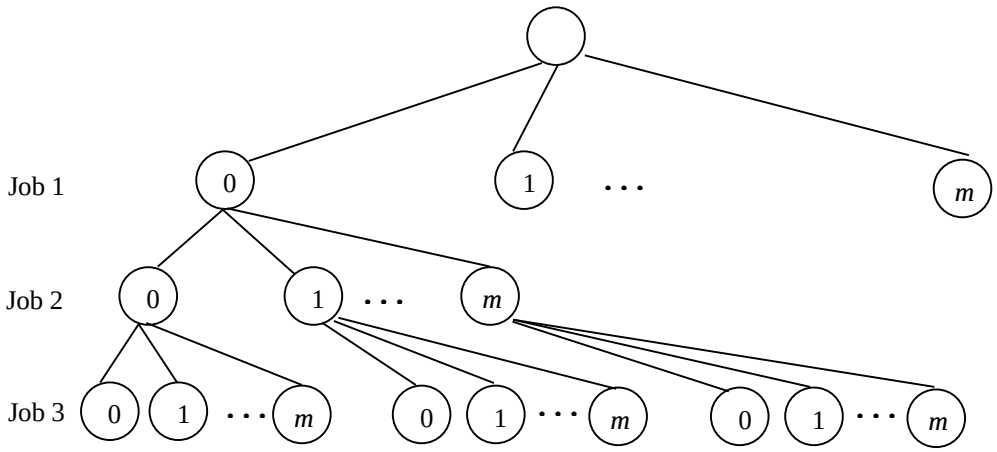


Figure 5.4. Branch and Bound tree for the OFJSG problem.

During the implementation of the branch and bound procedure, we fathom the node representing the rejection of job *j* if there exists a scheduled job *i* on machine *k*, and job *j* dominates job *i* on machine *k* (otherwise Theorem 5.3.1 is violated).

Theorem 5.3.2 establishes another property of an optimal solution, which is a direct implication of Theorem 5.3.1. We hereafter say that job *j* dominates job *i* on machine *k*, if $w_{jk} > w_{ik}$, $r_i \le r_j$ and $d_i \ge d_j$. Let S_{ik} be the set of jobs that dominate job *i* on machine *k*.

Theorem 5.3.2: If $|S_{ik}| \ge m$, $\forall k$, then job *i* is not processed in any optimal solution.

Proof: We have already shown that, if job *j* dominates job *i* on a machine, say machine *l*, then a solution that includes job *j* on machine *l* and excludes job *i* is better than a solution that includes job *i* on machine *l* and excludes job *j*. If the number of such

dominating jobs on machine l is no less than the number of machines, then job i cannot be processed on machine l in any optimal solution.

When this condition is valid for all machines, job i cannot be processed on any machine in any optimal solution. c

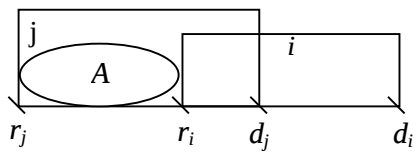
Before implementing our branch and bound procedure, we reduce the problem size by eliminating all jobs that satisfy the condition of Theorem 5.3.2.

Our algorithm uses a depth-first strategy that introduces one job at a time to the partial schedule. The strategy has relatively low memory requirements and complete solutions are likely to be reached earlier, which enables us to get high quality lower bounds earlier, thereby increasing the power of fathoming. We generate all subproblems of a node and select the one having the maximum upper bound for branching. The remaining subproblems are sorted in nonincreasing order of their upper bounds and stored in a candidate list. Each time the current node is revisited, the next subproblem is removed from the list and selected for branching.

Theorems 5.3.3 and 5.3.4 establish some other properties of an optimal solution.

Theorem 5.3.3: If $r_j < r_i < d_j < d_i$, $w_{jk} > w_{ik}$ for machine k, and no job is available for processing during $[r_j,r_i]$, then any optimal solution that includes job i on machine k should also include job j.

Proof: The condition of the theorem is illustrated by the following figure:

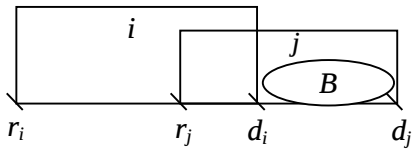


Assume that there is an optimal solution that includes job i on machine k, but job j is not scheduled on any machine. Assume we remove job i, and insert job j to its place. Note that, there is no job in set A. The objective function improves by $w_{jk} - w_{ik} > 0$ units.

Hence, a solution that includes job i on machine k, but not job j, cannot be optimal. c

Theorem 5.3.4: If $r_i < r_j < d_i < d_j$, $w_{jk} > w_{ik}$ for machine k, and no job is available for processing during $[d_i,d_j]$, then any optimal solution that includes job i on machine k should also include job j.

Proof: The condition of the theorem is illustrated by the following figure:



Assume that there is an optimal solution that includes job i on machine k, but job j is not scheduled on any machine. Assume we remove job i, and insert job j to its place. Note that, there is no job in set B. The objective function improves by $w_{jk} - w_{ik} > 0$ units. Hence, a solution that includes job i on machine k, but not job j, cannot be optimal. c

To implement Theorems 5.3.3 and 5.3.4, we proceed as follows: Assume that job i is unscheduled (eliminated), and a decision about job j, such that $r_j \geq d_i$ is about to be made. Then, if $\sum_{l \mid l \text{ overlaps with } i} w_{lk} X_{lk} < w_{ik}$ at the current node, and job i can feasibly be inserted into the schedule if the jobs contributing to the above summation are removed, then the node is fathomed (otherwise any one of the Theorems 5.3.3 and 5.3.4 are violated).

We calculate an upper bound, for each node that cannot be eliminated by the above conditions. We discard the node if its upper bound is no greater than the best known lower bound. We update the best known lower bound when a complete solution with a better objective function value is found.

An optimal solution to the preemptive OFJSE problem where partial solutions are allowed (i.e. $x_{jk} \leq 1$) constitutes an upper bound to the OFJSG problem, since it is a special case. Algorithm 5.3.1 below solves this problem optimally. Note that the algorithm is actually identical to Algorithm 5.2.1, which solves the problem when jobs have unit-processing times.

Algorithm 5.3.1:

S1. Let $\{t_1, t_2,...\}$ be the sorted sequence of the r_js in chronological order with duplicates removed. Let S_a be the set of jobs available in the interval $[t_a, t_a + 1)$ for a=1, 2,...

S2. For each interval $[t_a, t_a + 1)$ for a=1, 2,... :

Choose the most-weighted subset of Min$\{|S_a|, m\}$ jobs among the jobs in S_a by solving the following assignment problem:

$$\text{Maximize } Z_a = \sum_k \sum_j \hat{w}_{jk} x_{jk}$$

$$\text{s.t.} \quad \sum_j x_{jk} \leq 1 \qquad \forall k \qquad (1)$$

$$\sum_k x_{jk} \leq 1 \qquad \forall j \in S_a \qquad (2)$$

$$x_{jk} \in \{0,1\} \qquad \forall j,k,$$

where

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

and

$$\hat{w}_{jk} = \frac{w_{jk}}{p_j} \text{ (weight of job } j \text{ on machine } k \text{ per unit time).}$$

S3. $UBG = \sum_a Z_a$.

Theorem 5.3.2: Algorithm 5.3.1 solves the preemptive OFJSE problem where partial solutions are allowed.

Proof: As $x_{jk} \leq 1$, and r_js are integers, we can treat each job i as $v = p_i$ subjobs with unit processing times. If we let r_{ij} and w_{ijk} be the ready time and the weight of the j^{th} subjob of job i, then the ready

times of these subjobs can be defined as $r_{i1} = r_i$, $r_{i2} = r_i + 1$, …, $r_{iv} = r_i + v - 1$, and weights as $w_{ijk} = w_{ik} / p_i$ for all j. Note that, an optimal solution to the problem with subjobs provides an optimal solution to the OFJSE problem when we impose a constraint that requires either all subjobs of the same job are processed consecutively or none are processed. As we relax this constraint, the resulting problem is the unit processing time problem with $n_1 = \sum_{i=1}^{n} p_i$ jobs, which can be solved by Algorithm 5.3.1. Hence, the solution generated by Algorithm 5.3.1 is optimal. Note that, in Algorithm 5.3.1, the intervals of unit length are merged if they have the same set of available jobs. This reduces the number of intervals from $O(n_1)$ to $O(2n)$. Note that an optimal solution of each interval can be found by solving the assignment model described in S2. c

Corollary 5.3.1: Algorithm 5.3.1 runs in $O(n^4)$ time.

Proof: The complexity of Algorithm 5.3.1 is computed as follows. In S1, the sorting operation takes $O(n \log n)$ time. In S2, for each of the $O(n)$ intervals, an assignment algorithm is solved for $O(n)$ jobs in $O(n^3)$ steps. Hence, the overall complexity of the algorithm is $O(n^4)$. c

Algorithm 5.3.1 can be implemented more efficiently. Consider the following conditions:

- Any job j that completes at $t_a + 1$, i.e. $d_j = t_a + 1$, does not appear in the optimal assignment of interval $[t_a, t_a + 1)$.
- Any job j that arrives at $t_a + 1$, i.e. $r_j = t_a + 1$, satisfies:

  $$w_{jk} \le w^*_{ik}, \forall i \in A_k, \forall k,$$

  where A_k is the set of jobs assigned to machine k in interval $[t_a, t_a + 1)$ and no job in A_k is completed at $t_a + 1$.

When the above conditions hold for all arriving or completing jobs, the optimal assignment of interval $[t_a, t_a+1)$ is also optimal for interval $[t_a+1, t_a+2)$, hence we do not need to solve an assignment problem for that interval.

Instead of using exact assignment solutions in Algorithm 5.3.1, some upper bounds on these solutions can be used. In this case, we sacrifice from quality for the sake of getting quick bounds. Two such bounds can be easily obtained by relaxing the constraint sets of the assignment problem.

Assume we relax constraint set (1), i.e. allow the assignment of more than one job to each particular machine k. The optimal solution to this relaxation is available through the following rule: Starting from the maximum weight, assign the jobs in set S_a to the available machines. When a job is assigned, ignore all remaining weights associated with that job. Stop when m (or $O(m)$) assignments are done. Let UBG_1 be the resulting objective function value. The solution is feasible, hence optimal, if all jobs are assigned to different machines.

Now consider the relaxation of the constraint set (2), i.e. allow the assignment of a particular job to more than one machine. The optimal solution to this relaxation is obtained as follows: Starting from the maximum weight, assign the jobs to the machines. When any job is assigned to a machine, ignore all remaining weights associated with that machine. Stop when m (or $O(m)$) assignments are done. Let UBG_2 be the resulting objective function value. The solution is optimal if all machines are assigned with different jobs.

A third upper bound for the OFJSG problem can be established as follows: Let w_j denote $\underset{k}{Max}\{w_{jk}\}$, i.e. the maximum weight associated with job j. The problem that uses w_js as the weight of jobs is the operational maximal weight problem, and its resulting optimal solution, Z_P^*, is never smaller than the optimal solution of the OFJSG problem. We state this result formally in Theorem 5.3.3.

Theorem 5.3.3: Z_P^* is an upper bound on the OFJSG problem.

Proof: Let X_{jk}^* and Y_{jk}^* be the optimal solutions of the OFJSG and OFJS problems, respectively. Note that:

$$\sum_{j,k} w_{jk} X_{jk}^* \le \sum_{j,k} \underset{k}{Max}\{w_{jk}\} X_{jk}^* = \sum_{j,k} w_j X_{jk}^* \qquad (1),$$

and

$$\sum_{j,k} w_j X_{jk}^* \le \sum_{j,k} w_j Y_{jk}^* \qquad (2),$$

as Y_{jk}^* is the optimal solution of the OFJS problem with weights w_j. Combining (1) and (2) yields:

$$\sum_{j,k} w_j X_{jk}^* \le \sum_{j,k} w_j Y_{jk}^* = Z_P^* .$$

Therefore, Z_P^* is an upper bound on the solution of the OFJSG problem. c

This upper bounding procedure runs in $O(n^3)$ time, as Z_P^* can be found in $O(n^3)$ time by the MCNF algorithm.

The smallest of the above bounds is used as an upper bound at each node of the branch and bound tree, in place of using exact solutions in Algorithm 5.3.1. We also employ the above conditions for a more efficient implementation.

We use two procedures to find an initial feasible solution to our problem. The first one is a greedy approach that assigns jobs to machines starting with the largest weight. The following procedure formally states this first initial lower bounding procedure, which is used at node zero of the branch and bound tree.

Algorithm 5.3.2:

S1. Store all weights in a list in a nonincreasing order. Set $LBG_1 = 0$;

S2. Do:

Let w_{il} be the first weight in the list. Set *job* = *i*, *machine* = *l*.

If *machine* is available during $[r_{job}, d_{job}]$,

Schedule *job* to *machine,*

$LBG_1 = LBG_1 + w_{il}$,

Remove all weights associated with *job* from L.

Else, remove weight w_{il} from L.

Until the list L is empty.

Corollary 5.3.1: Algorithm 5.3.2 runs in $O(nm \log nm)$ time.

Proof: The complexity of Algorithm 5.3.2 is computed as follows: The sorting operation in S1 take $O(nm \log nm)$ time since there are a total of $O(nm)$ weights. In step S2, there are $O(nm)$ checks to be made, therefore the time complexity of this step is $O(nm)$. The overall complexity of the algorithm is $O(nm \log nm)$. c

The second lower bounding procedure is based on decomposition idea, and solves m longest path problems in a sequential manner, one for each machine. The procedure starts with the first machine. Once the longest path is determined on a machine, the jobs in the longest path are eliminated from further consideration. Algorithm 5.3.3 states the procedure formally.

Algorithm 5.3.3:

S1. Set P=∅. Set $A = \{1, 2, \ldots, n\}$.

S2. For each machine $k = 1,\ldots,m$ in sequence, do:

Set $A = A \setminus P$.

Solve a longest path problem with all $j \in A$ using associated weights w_{jk}. Let the resulting objective function value be Z_k, and the jobs in the longest path be LP_k.

Set $P = P \cup LP_k$.

Set $LBG_2 = \sum_{k=1}^{m} Z_k$.

Corollary 5.3.2: Algorithm 5.3.3 runs in $O(mn^2)$ time.

Proof: A Longest Path problem is solved in $O(n^2)$ time for each machine. Algorithm 5.3.3 considers all machines, hence runs in $O(mn^2)$ time. c

The computational experience of the developed optimization procedure for the OFJSG problem will be presented in Chapter 6.

As a final note in this chapter, we present the following theorem, which may be valuable in developing solution strategies for the OFJSG problem, by decomposing the problem into several smaller subproblems. The subproblems can be solved independently by our optimization procedure.

Theorem 5.3.4: (The Decomposition Theorem) Assume that the jobs are indexed in increasing order of their ready times, i.e. $r_1 \leq r_2 \leq \ldots \leq r_n$. If $r_j \geq d_i$, $\forall i<j$, holds for s jobs, then the problem optimally decomposes into $s + 1$ subproblems.

Proof: Note that any job that arrives after time r_j does not overlap with any job that arrives before or on time r_j, when $r_j \geq d_i$ holds for $\forall i<j$. Therefore, the problem optimally decomposes into two subproblems: First one for the jobs that arrive before or on r_j, and second for the jobs that arrive after time r_j. Similarly, if we have s jobs that satisfy the condition, then the problem decomposes into $s + 1$ subproblems. c

CHAPTER 6

COMPUTATIONAL EXPERIENCE

In this chapter, we describe our computational experimentation for the OFJSW, OFJSS and OFJSG problems, and analyze the results. We give the details of our experimental design in Section 6.1, and discuss the results in Section 6.2.

6.1. DESIGN OF EXPERIMENTS

We describe our data generation procedure in this section. There are some common issues on the experimental design for the three problems, while problem specific data are also generated.

As a common design issue, random test problems are generated starting from $n = 20$ jobs and increasing by increments of 10 up to 100 jobs. We use $m = 2$, 3, and 4 machines for all problems.

Two sets of ready times are generated as in Fischetti *et al.* (1989) for all problems. In the first set (when $r = 1$), ready times follow a discrete uniform distribution in the range [0,200]. In the second set (when $r = 2$), a peak time distribution is considered, where 30% of the ready times are uniformly random in the range [30,40], 30% in the range [130,140], and the remaining 40% in ranges [0,29], [41,129], and [141,200].

For each ready time set, we use two uniform distributions for processing times for all problems: [5,10] ($p = 1$) and [5,40] ($p = 2$).

For the OFJSW and OFJSS problems, we consider three weight sets: $w_j = p_j$ ($w = 1$), uniform in range [5,10] (low variability case, $w = 2$), and [5,40] (high variability case, $w = 3$). Note that, when $w_j = p_j$, the objective function of the OFJSW problem reduces to the maximization of the total processing time. The working time limit, T, is taken as 50 time units in the OFJSW problem, while the spread time limit, S, is determined as 100 time units for the OFJSS problem.

For the OFJSG problem, we consider two weight sets: w_{jk} uniform in range [0,10] (low variability case, $w = 1$), and [0,40] (high variability case, $w = 2$). Note that, zero weights are also allowed to have the most general weight environment for the problem.

With the given experimental factors and their associated levels, a total of 486 different parameter combinations are created for the OFJSW and OFJSS problems, while this number is 324 for the OFJSG problem.

We generate 10 random instances for each problem combination. All algorithms are coded in MS Visual C++ 6.0, and solved on a computer having 2.8 GHz Pentium IV processor with 1 GB memory. We terminate execution of the branch and bound algorithm, if an optimal solution is not found in 1 hour of Central Processing Unit (CPU) time.

6.2. ANALYSIS OF RESULTS

In this section, we present the results of our computational experience for each of the three problems described in previous chapters, i.e. the OFJSW, OFJSS and OFJSG problems. Discussion of computational results is presented, and relations between the results for different problems are explored, as well.

6.2.1. Results for the OFJSW Problem

We first investigate the performance of our lower and upper bounds. Table 6.2.1 and Table 6.2.2 report the average and maximum root node lower bound and upper bound deviations as a percentage of optimal solutions for r =1 and r =2, respectively. The "opt." columns present number of times the lower bound finds the optimal solution at root node (out of 10 instances). An empty entry in the tables indicates that the algorithm could not return an optimal solution within the one-hour time limit for any of the 10 problem instances. As can be observed from the tables, the lower and upper bounds are quite powerful. When r = 1, the lower and upper bounding procedures yield respective deviations of no more than 14% and 11% from the optimal solutions. When r = 2, the maximum respective deviations become 18% and 25%.

Next, we present the results of the computational runs. In Tables 6.2.3 and 6.2.4, the average and maximum CPU times (in seconds) and the average and maximum number of nodes are presented for r =1 and r =2, respectively. The numbers in parentheses are the number of instances that cannot be solved within our termination limit of one hour. It can be seen that r = 1 case is easier than r = 2 case. Note that when r = 1, the upper bound performances are better, so are the solution times. When the processing times are concerned, p = 1 (low variability) combination reduces the efficiency, as expected. Note that, when the processing times are larger the number of alternative machines is lower; hence, fewer number of node evaluations are necessary. We observe that when $w_j = p_j$, the problem becomes easier. When the weight variability is higher, a small change in the solution is likely to reduce the objective function value, which in turn causes the best solution to be updated more frequently, thereby increasing solution times.

Table 6.2.1. LB and UB deviations at root node for r =1 (OFJSW)

			m=2					m=3					m=4				
			LB Gap (%)		UB Gap (%)			LB Gap		UB Gap			LB Gap		UB Gap		
n	p	w	Avg.	Max.	Avg.	Max.	opt.	Avg.	Max.	Avg.	Max.	opt.	Avg.	Max.	Avg.	Max.	opt.
20	1	1	3.00	7.00	0.00	0.00	1	2.20	4.03	0.42	2.90	3	0.00	0.00	0.30	2.96	10
		2	2.81	6.15	1.03	2.52	2	2.02	6.08	0.84	2.88	4	0.00	0.00	0.50	2.88	10
		3	0.74	2.17	1.64	3.54	3	1.71	3.95	0.62	3.59	2	0.00	0.00	0.00	0.00	10
	2	1	3.40	13.00	0.00	0.00	1	5.13	14.00	0.00	0.00	0	4.32	8.63	0.25	1.52	1
		2	4.24	8.77	4.14	7.46	1	6.01	10.53	3.30	6.58	0	7.25	13.98	3.41	10.81	1
		3	3.83	13.44	4.55	8.63	2	6.13	11.69	2.47	7.17	0	5.06	8.51	2.85	6.16	0
30	1	1	2.30	5.00	0.00	0.00	2	1.80	3.33	0.00	0.00	0	2.45	5.00	0.20	1.52	1
		2	2.68	6.14	0.34	0.88	1	2.96	5.45	0.17	0.60	0	3.00	4.93	0.67	1.48	0
		3	2.86	5.68	1.64	6.75	0	1.55	3.26	1.25	3.27	2	0.88	2.37	0.41	0.98	2
	2	1	1.30	4.00	0.00	0.00	4	2.07	4.00	0.00	0.00	2	2.45	9.50	0.00	0.00	2
		2	3.46	7.04	3.34	6.33	2	5.33	9.78	1.79	5.43	1	7.12	9.82	1.58	4.31	0
		3	3.78	7.60	2.02	4.29	2	4.12	8.97	2.23	4.92	0	5.97	8.66	2.34	6.50	0
40	1	1	1.50	3.00	0.00	0.00	3	1.93	3.33	0.00	0.00	2	1.80	4.50	0.51	2.04	2
		2	2.32	5.04	0.67	1.63	0	2.07	3.59	0.37	1.60	0	-	-	-	-	-
		3	3.49	7.19	0.85	2.25	0	2.51	5.66	0.88	2.69	0	-	-	-	-	-
	2	1	2.00	5.00	0.00	0.00	3	2.67	6.67	0.00	0.00	1	2.70	9.50	0.00	0.00	0
		2	5.02	10.71	2.40	6.58	0	5.93	11.21	0.91	2.75	1	6.00	9.09	0.63	2.63	0
		3	3.75	7.97	2.81	6.03	1	5.16	9.54	1.29	4.62	0	6.15	8.64	0.66	1.36	0
50	1	1	1.90	4.00	0.00	0.00	2	1.53	2.67	0.00	0.00	2	2.05	4.00	0.25	2.04	1
		2	3.08	5.81	0.55	1.49	0	3.00	6.00	0.15	0.54	0	-	-	-	-	-
		3	2.88	7.34	0.66	1.89	1	2.89	5.30	0.31	0.74	0	-	-	-	-	-
	2	1	1.30	3.00	0.00	0.00	4	1.67	3.33	0.00	0.00	0	0.85	2.00	0.00	0.00	2
		2	5.36	12.90	0.74	3.95	1	5.56	8.74	0.60	1.72	0	5.12	9.02	0.79	2.14	0
		3	4.73	11.30	2.03	3.56	1	4.72	8.87	1.83	3.29	1	4.24	7.71	0.90	1.81	0
60	1	1	1.10	3.00	0.00	0.00	2	1.67	4.67	0.00	0.00	3	0.60	1.52	0.92	3.09	4
		2	2.54	5.52	0.20	0.66	2	2.28	4.05	0.21	1.59	0	-	-	-	-	-
		3	3.15	6.17	0.54	1.35	1	3.70	5.36	0.17	0.56	0	-	-	-	-	-
	2	1	1.10	4.00	0.00	0.00	5	1.07	3.33	0.00	0.00	3	1.40	3.00	0.00	0.00	2
		2	2.81	8.00	2.11	4.17	1	5.21	10.83	0.73	3.33	0	5.87	8.61	0.28	0.71	0
		3	2.81	5.78	2.30	5.31	2	5.14	8.10	1.12	4.38	0	4.79	7.71	0.66	1.75	0
70	1	1	0.50	3.00	0.00	0.00	7	1.47	3.33	0.00	0.00	1	1.15	2.50	0.36	2.04	3
		2	2.69	5.44	0.54	1.36	0	2.56	4.29	0.00	0.00	0	-	-	-	-	-
		3	3.51	7.06	0.49	1.65	0	3.03	3.85	0.20	0.42	0	-	-	-	-	-
	2	1	0.60	2.00	0.00	0.00	6	0.73	2.00	0.00	0.00	3	0.80	2.50	0.00	0.00	4
		2	3.82	8.82	1.81	5.05	1	5.13	11.58	0.62	2.21	1	5.24	9.42	0.60	1.81	0
		3	3.10	7.14	0.87	3.79	1	3.99	8.08	0.58	1.14	0	5.04	11.62	0.77	1.69	0
80	1	1	0.00	0.00	0.00	0.00	10	1.47	4.00	0.00	0.00	2	1.25	3.00	0.30	1.52	4
		2	2.30	4.17	0.82	1.33	1	3.28	5.94	0.18	0.86	0	-	-	-	-	-
		3	3.45	6.79	0.52	1.40	0	2.31	3.58	0.18	1.37	0	-	-	-	-	-
	2	1	0.80	4.00	0.00	0.00	7	0.87	2.67	0.00	0.00	3	0.65	2.50	0.00	0.00	4
		2	3.53	5.93	0.96	2.17	1	4.55	8.26	0.93	2.63	0	5.38	8.43	0.28	1.06	0
		3	3.93	6.69	1.96	4.00	1	3.38	7.97	0.60	1.14	0	3.35	5.29	0.38	0.78	0
90	1	1	0.00	0.00	0.00	0.00	10	1.20	2.00	0.00	0.00	1	0.55	1.50	0.36	2.04	4
		2	2.61	4.67	0.31	0.66	1	3.05	4.63	0.23	0.99	0	-	-	-	-	-
		3	2.44	4.49	0.64	1.84	0	2.80	5.13	0.43	1.29	0	-	-	-	-	-
	2	1	0.10	1.00	0.00	0.00	9	0.93	3.33	0.00	0.00	4	0.80	3.50	0.00	0.00	3
		2	1.70	4.13	1.67	4.08	4	3.49	7.44	0.37	0.83	0	4.55	8.44	0.48	1.16	0
		3	2.90	7.08	1.63	4.75	1	4.41	7.62	0.85	2.57	1	3.91	7.00	0.63	2.41	0
100	1	1	0.00	0.00	0.00	0.00	10	0.40	2.00	0.00	0.00	6	0.90	2.50	0.25	1.52	3
		2	3.41	5.06	0.50	0.64	0	2.35	4.18	0.18	0.47	2	-	-	-	-	-
		3	2.94	5.65	0.70	1.80	0	2.16	4.44	0.28	0.86	0	-	-	-	-	-
	2	1	0.00	0.00	0.00	0.00	10	0.20	0.67	0.00	0.00	7	0.60	2.00	0.00	0.00	4
		2	2.86	6.93	0.63	1.90	2	3.73	8.63	0.46	1.87	0	4.49	6.67	0.17	0.57	0
		3	2.32	4.28	1.62	3.91	1	3.86	7.01	0.67	1.23	0	3.74	7.60	0.32	0.85	0

Table 6.2.2. LB and UB deviations at root node for *r*=2 (OFJSW)

			m =2					*m* =3					*m* =4				
			LB Gap (%)		UB Gap (%)			LB Gap		UB Gap			LB Gap		UB Gap		
n	*p*	*w*	Avg.	Max.	Avg.	Max.	opt.	Avg.	Max.	Avg.	Max.	opt.	Avg.	Max.	Avg.	Max.	opt.
20	1	1	3.87	10.64	4.52	25.45	3	3.12	8.96	7.47	15.24	4	0.95	3.92	4.08	8.00	7
		2	2.55	6.73	8.42	19.59	2	1.98	5.13	10.33	17.00	4	0.51	5.08	8.93	17.48	9
		3	1.48	9.56	12.81	23.45	6	1.58	5.83	7.81	15.79	6	0.33	1.97	4.30	19.65	8
	2	1	4.60	14.00	0.00	0.00	1	1.80	4.00	0.07	0.67	2	4.45	10.00	0.10	1.01	0
		2	6.46	12.00	9.40	15.79	1	6.31	10.75	5.14	13.70	0	4.97	8.70	6.68	20.25	1
		3	5.25	10.16	4.96	10.40	1	4.65	13.00	3.62	6.21	1	5.72	11.03	3.69	6.83	0
30	1	1	2.20	7.00	0.00	0.00	2	5.17	12.08	1.85	10.83	0	3.81	11.31	7.49	14.86	3
		2	3.30	5.88	3.55	10.34	0	2.96	6.75	7.95	19.35	1	1.68	3.98	11.87	19.08	4
		3	2.81	4.91	8.73	18.36	1	1.95	3.54	6.84	14.56	0	1.43	3.32	7.25	12.70	1
	2	1	1.60	4.00	0.00	0.00	2	2.67	5.33	0.00	0.00	0	4.15	12.00	0.00	0.00	0
		2	6.31	11.67	7.40	20.75	0	9.26	16.09	4.40	8.75	0	7.50	11.86	2.14	3.77	0
		3	4.93	13.33	10.62	17.73	1	4.71	9.92	6.34	15.15	2	6.32	9.64	5.66	12.46	0
40	1	1	2.40	4.00	0.00	0.00	1	2.45	5.67	1.06	6.38	1	2.61	5.50	2.76	7.53	2
		2	2.43	4.27	3.80	9.40	1	4.83	9.04	4.00	9.04	0	4.16	6.33	5.45	9.91	0
		3	3.54	6.48	5.17	8.80	1	4.35	8.25	9.86	21.32	0	3.24	7.50	6.25	11.87	0
	2	1	2.10	5.00	0.00	0.00	1	1.13	3.33	0.00	0.00	2	2.40	4.00	0.00	0.00	1
		2	6.56	13.51	5.70	12.99	0	6.93	13.19	3.90	7.69	0	6.91	11.21	3.43	7.89	0
		3	5.49	11.57	8.72	19.72	1	6.82	14.98	4.03	7.72	1	5.86	9.46	2.68	7.53	0
50	1	1	1.90	5.00	0.00	0.00	1	2.20	3.33	0.00	0.00	0	1.52	7.65	1.91	10.50	5
		2	4.12	7.75	4.67	8.80	0	4.47	11.23	3.15	5.24	0	-	-	-	-	-
		3	2.78	5.84	3.78	6.78	0	3.21	6.74	4.31	10.91	0	-	-	-	-	-
	2	1	1.00	4.00	0.00	0.00	4	1.20	3.33	0.00	0.00	2	2.15	4.50	0.00	0.00	0
		2	5.10	10.64	5.44	11.59	1	7.06	10.78	4.03	9.28	0	5.16	11.72	3.52	6.50	0
		3	4.74	10.80	5.98	20.49	3	6.35	10.30	5.64	10.88	0	5.87	8.32	3.56	6.47	0
60	1	1	1.50	4.00	0.00	0.00	2	2.27	3.33	0.00	0.00	0	1.40	3.00	0.82	3.09	2
		2	3.02	9.68	3.56	6.87	1	3.34	4.52	2.69	5.29	0	-	-	-	-	-
		3	3.82	7.49	4.00	8.26	1	2.61	4.23	3.97	6.76	0	-	-	-	-	-
	2	1	0.60	1.00	0.00	0.00	4	0.73	2.67	0.00	0.00	4	1.65	4.00	0.00	0.00	2
		2	6.56	13.64	4.97	12.12	0	7.50	10.19	5.21	13.68	0	5.91	9.36	3.83	7.25	0
		3	3.83	11.89	6.04	19.22	1	5.95	9.32	5.57	12.50	0	5.02	8.28	2.76	6.12	0
70	1	1	1.20	3.00	0.00	0.00	4	2.47	4.67	0.00	0.00	0	1.31	3.50	0.91	2.56	3
		2	2.55	5.30	2.30	5.19	0	4.22	6.45	2.85	6.45	0	-	-	-	-	-
		3	3.64	6.69	2.68	6.39	0	2.61	4.65	3.09	6.20	0	-	-	-	-	-
	2	1	0.60	2.00	0.00	0.00	7	0.87	6.00	0.00	0.00	6	0.90	2.00	0.00	0.00	2
		2	5.46	9.64	6.29	10.77	0	6.71	18.18	4.39	17.27	0	6.54	10.14	3.95	7.19	0
		3	3.86	10.03	5.98	9.89	1	5.83	7.99	2.24	6.76	0	6.02	8.60	1.91	5.02	0
80	1	1	1.30	3.00	0.00	0.00	3	1.53	3.33	0.00	0.00	0	1.80	4.00	0.41	2.04	1
		2	3.38	6.54	1.94	3.40	0	3.33	5.29	2.16	4.09	0	-	-	-	-	-
		3	2.75	4.64	2.34	4.43	0	2.84	5.94	2.70	5.42	0	-	-	-	-	-
	2	1	0.90	2.00	0.00	0.00	3	0.67	2.00	0.00	0.00	3	1.20	3.50	0.00	0.00	2
		2	4.90	10.23	6.33	10.48	0	6.82	11.86	4.74	11.21	0	6.40	9.09	3.09	7.88	0
		3	2.96	4.89	4.57	13.42	2	4.77	8.07	6.16	11.06	0	5.11	9.28	2.58	5.85	0
90	1	1	1.00	2.00	0.00	0.00	3	1.67	4.67	0.00	0.00	2	1.45	3.50	0.56	2.56	3
		2	3.15	4.67	2.66	5.04	0	2.70	5.74	2.19	4.76	0	-	-	-	-	-
		3	2.53	5.69	3.30	5.44	1	3.73	5.50	2.81	6.09	0	-	-	-	-	-
	2	1	0.60	4.00	0.00	0.00	7	0.60	2.00	0.00	0.00	5	0.30	1.00	0.00	0.00	5
		2	4.45	11.24	5.28	13.48	1	4.95	11.40	3.02	10.71	1	5.69	10.53	3.71	8.19	0
		3	4.78	10.06	4.99	19.53	0	4.62	7.18	4.78	6.81	0	4.88	8.85	2.98	6.78	0
100	1	1	1.10	4.00	0.00	0.00	5	1.47	3.33	0.00	0.00	2	1.31	2.04	0.72	3.09	0
		2	3.50	6.54	1.78	3.87	0	4.00	7.14	2.02	3.57	0	-	-	-	-	-
		3	2.81	5.83	2.50	4.04	1	2.62	5.08	3.40	8.13	0	-	-	-	-	-
	2	1	0.80	4.00	0.00	0.00	6	1.13	4.00	0.00	0.00	2	0.60	2.00	0.00	0.00	4
		2	3.10	4.26	3.51	7.76	1	6.44	12.12	3.57	5.65	0	6.28	8.23	4.73	9.49	0
		3	3.59	7.92	5.45	9.66	0	5.53	9.98	3.46	11.74	0	3.40	4.85	3.76	5.21	0

Table 6.2.3. Branch and bound performance for r =1 (OFJSW)

			m =2				*m* =3				*m* =4			
			CPU time		# of nodes		CPU time		# of nodes		CPU time		# of nodes	
n	*p*	*w*	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.
20	1	1	0.04	0.28	2,255	14,751	1.12	4.47	61,905	236,514	12.08	120.84	455,175	4,551,746
		2	0.03	0.17	1,704	9,384	2.17	11.03	130,228	669,221	0.00	0.00	1	4
		3	0.03	0.06	1,357	3,264	7.35	56.92	332,563	2,384,084	0.00	0.00	0	0
	2	1	0.01	0.03	547	1,713	0.32	1.98	14,843	88,113	1.26	4.39	67,434	260,438
		2	0.01	0.02	254	762	0.03	0.05	907	2,336	0.17	0.42	8,200	22,063
		3	0.01	0.05	266	1,113	0.03	0.08	1,409	4,602	0.07	0.33	3,130	15,714
30	1	1	0.08	0.47	3,437	22,349	2.19	12.34	106,510	602,532	963.05 **(2)**	3600.00	47,706,930	188,082,533
		2	0.10	0.78	5,465	41,529	37.14	277.77	1,991,300	14,838,614	2520.02 **(7)**	3600.00	116,987,251	187,180,706
		3	0.15	0.52	8,170	28,359	336.67	2418.14	17,043,266	125,146,611	2881.15 **(7)**	3600.00	138,628,253	181,540,361
	2	1	0.01	0.03	277	1,033	0.14	0.66	6,632	31,344	4.40	32.67	184,036	1,357,997
		2	0.02	0.05	862	1,713	0.13	0.77	6,101	38,496	2.24	9.89	95,917	408,137
		3	0.02	0.05	701	1,982	0.10	0.39	4,813	19,861	0.79	3.72	35,488	176,688
40	1	1	0.06	0.17	2,524	7,446	3.27	13.31	131,369	537,121	1533.50 **(4)**	3600.00	59,914,293	149,620,847
		2	0.44	3.45	20,766	160,614	93.57	492.23	3,894,037	19,982,092	-	-	-	-
		3	0.28	1.80	14,339	94,792	249.13	1892.11	10,656,156	82,029,375	-	-	-	-
	2	1	0.01	0.03	283	897	0.36	1.06	13,061	36,880	11.76	99.72	421,261	3,509,020
		2	0.04	0.08	1,386	3,067	0.71	1.84	26,221	72,336	6.40	30.22	243,168	1,119,148
		3	0.04	0.17	1,639	8,004	0.41	1.53	16,697	67,787	4.30	27.22	175,171	1,135,865
50	1	1	0.06	0.17	2,540	7,079	3.07	22.09	113,939	843,035	1165.81 **(2)**	3600.00	40,755,533	123,408,814
		2	1.81	16.05	82,142	725,648	318.68	945.11	12,820,574	39,482,886	-	-	-	-
		3	0.38	2.52	17,244	115,344	276.46	1286.64	11,493,919	49,969,862	-	-	-	-
	2	1	0.03	0.14	976	6,204	0.20	0.81	7,179	30,927	14.81	51.42	503,061	1,719,003
		2	0.02	0.05	554	1,738	0.58	1.89	22,576	77,077	44.96	198.92	1,543,412	6,903,348
		3	0.05	0.14	2,011	6,037	0.59	2.09	22,931	81,819	7.95	25.31	288,802	919,564

Table 6.2.3. (cont'd) Branch and bound performance for r =1 (OFJSW)

			m =2				m =3				m =4			
			CPU time		# of nodes		CPU time		# of nodes		CPU time		# of nodes	
n	p	w	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.
60	1	1	0.04	0.06	1,228	2,515	13.49	73.09	443,468	2,366,539	1779.68 **(4)**	3600.00	53,837,505	115,986,396
		2	0.26	0.88	11,090	38,892	739.94 **(1)**	3600.00	27,273,438	129,892,759	-	-	-	-
		3	0.71	3.44	30,117	148,358	773.43 **(1)**	3600.00	29,729,577	138,196,738	-	-	-	-
	2	1	0.01	0.06	387	2,167	0.28	1.31	10,114	49,123	8.97	31.78	271,624	977,645
		2	0.08	0.27	3,199	11,431	0.91	1.56	34,334	63,447	125.84	807.06	4,185,758	27,441,930
		3	0.05	0.20	2,190	7,807	1.36	6.38	49,788	235,553	80.90	361.25	2,584,951	11,979,150
70	1	1	0.05	0.22	1,618	8,113	27.05	248.58	772,443	7,012,419	750.06 **(2)**	3600.00	21,208,634	101,877,972
		2	0.52	2.72	20,630	105,039	32.68	184.49	1,083,869	6,002,817	-	-	-	-
		3	1.11	4.20	44,356	167,444	718.71 **(1)**	3600.00	25,756,284	132,163,848	-	-	-	-
	2	1	0.02	0.09	683	2,751	0.49	1.53	16,098	47,114	8.20	38.58	236,023	1,146,168
		2	0.13	0.45	5,273	17,519	4.11	17.78	142,954	622,702	73.13	483.73	2,157,757	14,365,845
		3	0.06	0.19	2,069	6,950	2.76	12.41	99,147	463,961	165.67	1013.55	4,759,861	29,319,246
80	1	1	0.00	0.00	0	0	35.59	231.00	1,069,718	6,892,184	1286.10 **(3)**	3600.00	33,229,105	100,026,328
		2	4.84	27.77	196,588	1,142,307	360.22	3056.53	11,851,756	101,496,713	-	-	-	-
		3	2.75	12.49	105,645	469,581	363.40	1087.00	12,030,410	37,132,849	-	-	-	-
	2	1	0.00	0.02	101	619	0.16	0.52	4,391	15,272	5.26	19.94	136,086	510,438
		2	0.06	0.20	2,338	7,881	47.60	457.55	1,610,061	15,493,529	1025.06 **(2)**	3600.00	30,348,360	111,404,924
		3	0.08	0.19	3,006	7,498	2.48	6.52	79,249	208,546	84.36	267.52	2,432,865	8,414,521
90	1	1	0.00	0.00	0	0	26.69	222.14	812,143	6,811,096	1280.50 **(2)**	3600.00	31,361,953	87,459,048
		2	3.04	12.17	113,407	458,314	1257.52 **(3)**	3600.00	38,198,267	114,949,812	-	-	-	-
		3	5.76	37.47	218,731	1,441,512	2095.43 **(4)**	3600.00	64,554,775	112,365,762	-	-	-	-
	2	1	0.00	0.03	52	523	0.40	2.50	11,014	71,299	32.49	223.33	803,980	5,508,533
		2	0.33	1.58	12,742	60,096	6.33	42.56	194,613	1,270,281	256.57	870.69	6,758,644	22,420,936
		3	0.15	0.31	5,685	11,722	18.14	132.03	579,628	4,217,326	551.73 **(1)**	3600.00	14,376,542	93,323,439
100	1	1	0.00	0.00	0	0	12.76	70.27	327,346	1,759,495	930.96 **(2)**	3600.00	21,675,000	86,433,080
		2	5.87	27.09	191,442	884,784	494.09 **(1)**	3600.00	13,987,574	105,322,481	-	-	-	-
		3	21.07	129.66	680,606	4,241,219	1756.85 **(3)**	3600.00	47,757,270	103,131,644	-	-	-	-
	2	1	0.00	0.00	0	0	0.93	7.75	23,428	194,441	1.73	13.77	37,434	297,511
		2	0.08	0.33	2,845	11,743	4.02	19.33	112,163	496,219	443.78	2871.89	10,341,580	67,096,572
		3	0.10	0.44	3,395	14,913	7.31	23.89	206,110	699,688	1059.38 **(2)**	3600.00	25,576,680	89,482,565

Table 6.2.4. Branch and bound performance for r =2 (OFJSW)

			m =2				m =3				m =4			
			CPU time		# of nodes		CPU time		# of nodes		CPU time		# of nodes	
n	p	w	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.
20	1	1	0.04	0.13	1,701	5,802	3.77	13.22	152,248	546,482	16.56	91.64	613,899	3,455,385
		2	0.06	0.19	2,553	8,428	1.42	11.61	61,085	510,281	17.87	108.38	675,444	3,980,648
		3	0.03	0.08	1,383	3,134	1.10	4.77	43,538	194,579	4.77	23.69	177,641	894,889
	2	1	0.01	0.03	652	1,591	0.08	0.19	3,680	9,839	1.17	3.48	60,556	196,063
		2	0.01	0.02	216	818	0.04	0.22	1,953	11,715	0.16	1.13	7,806	56,904
		3	0.01	0.02	211	395	0.02	0.06	751	3,968	0.04	0.09	1,705	4,496
30	1	1	0.05	0.34	2,490	17,059	36.08	112.66	1,509,165	4,431,131	1984.63 **(3)**	3600.00	64,845,823	122,148,892
		2	0.35	1.08	17,938	54,897	84.89	457.11	3,808,246	19,970,149	1385.93 **(3)**	3600.00	44,959,175	122,209,269
		3	0.23	0.63	11,401	32,706	96.39	757.00	4,221,202	33,053,703	1588.04 **(1)**	3600.00	51,829,897	117,327,504
	2	1	0.01	0.03	368	1,230	0.28	2.14	12,298	93,744	5.46	28.70	229,401	1,130,915
		2	0.01	0.03	415	1,254	0.09	0.34	4,015	15,300	0.33	1.86	13,837	75,895
		3	0.02	0.05	734	1,869	0.12	0.48	5,611	24,370	0.65	3.24	27,776	141,041
40	1	1	0.07	0.22	2,581	9,099	370.78 **(1)**	3600.00	13,314,187	129,205,355	2453.74 **(5)**	3600.00	79,957,005	130,434,318
		2	0.75	3.11	32,930	133,484	640.87 **(1)**	3600.00	25,732,711	144,011,865	2720.42 **(7)**	3600.00	92,314,270	129,202,858
		3	1.16	4.14	54,772	209,455	884.82 **(1)**	3600.00	33,919,851	140,469,090	3069.20 **(8)**	3600.00	105,064,084	147,289,758
	2	1	0.05	0.13	2,194	5,452	0.23	0.77	8,871	29,665	3.87	22.13	141,709	794,474
		2	0.02	0.08	838	4,435	0.64	4.23	26,450	171,383	1.73	7.25	67,835	281,170
		3	0.03	0.14	1,336	6,437	0.55	1.64	22,566	67,021	6.82	25.08	243,521	872,498
50	1	1	0.11	0.52	4,008	20,094	13.31	67.66	450,065	2,185,537	2691.28 **(7)**	3600.00	79,566,263	113,804,538
		2	2.40	7.11	105,903	331,529	698.90 **(1)**	3600.00	25,968,710	132,816,308	-	-	-	-
		3	2.02	8.39	91,900	370,013	1122.13 **(2)**	3600.00	41,736,065	133,008,494	-	-	-	-
	2	1	0.02	0.08	631	3,636	0.60	4.28	23,381	172,457	7.85	37.80	256,636	1,160,066
		2	0.04	0.08	1,707	3,396	1.72	9.53	69,934	393,340	8.87	29.53	306,621	1,058,260
		3	0.08	0.66	3,453	28,351	0.94	4.06	36,893	154,758	34.31	259.44	1,182,937	9,003,661

Table 6.2.4. (cont'd) Branch and bound performance for r =2 (OFJSW)

			m =2				m =3				m =4			
			CPU time		# of nodes		CPU time		# of nodes		CPU time		# of nodes	
n	p	w	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.
60	1	1	0.28	0.92	9,802	30,220	23.96	145.02	785,185	4,909,083	1832.69 **(5)**	3600.00	52,629,602	104,730,240
		2	4.22	16.63	153,300	548,784	1479.43 **(2)**	3600.00	54,027,089	130,330,895	-	-	-	-
		3	9.21	40.34	376,500	1,615,600	1356.94 **(3)**	3600.00	48,640,324	129,454,562	-	-	-	-
	2	1	0.03	0.17	1,086	5,799	1.07	5.52	36,551	189,688	9.19	65.88	264,184	1,852,513
		2	0.06	0.14	2,216	5,717	8.43	75.38	314,613	2,799,067	65.54	262.58	2,091,232	8,318,734
		3	0.08	0.22	3,061	9,174	1.81	7.13	67,694	250,811	39.23	258.72	1,227,167	7,800,135
70	1	1	0.22	1.48	7,910	55,217	34.61	279.78	1,003,681	8,023,430	2425.63 **(6)**	3600.00	64,446,746	111,486,609
		2	1.33	4.11	55,543	183,369	2045.36 **(5)**	3600.00	67,746,358	133,064,064	-	-	-	-
		3	1.70	8.19	68,507	325,967	1548.08 **(2)**	3600.00	52,444,156	124,535,445	-	-	-	-
	2	1	0.01	0.05	241	1,876	0.40	2.74	11,923	81,741	11.84	55.47	326,960	1,536,813
		2	0.09	0.19	3,859	7,696	1.56	9.97	58,156	375,302	143.98	982.38	4,233,507	28,324,634
		3	0.17	1.03	7,365	47,836	3.18	24.31	101,803	746,412	13.90	66.14	406,962	1,855,022
80	1	1	0.98	4.70	32,375	152,709	16.24	133.14	471,083	3,866,467	1194.81 **(3)**	3600.00	31,555,975	104,083,969
		2	5.39	38.64	204,012	1,444,332	1832.13 **(4)**	3600.00	59,440,138	122,309,899	-	-	-	-
		3	3.01	13.19	120,631	522,196	1160.48 **(2)**	3600.00	37,419,184	116,600,539	-	-	-	-
	2	1	0.02	0.08	738	2,578	3.04	28.00	83,888	771,519	8.19	35.89	209,135	888,639
		2	0.33	1.13	14,540	51,526	17.93	136.00	643,799	4,968,803	502.68	2200.52	14,824,632	59,611,463
		3	0.07	0.25	2,761	9,900	31.62	248.97	1,164,729	9,145,666	171.11	947.38	4,808,975	26,731,680
90	1	1	0.23	1.00	7,516	34,307	6.45	20.86	172,628	574,275	1755.49 **(3)**	3600	41751941.5	92437815
		2	2.80	8.95	113,809	362,239	1985.26 **(4)**	3600.00	57,547,364	109,787,871	-	-	-	-
		3	21.42	192.06	801,850	7,157,069	1684.64 **(3)**	3600.00	53,435,972	118,592,306	-	-	-	-
	2	1	0.03	0.16	797	4,683	0.68	3.47	18,958	96,387	19.57	188.53	463,451	4,454,333
		2	0.23	0.78	9,665	33,852	95.01	864.08	3,361,004	30,621,360	1007.54 **(2)**	3600.00	27,289,946	100,966,740
		3	0.79	6.75	33,353	288,005	27.94	126.33	961,168	4,397,512	205.05	829.06	5,467,257	21,805,707
100	1	1	2.09	18.28	58,700	513,324	51.05	306.69	1,298,562	7,873,267	1509.23 **(4)**	3600.00	32,669,626	79,635,312
		2	4.68	30.34	151,832	959,209	1676.45 **(4)**	3600.00	46,800,843	105,133,712	-	-	-	-
		3	17.96	91.80	626,062	3,181,592	2614.42 **(6)**	3600.00	72,376,520	101,598,681	-	-	-	-
	2	1	0.04	0.23	835	4,628	0.99	6.19	24,084	150,273	3.65	19.61	80,760	433,583
		2	0.37	2.56	13,325	89,576	36.55	208.45	1,169,569	6,884,457	1890.09 **(3)**	3600.00	47,173,985	96,236,669
		3	0.23	0.69	9,074	27,417	97.45	879.31	2,991,230	27,053,844	1830.76 **(4)**	3600.00	43,822,443	95,097,436

To see the effect of number of machines on solution time and percent deviations more clearly, some additional runs are performed with 6 and 8 machines for 20 to 70 jobs with the combination r = 1 and p = 2. Table 6.2.5 presents the results of associated runs. The exponential effect of the number of machines on the solution time and the number of nodes are quite obvious from the table. However, as the number of machines increase, our initial upper bound seems to get better. This is due to the fact that more jobs (even all jobs) can be processed feasibly as the number of machines increases for a given n, thereby the gap between the surrogate relaxation and the optimal solution decreases.

Next, we investigate the effect of the upper and lower bounding mechanisms and the dominance relations on the performance of the algorithm. For presenting the effects, the (n, m) = (20,3) and (30,2) combinations are selected as all instances of these combinations can be solved within the one-hour CPU time limit. The effects are similar for other combinations, as well.

In Table 6.2.6, the CPU times, the number of nodes, and the upper bound deviations are presented, when UB_1 and UB_2 are utilized as upper bounds. The values are also presented for the case when a simple upper bound is used. The simple upper bound takes the weights of all not-yet-considered jobs. Note that, with simple upper bound case, no instance can be solved for n =30, m =2, r =1, p =1, w =1 combination, however all these instances are solved within one second when UB_1 and/or UB_2 are used.

In Table 6.2.7, the effect of our lower bound on the efficiency of the branch and bound algorithm is reported. When an initial lower bounding procedure is not used, the lower bound is simply taken as zero. In all instances, the average number of nodes, in turn CPU times, increase slightly, but not significantly, when the lower bound is removed. This is due to the fact that depth-first strategy returns a complete solution at early nodes, and this solution is likely to be a strong one due to the strong upper bounds.

Table 6.2.5. Effect of *m* on the BB performance (*r* =1, *p* = 2) (OFJSW)

		n =20								*n* =30							
		CPU time		# of nodes		LB Gap		UB Gap		CPU time		# of nodes		LB Gap		UB Gap	
m	*w*	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.
2	1	0.01	0.03	547	1,713	3.40	13.00	0.00	0.00	0.01	0.03	277	1,033	1.30	4.00	0.00	0.00
	2	0.01	0.02	254	762	4.24	8.77	4.14	7.46	0.02	0.05	862	1,713	3.46	7.04	3.34	6.33
	3	0.01	0.05	266	1,113	3.83	13.44	4.55	8.63	0.02	0.05	701	1,982	3.78	7.60	2.02	4.29
4	1	1.26	4.39	67,434	260,438	4.32	8.63	0.25	1.52	4.40	32.67	184,036	1,357,997	2.45	9.50	0.00	0.00
	2	0.17	0.42	8,200	22,063	7.25	13.98	3.41	10.81	2.24	9.89	95,917	408,137	7.12	9.82	1.58	4.31
	3	0.07	0.33	3,130	15,714	5.06	8.51	2.85	6.16	0.79	3.72	35,488	176,688	5.97	8.66	2.34	6.50
6	1	32.36	123.84	1,396,462	5,322,328	4.06	12.33	1.09	2.74	877.04 **(1)**	3600.00	33,612,267	140,644,420	3.77	11.00	0.03	0.33
	2	0.98	5.34	43,996	237,480	5.69	8.73	1.92	4.35	23.33	78.39	837,681	2,903,934	7.55	9.93	0.70	1.45
	3	0.82	4.63	36,149	205,950	3.78	8.40	2.93	9.25	17.93	80.63	708,896	3,463,208	4.29	6.39	1.71	3.17
8	1	48.76	237.14	1,840,429	10,039,399	2.76	8.77	3.60	14.29	2749.69 **(6)**	3600.00	86,141,854	137,849,126	3.77	7.75	0.48	2.04
	2	2.67	16.78	112,125	747,403	3.98	8.15	2.18	7.20	1178.06 **(3)**	3600.00	35,109,303	109,556,513	5.63	9.93	1.23	1.83
	3	4.34	31.56	152,596	1,058,180	1.95	4.76	2.78	7.73	544.29 **(1)**	3600.00	17,690,621	120,554,552	5.08	7.96	1.40	3.08

		n =40								*n* =50							
		CPU time		# of nodes		LB Gap		UB Gap		CPU time		# of nodes		LB Gap		UB Gap	
m	*w*	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.
2	1	0.01	0.03	283	897	2.00	5.00	0.00	0.00	0.03	0.14	976	6,204	1.30	3.00	0.00	0.00
	2	0.04	0.08	1,386	3,067	5.02	10.71	2.40	6.58	0.02	0.05	554	1,738	5.36	12.90	0.74	3.95
	3	0.04	0.17	1,639	8,004	3.75	7.97	2.81	6.03	0.05	0.14	2,011	6,037	4.73	11.30	2.03	3.56
4	1	11.76	99.72	421,261	3,509,020	2.70	9.50	0.00	0.00	14.81	51.42	503,061	1,719,003	0.85	2.00	0.00	0.00
	2	6.40	30.22	243,168	1,119,148	6.00	9.09	0.63	2.63	44.96	198.92	1,543,412	6,903,348	5.12	9.02	0.79	2.14
	3	4.30	27.22	175,171	1,135,865	6.15	8.64	0.66	1.36	7.95	25.31	288,802	919,564	4.24	7.71	0.90	1.81
6	1	682.81	2703.13	21,925,012	82,154,631	3.04	11.00	0.00	0.00	458.96	2498.53	12,760,873	69,861,017	1.73	3.00	0.00	0.00
	2	919.06 **(2)**	3600.00	31,373,231	135,276,346	5.79	9.09	0.74	1.75	1758.51 **(2)**	3600.00	47,055,836	100,160,719	7.51	10.42	0.62	1.62
	3	320.97	1182.66	10,504,905	41,112,652	5.25	8.79	1.29	4.95	1731.71 **(4)**	3600.00	47,982,999	104,944,678	5.12	9.65	0.54	1.44
8	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	2	1986.17 **(5)**	3600.00	56,691,660	111,271,281	7.07	9.26	0.81	2.28	-	-	-	-	-	-	-	-
	3	2913.70 **(7)**	3600.00	80,799,304	110,991,764	4.08	6.75	1.16	2.85	-	-	-	-	-	-	-	-

Table 6.2.5. (cont'd) Effect of *m* on the BB performance (*r* =1, *p* = 2) (OFJSW)

		n=60								*n*=70							
		CPU time		# of nodes		LB Gap		UB Gap		CPU time		# of nodes		LB Gap		UB Gap	
m	*w*	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.
2	1	0.01	0.06	387	2,167	1.10	4.00	0.00	0.00	0.02	0.09	683	2,751	0.60	2.00	0.00	0.00
	2	0.08	0.27	3,199	11,431	2.81	8.00	2.11	4.17	0.13	0.45	5,273	17,519	3.82	8.82	1.81	5.05
	3	0.05	0.20	2,190	7,807	2.81	5.78	2.30	5.31	0.06	0.19	2,069	6,950	3.10	7.14	0.87	3.79
4	1	8.97	31.78	271,624	977,645	1.40	3.00	0.00	0.00	8.20	38.58	236,023	1,146,168	0.80	2.50	0.00	0.00
	2	125.84	807.06	4,185,758	27,441,930	5.87	8.61	0.28	0.71	73.13	483.73	2,157,757	14,365,845	5.24	9.42	0.60	1.81
	3	80.90	361.25	2,584,951	11,979,150	4.79	7.71	0.66	1.75	165.67	1013.55	4,759,861	29,319,246	5.04	11.62	0.77	1.69
6	1	1102.09 **(1)**	3600.00	27,291,029	86,624,145	1.77	3.33	0.03	0.33	879.32	3261.41	20,443,914	74,694,778	1.10	4.33	0.00	0.00
	2	2578.84 **(7)**	3600.00	66,580,128	100,350,675	4.36	9.94	0.60	1.60	-	-	-	-	-	-	-	-
	3	1909.22 **(4)**	3600.00	49,770,090	112,076,808	4.07	6.36	0.48	1.09	2206.22 **(6)**	3600.00	48,918,015	87,078,753	3.73	5.08	0.40	1.42

Table 6.2.6. Effect of upper bound on BB performance (OFJSW)

					With upper bounds						Without upper bounds			
					CPU time		# of nodes		UB Gap		CPU time		# of nodes	
n	*m*	*r*	*p*	*w*	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.
20	3	1	1	1	1.12	4.47	61,905	236,514	0.42	2.90	16.35	77.72	1,001,145	4,470,727
				2	2.17	11.03	130,228	669,221	0.84	2.88	9.82	44.23	588,674	2,698,315
				3	7.35	56.92	332,563	2,384,084	0.62	3.59	20.49	155.86	932,638	6,446,176
			2	1	0.32	1.98	14,843	88,113	0.00	0.00	12.23	33.98	726,365	1,987,520
				2	0.03	0.05	907	2,336	3.30	6.58	0.93	2.25	60,037	140,979
				3	0.03	0.08	1,409	4,602	2.47	7.17	2.76	10.67	179,541	710,483
		2	1	1	3.77	13.22	152,248	546,482	7.47	15.24	29.05	181.25	1,211,163	7,506,147
				2	1.42	11.61	61,085	510,281	10.33	17.00	6.47	59.08	273,510	2,492,294
				3	1.10	4.77	43,538	194,579	7.81	15.79	4.89	35.41	206,990	1,488,643
			2	1	0.08	0.19	3,680	9,839	0.07	0.67	2.87	5.38	197,078	382,918
				2	0.04	0.22	1,953	11,715	5.14	13.70	0.69	2.95	45,580	206,012
				3	0.02	0.06	751	3,968	3.62	6.21	0.31	0.98	20,650	65,960
30	2	1	1	1	0.08	0.47	3,437	22,349	0.00	0.00	-	-	-	-
				2	0.10	0.78	5,465	41,529	0.34	0.88	2776.05 **(4)**	3600.00	174,508,925	234,368,295
				3	0.15	0.52	8,170	28,359	1.64	6.75	939.92 **(1)**	3600.00	59,288,373	225,603,339
			2	1	0.01	0.03	277	1,033	0.00	0.00	26.19	87.94	1,723,301	5,494,243
				2	0.02	0.05	862	1,713	3.34	6.33	4.46	13.70	296,679	919,803
				3	0.02	0.05	701	1,982	2.02	4.29	3.38	10.11	231,784	701,722
		2	1	1	0.05	0.34	2,490	17,059	0.00	0.00	2201.54 **(4)**	3600.00	143,060,735	263,158,453
				2	0.35	1.08	17,938	54,897	3.55	10.34	78.18	264.05	4,863,024	17,179,723
				3	0.23	0.63	11,401	32,706	8.73	18.36	9.75	26.59	598,357	1,567,112
			2	1	0.01	0.03	368	1,230	0.00	0.00	13.43	106.61	773,876	5,864,370
				2	0.01	0.03	415	1,254	7.40	20.75	0.49	1.42	31,251	99,287
				3	0.02	0.05	734	1,869	10.62	17.73	0.54	2.84	35,927	196,604

Table 6.2.7. Effect of lower bound on BB performance (OFJSW)

					With LB						Without LB			
					CPU time		# of nodes		LB Gap		CPU time		# of nodes	
n	*m*	*r*	*p*	*w*	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.
20	3	1	1	1	1.12	4.47	61,905	236,514	2.20	4.03	1.45	4.48	81,603	237,571
				2	2.17	11.03	130,228	669,221	2.02	6.08	2.32	10.94	137,960	669,340
				3	7.35	56.92	332,563	2,384,084	1.71	3.95	6.05	43.25	332,762	2,385,116
			2	1	0.32	1.98	14,843	88,113	5.13	14.00	0.33	2.00	15,006	88,906
				2	0.03	0.05	907	2,336	6.01	10.53	0.03	0.06	909	2,336
				3	0.03	0.08	1,409	4,602	6.13	11.69	0.03	0.09	1,431	4,635
		2	1	1	3.77	13.22	152,248	546,482	3.12	8.96	3.06	11.13	152,277	546,484
				2	1.42	11.61	61,085	510,281	1.98	5.13	1.11	8.94	61,793	510,282
				3	1.10	4.77	43,538	194,579	1.58	5.83	0.99	4.11	48,400	194,590
			2	1	0.08	0.19	3,680	9,839	1.80	4.00	0.09	0.22	4,422	11,206
				2	0.04	0.22	1,953	11,715	6.31	10.75	0.05	0.22	1,958	11,715
				3	0.02	0.06	751	3,968	4.65	13.00	0.02	0.08	766	3,970
30	2	1	1	1	0.08	0.47	3,437	22,349	2.30	5.00	0.13	0.66	6,794	34,923
				2	0.10	0.78	5,465	41,529	2.68	6.14	0.11	0.77	5,784	41,544
				3	0.15	0.52	8,170	28,359	2.86	5.68	0.15	0.52	8,198	28,362
			2	1	0.01	0.03	277	1,033	1.30	4.00	0.02	0.03	544	1,070
				2	0.02	0.05	862	1,713	3.46	7.04	0.02	0.03	939	1,713
				3	0.02	0.05	701	1,982	3.78	7.60	0.02	0.06	792	2,597
		2	1	1	0.05	0.34	2,490	17,059	2.20	7.00	0.07	0.41	3,326	19,644
				2	0.35	1.08	17,938	54,897	3.30	5.88	0.37	1.19	19,856	64,014
				3	0.23	0.63	11,401	32,706	2.81	4.91	0.23	0.61	11,670	32,857
			2	1	0.01	0.03	368	1,230	1.60	4.00	0.02	0.03	573	1,378
				2	0.01	0.03	415	1,254	6.31	11.67	0.01	0.03	424	1,254
				3	0.02	0.05	734	1,869	4.93	13.33	0.02	0.05	750	1,869

The effect of the reduction mechanisms and the precedence relations on the performance is also examined. In Table 6.2.8, we report the CPU times, the number of nodes, and the average number of jobs after reduction when we incorporate the properties of the optimal solution into the BB algorithm. We compare these values with the cases where each one of the properties is not used. Note that, Theorem 3.2.3 (of which application requires a longest path solution) is not employed in our branch and bound procedure, since our initial experimentation revealed that employment of this property is not efficient. The effort spent for solving n longest path problems (one for each job) outweighs the savings obtained through reductions and partial solution eliminations.

The table reveals that our reduction and precedence mechanisms are quite effective on the branch and bound algorithm. The CPU times and the number of nodes increase significantly when any one of the employed properties is not used. Actually, when we remove all properties from the branch and bound procedure, the CPU times and the number of nodes nearly double in most of the instances.

As another measure, the node at which optimal solution is found and the total number of nodes are given in Table 6.2.9, for n = 40 and 50. One can easily observe that the optimal solution is mostly found before reaching half of the total search. This indicates that the solutions found at termination limit are likely to be optimal, and a truncated version of our branch and bound algorithm can be a good alternative to optimal solutions for the larger-size problems.

Finally, we investigate the effect of a working time limit that is defined as a function of processing times. To see this effect, we set the working time limit equal to a factor (α) times the total processing times of all jobs divided by the number of machines (instead of a fixed working time limit of 50).

Table 6.2.8. Effect of reduction mechanisms on BB performance (OFJSW)

					With reduction & precedence					Without Theorem 3.2.1				Without Theorem 3.2.2				Without Theorems 3.2.4 and 3.2.5			
					CPU time		# of nodes		avg. # of	CPU time		# of nodes		CPU time		# of nodes		CPU time		# of nodes	
n	*m*	*r*	*p*	*w*	Avg.	Max.	Avg.	Max.	rem. jobs	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.
20	3	1	1	1	1.12	4.47	61,905	236,514	20	2.46	10.06	110,525	436,705	1.12	4.47	61,905	236,514	1.34	5.36	74,285	283,817
				2	2.17	11.03	130,228	669,221	20	5.10	35.57	226,095	1,550,702	2.17	11.03	130,228	669,221	2.60	13.24	156,274	803,065
				3	7.35	56.92	332,563	2,384,084	20	14.22	105.70	537,301	3,738,742	7.35	56.92	332,563	2,384,084	8.82	68.31	399,076	2,860,901
			2	1	0.32	1.98	14,843	88,113	20	0.90	3.17	36,873	130,196	0.32	1.98	14,843	88,113	0.39	2.38	17,811	105,736
				2	0.03	0.05	907	2,336	20	0.05	0.09	2,156	4,280	0.03	0.05	907	2,336	0.03	0.06	1,088	2,803
				3	0.03	0.08	1,409	4,602	20	0.08	0.24	3,561	12,038	0.03	0.08	1,409	4,602	0.04	0.09	1,690	5,522
		2	1	1	3.77	13.22	152,248	546,482	20	4.03	14.35	152,906	538,089	3.77	13.22	152,248	546,482	4.52	15.86	182,697	655,778
				2	1.42	11.61	61,085	510,281	20	6.12	57.77	231,409	2,178,575	1.42	11.61	61,085	510,281	1.70	13.93	73,302	612,337
				3	1.10	4.77	43,538	194,579	19	1.88	9.89	73,254	386,934	1.41	7.42	54,941	290,200	1.32	5.72	52,245	233,495
			2	1	0.08	0.19	3,680	9,839	20	0.75	3.25	30,992	134,030	0.08	0.19	3,680	9,839	0.09	0.22	4,416	11,807
				2	0.04	0.22	1,953	11,715	19	0.11	0.72	5,241	37,466	0.08	0.54	3,931	28,099	0.05	0.26	2,343	14,058
				3	0.02	0.06	751	3,968	19	0.04	0.17	1,541	7,752	0.03	0.13	1,156	5,814	0.02	0.08	901	4,762
30	2	1	1	1	0.08	0.47	3,437	22,349	30	0.09	0.55	3,792	25,341	0.08	0.47	3,437	22,349	0.09	0.56	4,124	26,819
				2	0.10	0.78	5,465	41,529	29	0.10	0.74	5,598	41,110	0.08	0.55	4,199	30,833	0.13	0.94	6,558	49,835
				3	0.15	0.52	8,170	28,359	30	0.20	0.64	10,749	37,027	0.15	0.52	8,170	28,359	0.18	0.62	9,804	34,031
			2	1	0.01	0.03	277	1,033	30	0.02	0.10	823	3,856	0.01	0.03	277	1,033	0.01	0.04	333	1,240
				2	0.02	0.05	862	1,713	27	0.03	0.06	1,387	2,704	0.03	0.05	1,040	2,028	0.03	0.06	1,035	2,056
				3	0.02	0.05	701	1,982	26	0.03	0.10	1,302	4,918	0.02	0.08	977	3,688	0.02	0.06	841	2,378
		2	1	1	0.05	0.34	2,490	17,059	30	0.08	0.59	3,752	28,579	0.05	0.34	2,490	17,059	0.07	0.41	2,988	20,471
				2	0.35	1.08	17,938	54,897	27	0.72	2.18	37,986	119,895	0.54	1.63	28,490	89,921	0.42	1.29	21,525	65,876
				3	0.23	0.63	11,401	32,706	27	0.40	1.51	20,388	74,822	0.30	1.13	15,291	56,116	0.28	0.75	13,681	39,247
			2	1	0.01	0.03	368	1,230	30	0.02	0.06	882	2,704	0.01	0.03	368	1,230	0.01	0.04	441	1,476
				2	0.01	0.03	415	1,254	25	0.02	0.06	962	2,935	0.02	0.05	721	2,201	0.01	0.04	498	1,505
				3	0.02	0.05	734	1,869	22	0.04	0.09	1,690	3,914	0.03	0.07	1,268	2,936	0.03	0.06	881	2,243

Table 6.2.9. Optimal solution node vs. total number of nodes (OFJSW)

				m =2				*m* =3				*m* =4			
				solution node		# of nodes		solution node		# of nodes		solution node		# of nodes	
n	*r*	*p*	*w*	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.
40	1	1	1	2,524	7,446	2,524	7,446	131,369	537,121	131,369	537,121	3,386,016	33,802,779	59,914,293	149,620,847
			2	3,422	17,947	20,766	160,614	2,082,536	19,982,092	3,894,037	19,982,092	-	-	-	-
			3	10,354	86,870	14,339	94,792	311,323	1,748,471	10,656,156	82,029,375	-	-	-	-
		2	1	283	897	283	897	13,061	36,880	13,061	36,880	421,261	3,509,020	421,261	3,509,020
			2	881	2,650	1,386	3,067	16,989	72,336	26,221	72,336	58,984	415,846	243,168	1,119,148
			3	1,158	7,850	1,639	8,004	12,096	39,578	16,697	67,787	67,645	384,130	175,171	1,135,865
	2	1	1	2,581	9,099	2,581	9,099	8,322,124	79,881,331	13,314,187	129,205,355	30,754,274	79,944,378	79,957,005	130,434,318
			2	10,068	46,819	32,930	133,484	4,857,963	29,610,890	25,732,711	144,011,865	23,760,122	91,946,094	92,314,270	129,202,858
			3	15,249	70,867	54,772	209,455	18,935,580	107,133,830	33,919,851	140,469,090	1,455,832	11,352,088	105,064,084	147,289,758
		2	1	2,194	5,452	2,194	5,452	8,871	29,665	8,871	29,665	141,709	794,474	141,709	794,474
			2	236	1,001	838	4,435	15,501	121,437	26,450	171,383	32,455	91,449	67,835	281,170
			3	809	4,580	1,336	6,437	13,186	46,543	22,566	67,021	82,633	252,695	243,521	872,498
50	1	1	1	2,540	7,079	2,540	7,079	113,939	843,035	113,939	843,035	17,678,807	82,214,849	40,755,533	123,408,814
			2	60,163	580,535	82,142	725,648	7,481,527	39,482,886	12,820,574	39,482,886	-	-	-	-
			3	6,247	44,595	17,244	115,344	3,095,416	16,517,042	11,493,919	49,969,862	-	-	-	-
		2	1	976	6,204	976	6,204	7,179	30,927	7,179	30,927	503,061	1,719,003	503,061	1,719,003
			2	302	1,218	554	1,738	2,146	13,538	22,576	77,077	408,078	3,740,471	1,543,412	6,903,348
			3	544	2,779	2,011	6,037	10,176	41,921	22,931	81,819	112,318	310,603	288,802	919,564
	2	1	1	4,008	20,094	4,008	20,094	450,065	2,185,537	450,065	2,185,537	15,194,498	57,184,212	79,566,263	113,804,538
			2	32,382	125,251	105,903	331,529	7,980,230	42,392,985	25,968,710	132,816,308	-	-	-	-
			3	35,901	125,492	91,900	370,013	12,910,496	88,443,926	41,736,065	133,008,494	-	-	-	-
		2	1	631	3,636	631	3,636	23,381	172,457	23,381	172,457	256,636	1,160,066	256,636	1,160,066
			2	589	3,034	1,707	3,396	41,200	247,636	69,934	393,340	81,399	375,145	306,621	1,058,260
			3	863	7,016	3,453	28,351	21,498	75,321	36,893	154,758	157,708	867,913	1,182,937	9,003,661

The formula for the new working time limit then becomes:

$$T = a \left\lfloor \sum_j p_j \Big/ m \right\rfloor .$$

If α=1, m = 1, and there are no job overlaps, then all jobs will be processed. In general, we expect that α of the jobs would be processed in case of no job overlaps. We use α=1, 0.5 and 0.33. We use the hardest ready time and processing time combination, i.e. r = 2 and p = 1, on four moderate size problems with 40 and 50 jobs, 2 and 3 machines. Note that in this case, T is 150 on the average when α=1, n = 40, and m = 2, since we have uniform processing times between 5 and 10. Similarly, when α=0.33, average T is 50 for the same problem combination. Table 6.2.10 presents related performance measures.

One can easily observe that, when the working time limit is a function of processing times, the solution times as well as the bound deviations increase as α increases. This result is expected since, for a fixed problem combination, increasing the factor value means increasing the working time limit, thereby resulting in higher number of processed jobs.

Note that the CPU times and bound performances of T=50 case are very similar to those of α=0.33 case for n = 40, m = 2 combination, since their average working time limits are identical.

All computational results reveal that the algorithm returns optimal solutions for problem instances with up to 100 jobs within 1 hour of CPU time, and is likely to perform well for larger instances. The results also show that the efficiency of our BB algorithm is sensitive to the processing time and weight distribution, where higher variability brings higher computational times.

			n=40								n=50							
			CPU time		# of nodes		%LB Gap		%UB Gap		CPU time		# of nodes		%LB Gap		%UB Gap	
T	m	w	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.
50	2	1	0.07	0.22	2,581	9,099	2.40	4.00	0.00	0.00	0.11	0.52	4,008	20,094	1.90	5.00	0.00	0.00
		2	0.75	3.11	32,930	133,484	2.43	4.27	3.80	9.40	2.40	7.11	105,903	331,529	4.12	7.75	4.67	8.80
		3	1.16	4.14	54,772	209,455	3.54	6.48	5.17	8.80	2.02	8.39	91,900	370,013	2.78	5.84	3.78	6.78
	3	1	370.78 **(1)**	3600.00	13,314,187	129,205,355	2.45	5.67	1.06	6.38	13.31	67.66	450,065	2,185,537	2.20	3.33	0.00	0.00
		2	640.87 **(1)**	3600.00	25,732,711	144,011,865	4.83	9.04	4.00	9.04	698.90 **(1)**	3600.00	25,968,710	132,816,308	4.47	11.23	3.15	5.24
		3	884.82 **(1)**	3600.00	33,919,851	140,469,090	4.35	8.25	9.86	21.32	1122.13 **(2)**	3600.00	41,736,065	133,008,494	3.21	6.74	4.31	10.91
$0.33\left\lfloor\frac{\sum p_j}{m}\right\rfloor$	2	1	0.10	0.61	2,602	16,068	2.25	6.60	0.00	0.00	1.18	4.30	25,893	89,194	2.18	4.84	0.00	0.00
		2	0.89	3.47	22,074	89,732	2.34	8.06	3.38	7.83	18.66	132.39	436,393	2,995,168	3.75	6.94	5.58	11.11
		3	1.62	5.75	45,124	160,848	2.07	6.48	5.14	8.37	21.02	109.22	551,154	2,857,590	3.42	5.89	5.34	9.13
	3	1	1.00	4.33	19,585	84,049	3.63	7.84	0.00	0.00	2.35	9.95	48,344	206,935	2.65	4.65	0.00	0.00
		2	6.07	26.78	150,111	646,516	4.69	7.87	1.15	2.36	480.76 **(1)**	3600.00	11,654,536	87,486,503	4.90	7.45	2.56	4.94
		3	5.39	24.05	134,959	555,780	4.57	6.61	3.77	10.30	129.19	593.14	3,117,944	14,241,076	3.39	5.56	2.17	4.54
$0.5\left\lfloor\frac{\sum p_j}{m}\right\rfloor$	2	1	12.60	58.63	278,618	1,241,373	6.83	12.84	1.52	6.04	92.68	413.45	2,016,607	8,718,697	5.86	9.76	3.19	11.59
		2	2.75	7.72	65,014	194,266	5.03	8.11	8.66	13.24	75.43	622.42	1,709,820	14,211,153	4.91	11.86	11.10	16.67
		3	23.56	99.34	449,409	1,607,869	2.67	6.25	10.09	20.08	160.85	1009.08	3,639,427	22,371,743	3.99	7.13	10.71	13.78
	3	1	428.57	3204.64	6,132,461	47,006,820	2.80	5.67	0.64	6.38	1844.37 **(3)**	3600.00	35,772,713	73,351,501	2.50	9.04	0.88	4.05
		2	776.95 **(1)**	3600.00	18,106,337	84,261,155	4.27	9.04	3.86	9.04	3001.60 **(7)**	3600.00	63,740,200	79,930,935	4.92	7.24	4.60	7.69
		3	903.68	2578.16	21,181,940	61,427,844	4.08	9.94	9.04	20.90	3271.33 **(8)**	3600.00	69,330,207	79,226,667	3.71	5.93	6.50	14.66
$\left\lfloor\frac{\sum p_j}{m}\right\rfloor$	2	1	113.12	995.81	2,181,135	19,130,018	7.25	11.25	5.57	12.18	166.07	881.49	2,923,002	15,658,492	5.72	10.05	8.25	15.48
		2	3.58	9.38	73,698	195,647	4.94	8.11	12.02	14.97	100.67	881.49	1,791,551	15,658,492	4.54	8.76	13.83	22.28
		3	14.79	72.28	319,798	1,637,262	2.61	6.25	12.01	21.27	288.97	1686.83	5,060,238	28,994,282	3.69	7.89	14.16	20.15
	3	1	2921.46 **(8)**	3600.00	52,303,869	68,538,677	4.28	13.21	8.39	12.99	2784.49 **(7)**	3600.00	45,867,734	63,500,665	7.23	12.75	4.21	8.24
		2	1639.15 **(3)**	3600.00	31,325,335	69,267,153	4.18	10.99	12.14	21.69	-	-	-	-	-	-	-	-
		3	2182.57 **(5)**	3600.00	28,076,197	48,187,978	1.51	3.19	13.33	21.32	-	-	-	-	-	-	-	-

6.2.2. Results for the OFJSS Problem

For the OFJSS problem, initial experimentation is performed to observe the performance of three sequential upper bounds in Phase 1. The third upper bound UBS_3 based on MCNF solution is coded in MS Visual C++, using ILOG CPLEX 9.0 with Concert Technology. Based on our initial experimentation results, we use only UBS_2 in Phase 1. This is due to the fact that UBS_1 is often dominated by UBS_2 and UBS_3, and the computation times get large when UBS_3 is employed. The results of partial initial runs with moderate problem sizes are provided in Table 6.2.11.

In analyzing the results of the OFJSS and OFJSG problems, we follow a similar approach. The performance of our lower and upper bounds is investigated first. Table 6.2.12 and Table 6.2.13 report the average and maximum lower and upper bound deviations as a percentage of optimal solutions for r =1 and r =2, respectively. Recall that, we compute lower and upper bounds at the leaf nodes (lowest level nodes) of Phase 1 of the branch and bound procedure. Therefore, the lower bound value used in computing the deviation (LB Gap) is the best lower bound found in Phase 1, while the upper bound value used for computing the deviation (UB Gap) is the Phase 1 upper bound value of the node with first jobs of the optimal solution. The tables also present the number of times the best lower bound found in Phase 1 finds the optimal solution. An empty entry in the tables indicates that the algorithm could not return an optimal solution within one-hour termination limit for any of the 10 problem instances.

One important observation from the tables is that the best lower bound value of Phase 1 yields the optimal solution in most of the problem instances. In particular, the average percentage of optimal solutions caught by the lower bound is 80% for both ready time combinations. This implies that our lower bounding procedure is exceedingly powerful, and in many cases it catches the optimal solution once the first jobs are determined.

Table 6.2.11. Results of some initial runs for the OFJSS problem

			n =40						*n* =50					
			CPU time		# of nodes		UB Gap %		CPU time		# of nodes		UB Gap %	
UB	*m*	*w*	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.
	2	1	0.98	3.88	21,433	80,875	13.90	21.01	3.16	10.94	55,531	196,005	12.15	17.88
		2	1.17	2.75	26,749	64,740	29.59	65.91	8.82	31.56	156,218	540,855	27.87	47.16
1		3	0.78	2.22	18,226	54,035	20.59	33.79	6.61	33.75	114,415	588,890	21.16	31.61
	3	1	58.33	177.92	1,183,307	3,464,317	11.10	18.78	591.13 **(1)**	3600.00	9,370,425	59,860,327	14.19	22.06
		2	45.75	128.63	917,399	2,358,654	15.40	23.32	909.92 **(2)**	3600.00	14,821,383	62,372,576	19.50	28.43
		3	29.71	124.31	603,559	2,514,693	15.75	30.07	202.47	771.28	3,245,117	11,665,634	15.05	24.66
	2	1	0.22	0.95	5,001	20,183	0.97	6.90	1.00	5.13	18,690	92,268	0.83	5.00
		2	0.51	1.45	12,069	34,402	15.39	65.91	3.36	10.84	60,460	203,444	5.58	47.16
2		3	0.45	2.49	9,265	46,445	1.69	8.77	1.48	8.58	29,665	174,275	1.45	11.05
	3	1	56.22	173.16	1,029,191	3,474,927	9.75	16.67	540.54	3351.11	4,092,988	12,323,658	5.62	13.24
		2	44.17	124.92	790,039	2,241,258	9.16	17.95	748.38 **(1)**	3600.00	10,904,570	53,449,052	6.35	11.85
		3	35.44	136.39	620,946	2,505,819	14.40	30.56	225.01	745.80	3,221,592	10,911,642	9.12	16.33
	2	1	1.40	3.09	6,219	18,244	6.37	28.28	3.46	8.30	20,551	102,095	3.42	29.80
		2	1.42	2.02	10,729	28,284	16.64	65.91	5.81	17.34	65,596	225,743	18.05	50.00
3		3	1.48	2.80	9,097	36,900	2.16	15.10	3.57	10.92	32,374	184,731	6.94	33.74
	3	1	98.89	233.67	1,178,773	4,195,277	28.98	41.44	627.26 **(1)**	3600.00	7,780,893	47,452,004	43.07	56.85
		2	82.17	185.22	909,694	2,269,681	28.30	43.16	985.06 **(2)**	3600.00	12,723,490	51,426,465	36.04	49.51
		3	59.47	179.77	578,073	2,427,828	25.23	47.97	283.17	797.52	2,901,325	9,795,497	28.49	35.89
	2	1	0.20	0.98	4,989	20,175	0.97	6.90	0.97	4.69	18,669	92,268	0.83	5.00
1		2	0.49	1.44	12,063	34,400	14.94	65.91	3.05	9.56	60,438	203,444	5.58	47.16
+		3	0.47	2.83	9,256	46,433	1.69	8.77	1.43	8.30	29,656	174,275	1.27	9.30
2	3	1	61.90	175.22	1,032,397	3,435,690	9.75	16.67	561.97 **(1)**	3600.00	7,205,720	43,458,864	5.62	13.24
		2	45.52	141.08	783,247	2,240,334	9.16	17.95	742.24 **(1)**	3600.00	10,787,617	52,311,274	6.35	11.85
		3	44.56	143.28	616,749	2,504,936	14.40	30.56	246.30	731.30	3,128,615	10,649,645	9.12	16.33
	2	1	1.14	2.02	6,212	18,244	3.66	14.39	3.31	8.19	20,545	102,095	1.63	11.92
1		2	1.26	1.94	10,727	28,284	15.81	65.91	5.56	17.44	65,586	225,743	14.19	47.16
+		3	1.25	2.38	9,094	36,900	2.16	15.10	3.28	10.97	32,365	184,731	4.23	23.55
3	3	1	74.02	185.47	1,125,270	3,440,888	11.10	18.78	622.77 **(1)**	3600.00	7,536,915	45,170,634	14.19	22.06
		2	67.81	178.33	817,774	2,268,716	14.88	23.32	965.79 **(2)**	3600.00	12,595,156	52,168,435	19.50	28.43
		3	47.09	159.30	576,632	2,427,119	15.75	30.07	267.71	784.03	2,831,217	9,157,929	15.05	24.66
	2	1	0.95	1.24	5,001	20,183	0.97	6.90	2.78	6.55	18,690	92,268	0.83	5.00
2		2	0.96	1.98	12,069	34,402	15.39	65.91	4.82	12.76	60,460	203,444	5.58	47.16
+		3	1.30	2.72	9,265	46,445	1.69	8.77	2.35	7.21	29,665	174,275	1.45	11.05
3	3	1	98.89	233.67	1,029,191	3,474,927	9.75	16.67	711.42 **(1)**	3600.00	4,092,988	12,323,658	5.62	13.24
		2	82.17	185.22	790,039	2,241,258	9.16	17.95	852.33 **(1)**	3600.00	10,904,570	53,449,052	6.35	11.85
		3	59.47	179.77	620,946	2,505,819	14.40	30.56	345.29	814.25	3,221,592	10,911,642	9.12	16.33
1		1	1.05	1.94	4,187	14,531	0.97	6.90	2.98	6.67	15,508	78,128	0.77	4.37
+	2	2	1.26	1.98	9,711	28,284	14.72	65.91	5.90	13.86	54,067	198,522	5.58	47.16
2		3	1.30	2.72	7,994	36,900	1.00	5.64	2.85	7.89	25,324	133,932	0.75	4.07
+		1	90.55	202.92	1,031,666	3,429,648	9.75	16.67	726.40 **(1)**	3600.00	7,797,056	49,394,322	5.62	13.24
3	3	2	82.00	159.89	778,414	2,240,242	9.16	17.95	874.15 **(1)**	3600.00	11,672,878	61,243,229	6.35	11.85
		3	67.29	160.58	601,727	2,422,169	9.88	16.10	369.13	842.00	2,863,265	9,999,130	7.78	10.89

Table 6.2.12. LB and UB deviations at root node for r =1 (OFJSS)

			m=2					m=3					m=4				
			LB Gap (%)		UB Gap (%)			LB Gap		UB Gap			LB Gap		UB Gap		
n	p	w	Avg.	Max.	Avg.	Max.	opt.	Avg.	Max.	Avg.	Max.	opt.	Avg.	Max.	Avg.	Max.	opt.
20	1	1	0.00	0.00	5.14	20.22	10	0.00	0.00	20.33	56.00	10	0.00	0.00	22.96	37.40	10
		2	0.37	1.96	2.13	12.50	8	0.00	0.00	12.90	18.85	10	0.40	3.25	34.26	50.75	8
		3	0.00	0.00	13.12	34.67	10	0.75	4.75	17.93	34.44	7	0.00	0.00	29.91	48.55	10
	2	1	0.00	0.00	23.60	83.57	10	0.60	4.66	8.72	67.98	8	0.14	1.35	3.52	10.24	9
		2	0.00	0.00	11.62	52.94	10	0.11	1.10	4.88	24.39	9	0.80	5.00	10.28	27.83	7
		3	0.00	0.00	8.01	75.20	10	0.00	0.00	6.14	24.32	10	0.83	3.25	10.68	26.90	6
30	1	1	0.15	1.47	1.76	9.90	9	0.00	0.00	10.35	20.13	10	0.00	0.00	22.45	31.84	10
		2	0.00	0.00	5.73	40.63	10	0.23	1.16	10.56	19.88	8	0.54	3.85	25.47	39.74	8
		3	0.19	1.92	8.50	52.30	9	0.40	1.74	13.59	22.15	7	0.48	3.00	26.67	46.10	7
	2	1	0.00	0.00	0.00	0.00	10	0.15	1.15	0.46	3.44	8	0.00	0.00	0.33	2.11	10
		2	0.12	1.16	8.23	56.10	9	0.98	5.71	5.32	12.38	7	0.75	3.05	6.04	13.48	6
		3	0.32	1.71	0.81	4.71	7	0.49	2.50	6.52	13.68	7	1.19	4.97	7.92	13.56	3
40	1	1	0.00	0.00	0.97	6.90	10	0.22	1.08	9.75	16.67	7	0.27	1.33	18.62	29.78	8
		2	0.00	0.00	15.39	65.91	10	0.72	2.54	9.16	17.95	6	0.39	1.91	17.26	29.39	7
		3	0.00	0.00	1.69	8.77	10	0.42	2.56	14.40	30.56	8	0.71	2.58	22.05	38.34	3
	2	1	0.00	0.00	26.18	95.63	10	0.04	0.36	0.33	1.82	9	0.03	0.28	7.46	73.41	9
		2	0.42	4.17	2.13	13.54	9	0.19	1.89	6.41	15.45	9	0.40	2.17	9.33	19.18	8
		3	0.42	2.28	6.16	38.92	8	0.46	3.00	4.96	13.86	7	0.71	1.71	9.45	17.16	2
50	1	1	0.00	0.00	0.83	5.00	10	0.10	0.50	5.62	13.24	8	0.04	0.35	18.79	25.79	9
		2	0.00	0.00	5.58	47.16	10	0.89	2.91	6.35	11.85	5	0.22	1.87	16.18	21.17	8
		3	0.00	0.00	1.45	11.05	10	0.82	2.41	9.12	16.33	5	0.20	0.74	19.21	27.92	7
	2	1	0.00	0.00	9.53	95.34	10	0.00	0.00	8.90	88.97	10	0.14	1.39	11.24	110.22	9
		2	0.00	0.00	7.40	71.25	10	1.14	4.76	5.22	10.29	4	0.32	2.58	10.68	21.21	8
		3	0.51	4.71	1.84	9.16	8	0.39	3.10	1.64	3.43	8	0.49	1.52	8.27	14.87	5
60	1	1	0.37	1.83	2.13	7.19	6	0.25	0.84	5.48	12.00	7	-	-	-	-	-
		2	0.21	2.05	1.19	4.10	9	0.81	2.51	5.50	10.24	4	-	-	-	-	-
		3	0.00	0.00	0.97	7.09	10	0.33	1.52	7.72	12.91	6	-	-	-	-	-
	2	1	0.00	0.00	11.62	116.23	10	0.00	0.00	0.04	0.35	10	0.00	0.00	0.03	0.26	10
		2	0.00	0.00	10.48	70.19	10	0.69	2.65	4.06	9.27	6	1.62	4.65	7.75	17.35	3
		3	0.04	0.38	8.33	73.62	9	0.58	2.96	4.46	10.19	7	0.50	1.80	10.08	28.92	4
70	1	1	0.00	0.00	16.38	88.55	10	0.47	2.04	3.76	7.53	6	-	-	-	-	-
		2	0.00	0.00	0.59	4.73	10	0.15	1.10	5.45	12.77	8	-	-	-	-	-
		3	0.00	0.00	7.71	63.97	10	0.42	1.48	9.99	12.07	4	-	-	-	-	-
	2	1	0.00	0.00	0.00	0.00	10	0.00	0.00	0.00	0.00	10	0.00	0.00	0.11	0.53	10
		2	0.37	1.75	1.11	4.39	7	0.51	1.52	4.69	13.86	5	1.00	2.25	5.63	8.56	5
		3	0.35	2.42	10.60	74.66	8	0.50	2.53	6.54	12.63	7	0.17	1.33	10.49	13.56	8
80	1	1	0.05	0.54	7.66	74.44	9	-	-	-	-	-	-	-	-	-	-
		2	0.20	1.48	15.65	72.82	8	-	-	-	-	-	-	-	-	-	-
		3	0.07	0.37	3.13	8.42	7	-	-	-	-	-	-	-	-	-	-
	2	1	0.00	0.00	0.05	0.51	10	0.00	0.00	10.71	107.09	10	0.00	0.00	0.18	1.57	10
		2	0.17	1.67	4.02	12.50	9	0.38	1.76	6.10	10.64	6	-	-	-	-	-
		3	0.44	2.58	9.03	74.49	8	0.93	3.15	5.24	12.64	5	-	-	-	-	-
90	1	1	0.00	0.00	19.14	94.25	10	-	-	-	-	-	-	-	-	-	-
		2	0.25	1.95	8.96	73.63	8	-	-	-	-	-	-	-	-	-	-
		3	0.05	0.27	2.46	8.03	8	-	-	-	-	-	-	-	-	-	-
	2	1	0.00	0.00	12.04	119.90	10	0.00	0.00	0.00	0.00	10	0.03	0.26	0.00	0.00	9
		2	0.28	2.78	10.75	86.00	9	1.04	4.14	6.18	18.52	5	-	-	-	-	-
		3	0.16	0.83	3.57	22.04	8	0.38	2.40	6.35	13.08	7	-	-	-	-	-
100	1	1	0.06	0.55	0.78	2.21	9	-	-	-	-	-	-	-	-	-	-
		2	0.29	2.87	17.52	87.20	9	-	-	-	-	-	-	-	-	-	-
		3	0.27	1.28	1.49	3.46	7	-	-	-	-	-	-	-	-	-	-
	2	1	0.00	0.00	0.00	0.00	10	0.03	0.34	0.03	0.34	9	0.00	0.00	0.00	0.00	10
		2	0.00	0.00	9.69	75.21	10	0.36	1.84	6.38	22.47	7	-	-	-	-	-
		3	0.00	0.00	1.31	8.84	10	0.94	2.49	7.30	15.03	3	-	-	-	-	-

Table 6.2.13. LB and UB deviations at root node for *r*=2(OFJSS)

			m =2					*m* =3					*m* =4				
			LB Gap (%)		UB Gap (%)			LB Gap		UB Gap			LB Gap		UB Gap		
n	*p*	*w*	Avg.	Max.	Avg.	Max.	opt.	Avg.	Max.	Avg.	Max.	opt.	Avg.	Max.	Avg.	Max.	opt.
20	1	1	0.00	0.00	16.27	57.14	10	0.00	0.00	17.29	28.97	10	0.00	0.00	21.50	33.90	10
		2	0.00	0.00	12.05	40.79	10	0.11	1.10	18.67	35.00	9	0.00	0.00	23.40	47.44	10
		3	0.79	7.86	1.82	10.74	9	0.23	2.31	25.19	56.51	9	0.00	0.00	31.13	46.78	10
	2	1	0.35	3.49	17.42	73.28	9	0.52	5.18	10.63	75.83	9	0.69	3.49	10.00	15.97	7
		2	0.00	0.00	17.54	67.92	10	0.00	0.00	13.17	36.36	10	0.71	7.14	9.89	23.88	9
		3	0.00	0.00	16.82	66.98	10	0.00	0.00	4.29	11.43	10	0.00	0.00	8.97	19.87	10
30	1	1	0.00	0.00	2.00	9.17	10	0.00	0.00	11.34	37.00	10	0.00	0.00	26.04	42.74	10
		2	0.00	0.00	9.09	28.45	10	0.00	0.00	19.93	47.66	10	0.08	0.82	23.33	40.16	9
		3	0.00	0.00	11.23	41.04	10	0.25	2.27	20.14	38.38	8	0.32	3.21	24.10	34.05	9
	2	1	0.00	0.00	22.94	92.95	10	0.00	0.00	1.60	9.88	10	0.23	1.97	2.63	7.95	8
		2	0.63	6.33	16.62	89.66	9	0.49	4.85	1.82	9.71	9	0.22	2.17	9.96	23.30	9
		3	0.37	3.65	10.15	47.00	9	0.31	3.08	10.58	57.89	9	0.27	2.48	8.71	27.34	8
40	1	1	0.00	0.00	3.03	18.92	10	0.59	4.05	9.29	18.12	8	0.17	1.00	19.23	33.93	8
		2	0.57	5.71	6.35	41.67	9	0.13	1.33	8.49	15.63	9	0.00	0.00	20.86	43.10	10
		3	0.00	0.00	6.86	48.84	10	0.13	1.31	20.46	36.87	9	0.25	1.51	26.76	35.91	8
	2	1	0.27	2.67	9.26	90.72	9	0.39	2.69	9.51	79.74	8	0.16	0.94	1.24	7.77	8
		2	0.00	0.00	13.78	45.68	10	0.46	2.00	4.78	16.00	7	0.46	3.45	8.02	26.90	8
		3	0.59	2.99	3.37	17.79	8	0.91	4.51	6.92	18.09	7	0.83	3.75	12.02	33.50	6
50	1	1	0.00	0.00	18.44	67.72	10	0.20	1.97	9.96	22.44	9	0.00	0.00	20.54	28.08	10
		2	0.16	1.61	19.75	60.61	9	0.53	2.97	11.95	19.62	8	1.44	13.43	11.97	18.97	8
		3	0.00	0.00	8.27	63.88	10	0.12	0.85	11.80	22.17	8	0.34	3.01	22.64	53.12	7
	2	1	0.00	0.00	0.00	0.00	10	0.00	0.00	8.78	87.39	10	0.09	0.85	1.14	5.81	9
		2	0.30	3.03	29.00	83.87	9	0.26	2.59	6.81	12.61	9	0.65	4.22	7.40	33.33	7
		3	0.00	0.00	14.83	70.05	10	0.27	2.69	4.42	11.93	9	0.31	1.13	11.49	18.75	7
60	1	1	0.00	0.00	1.90	7.83	10	0.39	3.87	9.05	17.82	9	0.00	0.00	22.86	34.93	10
		2	0.00	0.00	3.45	11.29	10	0.22	1.64	11.30	19.19	8	-	-	-	-	-
		3	0.85	5.54	3.49	20.37	8	0.49	1.83	12.85	24.42	6	0.12	1.05	25.90	35.57	8
	2	1	0.00	0.00	0.05	0.54	10	0.18	1.80	0.33	1.80	9	0.34	1.43	0.62	1.98	7
		2	0.00	0.00	14.54	71.95	10	0.39	1.72	2.98	9.56	7	0.81	5.42	8.15	13.82	7
		3	0.18	1.14	22.74	76.86	8	0.44	2.66	7.94	58.38	7	0.63	2.93	10.56	26.51	6
70	1	1	0.00	0.00	1.01	6.76	10	0.38	3.30	9.71	19.43	8	-	-	-	-	-
		2	0.00	0.00	0.87	5.13	10	0.56	2.33	6.83	15.71	6	-	-	-	-	-
		3	0.04	0.36	5.34	42.40	9	0.73	5.13	9.93	22.08	6	-	-	-	-	-
	2	1	0.00	0.00	10.00	99.45	10	0.00	0.00	0.00	0.00	10	0.00	0.00	0.62	3.43	10
		2	0.00	0.00	14.09	76.67	10	0.24	1.60	4.02	11.90	8	0.74	2.74	7.52	16.18	5
		3	0.00	0.00	19.51	79.47	10	0.35	2.30	6.21	14.43	7	0.58	2.46	4.72	12.22	6
80	1	1	0.27	1.94	14.04	65.52	8	0.24	0.98	6.19	11.76	7	-	-	-	-	-
		2	0.06	0.61	1.60	7.48	9	0.09	0.44	11.13	18.27	8	-	-	-	-	-
		3	0.23	1.86	3.66	17.37	8	0.06	0.26	15.41	32.52	5	-	-	-	-	-
	2	1	0.00	0.00	11.56	115.59	10	0.07	0.72	0.04	0.36	9	0.03	0.28	0.47	1.69	9
		2	0.00	0.00	17.86	102.50	10	0.09	0.90	5.45	12.50	9	0.12	1.21	6.63	13.21	8
		3	0.18	1.17	1.54	10.31	8	0.49	1.49	6.43	13.08	5	1.00	3.65	10.78	13.96	5
90	1	1	0.07	0.65	1.16	9.09	9	0.29	1.52	8.37	12.45	10	-	-	-	-	-
		2	0.31	1.82	4.30	14.55	8	-	-	-	-	-	-	-	-	-	-
		3	0.17	1.37	3.62	11.90	8	-	-	-	-	-	-	-	-	-	-
	2	1	0.00	0.00	21.64	116.39	10	0.00	0.00	0.00	0.00	10	0.00	0.00	0.00	0.00	10
		2	0.38	1.96	7.87	71.00	8	0.50	3.01	6.01	22.63	8	-	-	-	-	-
		3	0.00	0.00	0.71	4.97	10	0.57	3.44	4.90	14.07	6	-	-	-	-	-
100	1	1	0.00	0.00	8.06	73.29	10	-	-	-	-	-	-	-	-	-	-
		2	0.06	0.56	1.66	9.38	9	-	-	-	-	-	-	-	-	-	-
		3	0.00	0.00	7.68	70.53	10	-	-	-	-	-	-	-	-	-	-
	2	1	0.00	0.00	0.00	0.00	10	0.00	0.00	0.07	0.35	10	0.03	0.26	0.33	2.20	9
		2	0.00	0.00	9.42	87.76	10	0.44	1.50	6.37	12.95	7	-	-	-	-	-
		3	0.46	3.33	4.00	9.20	7	0.19	0.88	5.55	17.85	6	-	-	-	-	-

Another evidence of this strength is apparent by the percentage deviations. Namely, the lower bounding procedure yields average respective deviations of no more than 2% and 1% from the optimal solutions for the r = 1 and r = 2 cases. Although the upper bound is relatively weaker, the average respective deviations do not exceed 35% and 32% for the r = 1 and r = 2 cases. There is no significant effect of the number of jobs or machines on the performance of the upper bound.

Next, we present the results of the computational runs associated with the branch and bound algorithm. In Tables 6.2.14 and 6.2.15, the average and maximum CPU times (in seconds) and the average and maximum number of nodes visited are presented for r =1 and r =2, respectively. The numbers in parentheses are the number of instances that cannot be solved within our termination limit of one hour.

As opposed to the case in the OFJSW problem, it is observed from the two tables that r = 2 case seems easier than r = 1 case in terms of solution times, number of nodes and number of solved problem instances. Actually, this result is not revealed by Tables 6.2.12 and 6.2.13, as the upper and lower bound performances of the two ready time cases are similar. The difference between the difficulties of the two ready time cases may be due to the increased number of alternatives in Phase 1 with r = 1 case. Recall that the jobs are evenly distributed in the [0,200] interval in the r = 1 case. Hence, reducing the number of candidate Phase 1 nodes with developed dominance conditions that are mainly based on job overlaps (see Chapter 4), gets harder in this case. In contrast, r = 2 case that has many more overlapping jobs in peak periods offers more potential reduction opportunities in Phase 1. Similar to the case in the OFJSW problem, when the processing times are concerned, p = 1 (low variability in processing times) combination reduces the efficiency. When the processing times are larger, the number of alternative machines gets fewer, which in turn results in fewer number of node evaluations. When $w_j = p_j$, the problem also becomes easier than the other two combinations for weights. This is due to the fact that, with higher weight variability, a small change in the solution is likely to affect the objective function value considerably, hence the best solution is updated more frequently, and the solution times increase.

Table 6.2.14. Branch and Bound performance for r =1 (OFJSS)

			m =2				m =3				m =4			
			CPU time		# of nodes		CPU time		# of nodes		CPU time		# of nodes	
n	p	w	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.
20	1	1	0.02	0.08	712	2,977	0.10	0.23	2,329	5,254	0.66	3.58	12,384	59,539
		2	0.01	0.03	337	891	0.15	0.36	4,439	12,234	1.56	3.52	31,514	79,899
		3	0.01	0.02	320	521	0.12	0.38	3,288	8,096	0.80	4.27	15,963	85,115
	2	1	0.00	0.02	189	238	0.04	0.06	1,586	2,657	0.17	0.33	7,028	10,465
		2	0.01	0.02	210	374	0.04	0.06	1,695	2,350	0.32	0.67	9,830	15,726
		3	0.00	0.02	196	292	0.08	0.27	2,823	9,018	0.37	1.99	11,270	46,897
30	1	1	0.06	0.30	1,742	8,346	1.61	5.27	36,797	114,914	145.73	691.59	2,613,038	13,160,703
		2	0.05	0.20	1,274	5,610	2.16	7.42	49,389	182,493	38.60	117.42	709,848	2,325,239
		3	0.06	0.19	1,543	4,890	2.27	7.75	50,761	167,295	29.07	88.74	539,070	1,820,488
	2	1	0.01	0.02	449	486	0.11	0.25	4,779	6,802	0.86	1.20	35,156	41,941
		2	0.03	0.08	791	2,109	0.52	1.55	14,495	42,867	2.42	4.13	61,193	79,843
		3	0.01	0.02	449	695	0.14	0.23	4,893	6,765	3.17	10.25	74,445	188,346
40	1	1	0.22	0.95	5,001	20,183	56.22	173.16	1,029,191	3,474,927	1393.26 **(3)**	3600.00	25,237,255	75,384,238
		2	0.51	1.45	12,069	34,402	44.17	124.92	790,039	2,241,258	1630.86 **(2)**	3600.00	26,867,022	66,977,579
		3	0.45	2.49	9,265	46,445	35.44	136.39	620,946	2,505,819	848.76	3439.00	13,211,531	48,326,244
	2	1	0.02	0.03	762	823	0.28	0.72	11,557	19,826	3.05	5.61	107,069	133,988
		2	0.03	0.08	984	1,805	2.32	10.47	46,384	183,272	55.43	146.89	842,499	2,301,605
		3	0.03	0.13	959	3,279	0.97	2.59	22,125	49,641	59.29	276.88	977,439	4,319,289
50	1	1	1.00	5.13	18,690	92,268	540.54	3351.11	4,092,988	12,323,658	3096.09 **(7)**	3600.00	45,586,112	55,990,454
		2	3.36	10.84	60,460	203,444	748.38 **(1)**	3600.00	10,904,570	53,449,052	3222.45 **(8)**	3600.00	47,124,609	55,175,001
		3	1.48	8.58	29,665	174,275	225.01	745.80	3,221,592	10,911,642	2524.57 **(6)**	3600.00	36,157,671	54,669,518
	2	1	0.02	0.02	1,140	1,249	0.51	0.59	19,882	22,894	12.00	43.28	312,456	726,420
		2	0.03	0.06	1,048	1,841	9.10	54.41	151,068	822,146	595.51	2030.27	8,074,958	28,199,665
		3	0.12	0.50	2,648	10,241	2.45	6.00	45,361	94,002	197.85	1383.47	3,024,327	21,297,472

Table 6.2.14. (cont'd) Branch and Bound performance for r =1 (OFJSS)

			m =2				m =3				m =4			
			CPU time		# of nodes		CPU time		# of nodes		CPU time		# of nodes	
n	p	w	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.
60	1	1	12.01	65.70	196,876	1,084,297	1704.17 **(2)**	3600.00	21,165,127	47,372,548	-	-	-	-
		2	6.24	22.73	100,659	347,162	1804.50 **(2)**	3600.00	21,952,731	49,653,325	-	-	-	-
		3	3.53	14.63	54,590	225,878	620.06 **(1)**	3600.00	7,202,122	41,372,448	-	-	-	-
	2	1	0.04	0.05	1,723	1,973	1.00	1.52	33,936	41,383	17.41	29.23	532,720	695,805
		2	0.36	1.59	7,640	31,241	12.64	51.67	189,358	753,382	654.92	3399.69	7,182,570	36,289,340
		3	0.09	0.22	2,296	4,647	7.87	22.41	117,230	345,966	715.71 **(1)**	3600.00	8,147,766	41,566,323
70	1	1	56.78	474.94	792,895	6,615,184	2405.15 **(5)**	3600.00	24,084,826	38,833,075	-	-	-	-
		2	37.55	165.59	540,083	2,577,023	3468.17 **(9)**	3600.00	34,407,598	40,229,463	-	-	-	-
		3	17.74	54.48	245,083	731,080	3499.79 **(8)**	3600.00	35,455,310	38,582,376	-	-	-	-
	2	1	0.06	0.06	2,211	2,485	1.72	2.08	52,185	57,225	31.74	40.81	919,363	1,012,419
		2	0.44	2.50	8,112	41,507	75.89	216.11	901,287	2,817,318	2402.50 **(6)**	3600.00	26,976,212	42,785,693
		3	0.09	0.22	2,425	4,938	20.66	62.16	259,121	805,163	2985.89 **(6)**	3600.00	26,739,793	42,338,905
80	1	1	11.88	87.24	137,815	984,326	-	-	-	-	-	-	-	-
		2	54.90	172.09	624,701	2,061,072	-	-	-	-	-	-	-	-
		3	93.21	585.86	1,077,445	6,567,072	-	-	-	-	-	-	-	-
	2	1	0.08	0.09	2,819	3,049	2.81	3.39	78,659	87,652	60.30	74.97	1,540,554	1,723,668
		2	0.48	0.97	9,140	18,893	359.16	2050.24	4,246,314	25,056,921	-	-	-	-
		3	0.23	1.09	4,469	17,551	64.49	164.11	704,018	2,004,539	-	-	-	-
90	1	1	117.26	545.89	1,168,015	5,470,243	-	-	-	-	-	-	-	-
		2	253.33	795.31	2,302,535	6,320,317	-	-	-	-	-	-	-	-
		3	106.09	338.00	1,030,053	3,239,432	-	-	-	-	-	-	-	-
	2	1	0.11	0.14	3,549	3,999	4.25	5.24	109,357	121,575	126.33	162.13	2,467,042	2,678,198
		2	0.55	1.11	9,797	22,022	272.93	864.28	2,827,938	9,576,917	-	-	-	-
		3	0.79	2.45	12,087	30,860	137.69	376.91	1,276,083	3,811,141	-	-	-	-
100	1	1	115.13	689.67	952,270	5,919,689	-	-	-	-	-	-	-	-
		2	118.82	561.91	1,151,197	6,002,648	-	-	-	-	-	-	-	-
		3	168.37	555.59	1,513,398	5,048,556	-	-	-	-	-	-	-	-
	2	1	0.16	0.19	4,535	4,851	8.83	11.30	155,377	167,825	413.00	1089.09	3,532,223	4,087,975
		2	1.48	4.61	23,817	68,732	1044.47 **(1)**	3600.00	9,764,635	37,552,030	-	-	-	-
		3	0.50	1.27	8,045	18,902	739.42	2349.39	6,001,663	18,652,472	-	-	-	-

Table 6.2.15. Branch and Bound performance for r =2 (OFJSS)

			m =2				m =3				m =4			
			CPU time		# of nodes		CPU time		# of nodes		CPU time		# of nodes	
n	p	w	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.
20	1	1	0.02	0.05	412	1,473	0.21	1.24	6,043	34,883	1.73	8.38	36,157	166,517
		2	0.01	0.03	273	864	0.10	0.44	3,439	15,671	0.55	1.36	16,613	50,001
		3	0.00	0.02	198	461	0.06	0.11	1,837	3,154	0.59	2.77	14,094	53,864
	2	1	0.00	0.00	141	193	0.03	0.06	1,093	2,260	0.13	0.23	5,661	7,658
		2	0.00	0.02	142	372	0.04	0.14	1,560	4,079	0.20	0.31	6,357	8,629
		3	0.01	0.02	134	194	0.03	0.06	1,210	1,769	0.27	0.67	8,289	17,493
30	1	1	0.02	0.05	527	1,216	0.63	1.75	13,798	39,956	7.52	24.34	134,545	462,166
		2	0.03	0.05	766	1,459	1.13	5.66	27,117	144,285	5.59	16.22	118,298	297,412
		3	0.02	0.06	624	2,399	1.17	5.59	32,209	162,277	39.31	286.72	831,264	6,179,891
	2	1	0.01	0.02	378	611	0.13	0.47	5,039	13,020	1.26	4.33	40,990	95,375
		2	0.01	0.02	303	554	0.19	0.64	6,051	17,796	4.13	27.52	87,242	490,807
		3	0.00	0.02	234	350	0.11	0.25	3,931	7,696	1.30	2.94	39,125	73,212
40	1	1	0.13	0.53	3,216	11,706	2.85	16.00	60,042	333,351	208.77	1828.33	3,439,008	31,772,825
		2	0.04	0.08	1,151	1,943	2.58	7.77	52,434	159,672	108.05	494.80	1,895,264	9,919,175
		3	0.02	0.03	565	775	5.24	16.03	103,968	297,170	230.64	1206.05	3,808,951	20,261,447
	2	1	0.02	0.03	786	1,073	0.18	0.34	8,939	12,523	2.40	3.39	99,990	121,506
		2	0.02	0.05	730	1,590	0.82	3.03	19,698	62,341	32.20	140.84	570,067	2,134,300
		3	0.01	0.03	453	853	0.48	0.97	12,648	20,249	27.31	157.72	473,837	2,586,220
50	1	1	0.77	6.20	15,801	124,154	9.92	24.94	155,308	365,213	1312.52 **(3)**	3600.00	4,452,261	9,490,588
		2	0.46	3.03	10,550	69,388	106.35	754.53	1,719,497	12,187,053	3432.10 **(7)**	3600.00	61,027,280	63,154,258
		3	0.18	0.66	4,037	14,640	19.30	86.44	349,753	1,765,573	1135.13 **(2)**	3600.00	16,813,386	52,303,154
	2	1	0.02	0.03	1,005	1,225	0.37	0.56	17,237	23,138	7.82	22.88	265,004	462,919
		2	0.04	0.11	1,068	2,221	3.15	17.56	61,113	305,619	96.47	642.64	1,492,678	9,544,784
		3	0.02	0.09	847	2,050	1.35	6.22	28,750	104,968	150.00	1327.53	2,141,731	18,329,685

Table 6.2.15. (cont'd) Branch and Bound performance for r =2 (OFJSS)

			m =2				m =3				m =4			
			CPU time		# of nodes		CPU time		# of nodes		CPU time		# of nodes	
n	p	w	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.
60	1	1	0.26	0.78	4,765	14,499	202.11	1490.08	2,627,945	19,345,982	3083.85 **(6)**	3600.00	48,926,621	62,419,015
		2	0.57	1.64	11,216	31,452	436.44	2311.72	6,150,196	31,737,314	-	-	-	-
		3	0.63	3.91	12,143	71,125	444.25 **(1)**	3600.00	6,960,919	57,375,928	2954.26 **(5)**	3600.00	51,075,104	61,245,918
	2	1	0.03	0.05	1,490	1,770	0.77	1.16	30,863	37,039	18.12	50.91	533,483	828,074
		2	0.06	0.19	1,471	3,999	1.83	7.88	43,143	175,384	266.94	1229.94	3,669,089	15,574,389
		3	0.07	0.36	1,854	7,665	1.37	3.00	31,509	53,516	163.81	781.83	2,111,344	8,827,798
70	1	1	0.88	5.80	14,439	93,470	859.52 **(2)**	3600.00	8,958,948	37,216,263	-	-	-	-
		2	0.40	1.45	7,551	25,952	865.70 **(1)**	3600.00	10,654,577	45,912,077	-	-	-	-
		3	1.09	4.47	19,095	71,011	525.75 **(1)**	3600.00	6,451,681	44,209,812	-	-	-	-
	2	1	0.05	0.08	2,168	2,517	1.43	3.97	51,690	82,588	39.58	199.16	995,830	2,326,917
		2	0.11	0.41	2,711	8,611	12.05	35.44	182,740	475,654	1491.96 **(3)**	3600.00	13,700,427	34,627,424
		3	0.08	0.39	2,076	9,859	13.19	60.92	193,755	826,234	477.83 **(1)**	3600.00	4,330,709	28,592,235
80	1	1	7.55	47.19	98,151	591,556	1197.20 **(3)**	3600.00	11,271,494	37,080,562	-	-	-	-
		2	3.07	12.44	48,474	189,245	2523.86 **(6)**	3600.00	28,134,674	41,311,460	-	-	-	-
		3	1.40	6.73	20,787	98,545	2058.56 **(5)**	3600.00	18,021,171	37,320,222	-	-	-	-
	2	1	0.06	0.08	2,474	3,081	2.43	4.41	84,527	106,515	52.30	132.09	1,643,320	2,179,174
		2	0.13	0.38	2,886	7,801	20.32	54.84	284,448	740,166	2859.45 **(7)**	3600.00	28,124,668	36,256,897
		3	0.13	0.44	2,931	8,843	17.83	75.75	269,735	1,045,352	2325.80 **(6)**	3600.00	21,095,845	32,564,897
90	1	1	2.13	11.58	26,308	147,174	2611.10 **(6)**	3600.00	24,752,375	34,240,213	-	-	-	-
		2	7.98	51.80	131,080	881,758	-	-	-	-	-	-	-	-
		3	6.70	54.16	91,066	724,910	-	-	-	-	-	-	-	-
	2	1	0.09	0.11	3,380	3,828	2.92	4.47	104,491	121,638	88.39	110.58	2,585,259	2,819,593
		2	0.35	2.14	7,150	39,926	383.76 **(1)**	3600.00	3,744,493	34,256,403	-	-	-	-
		3	0.18	0.52	3,813	9,471	68.08	387.70	821,240	4,846,952	-	-	-	-
100	1	1	35.74	108.03	380,516	1,288,234	-	-	-	-	-	-	-	-
		2	47.34	239.67	594,075	2,788,495	-	-	-	-	-	-	-	-
		3	4.05	20.30	49,306	241,992	-	-	-	-	-	-	-	-
	2	1	0.13	0.23	4,285	5,600	14.91	74.91	221,506	667,651	190.38	291.88	4,016,269	4,176,028
		2	0.16	0.92	3,570	18,335	185.72	921.19	2,153,212	12,425,005	-	-	-	-
		3	0.25	0.84	5,015	14,825	254.78	1443.28	2,533,020	13,249,646	-	-	-	-

The effect of the number of machines on solution time and the percent deviations are investigated through some additional runs with 6 and 8 machines for 20 to 50 jobs when $r = 1$ and $p = 2$. Table 6.2.16 presents the results of the associated runs. The combinatorial nature of the problem reveals itself through the exponential effect of the number of machines on the solution time and the number of nodes. The difficulty of the OFJSS problem is highly dependent on the number of machines. The upper bound at the end of Phase 1 seems to get better as the number of machines increases, potentially due to the increased number of jobs that can be processed feasibly for a given n.

Next, we investigate the effect of the upper and lower bounding mechanisms and the dominance relations on the performance of the algorithm. For presenting the effects, $(n, m) = (30,3)$ and $(40,3)$ combinations are selected as all instances of these combinations can be solved within the one-hour CPU time limit. The effects are similar for other combinations, as well.

In Table 6.2.17, the CPU times, the number of nodes, and the upper bound deviations are presented, when UBS_1 and UBS_2 are employed in the branch and bound procedure. The values are also presented for the case when a simple upper bound is used. The simple upper bound takes the weights of all not-yet-considered jobs. It can be clearly seen that the number of nodes and the solution times increase significantly on all cases with the simple upper bound. Also note that, some instances cannot be solved when the upper bounds are not used.

In Table 6.2.18, the effect of our lower bound on the efficiency of the branch and bound algorithm is reported. When an initial lower bounding procedure is not used, the lower bound is simply taken as zero.

Table 6.2.16. Effect of *m* on the BB performance (*r* =1, *p* = 2) (OFJSS)

		n =20								*n* =30							
		CPU time		# of nodes		LB Gap		UB Gap		CPU time		# of nodes		LB Gap		UB Gap	
m	***w***	**Avg.**	**Max.**	**Avg.**	**Max.**	**Avg.**	**Max.**	**Avg.**	**Max.**	**Avg.**	**Max.**	**Avg.**	**Max.**	**Avg.**	**Max.**	**Avg.**	**Max.**
2	1	0.02	0.08	712	2,977	0.00	0.00	5.14	20.22	0.06	0.30	1,742	8,346	0.15	1.47	1.76	9.90
	2	0.01	0.03	337	891	0.37	1.96	2.13	12.50	0.05	0.20	1,274	5,610	0.00	0.00	5.73	40.63
	3	0.01	0.02	320	521	0.00	0.00	13.12	34.67	0.06	0.19	1,543	4,890	0.19	1.92	8.50	52.30
4	1	0.66	3.58	12,384	59,539	0.00	0.00	22.96	37.40	145.73	691.59	2,613,038	13,160,703	0.00	0.00	22.45	31.84
	2	1.56	3.52	31,514	79,899	0.40	3.25	34.26	50.75	38.60	117.42	709,848	2,325,239	0.54	3.85	25.47	39.74
	3	0.80	4.27	15,963	85,115	0.00	0.00	29.91	48.55	29.07	88.74	539,070	1,820,488	0.48	3.00	26.67	46.10
6	1	0.23	2.33	6,366	63,661	0.00	0.00	9.11	91.11	2027.00 **(6)**	3600.00	17,449,943	29,568,456	0.00	0.00	12.14	25.83
	2	0.00	0.00	65,985	225,642	0.00	0.00	0.00	0.00	1867.77 **(6)**	3600.00	17,854,523	29,965,412	0.00	0.00	13.71	25.83
	3	15.68	151.70	221,344	2,151,983	0.00	0.00	16.00	119.02	2532.24 **(7)**	3600.00	20,263,668	32,896,541	0.00	0.00	10.21	22.55
8	1	0.00	0.00	36,521	124,556	0.00	0.00	0.00	0.00	0.00	0.00	56,844	985,223	0.00	0.00	0.00	0.00
	2	0.00	0.00	23,514	122,457	0.00	0.00	0.00	0.00	0.00	0.00	568,400	568,442	0.00	0.00	0.00	0.00
	3	0.00	0.00	23,254	56,842	0.00	0.00	0.00	0.00	0.00	0.00	521,469	1,524,541	0.00	0.00	0.00	0.00

		n =40								*n* =50							
		CPU time		# of nodes		LB Gap		UB Gap		CPU time		# of nodes		LB Gap		UB Gap	
m	***w***	**Avg.**	**Max.**	**Avg.**	**Max.**	**Avg.**	**Max.**	**Avg.**	**Max.**	**Avg.**	**Max.**	**Avg.**	**Max.**	**Avg.**	**Max.**	**Avg.**	**Max.**
2	1	0.22	0.95	5,001	20,183	0.00	0.00	0.97	6.90	1.00	5.13	18,690	92,268	0.00	0.00	0.83	5.00
	2	0.51	1.45	12,069	34,402	0.00	0.00	15.39	65.91	3.36	10.84	60,460	203,444	0.00	0.00	5.58	47.16
	3	0.45	2.49	9,265	46,445	0.00	0.00	1.69	8.77	1.48	8.58	29,665	174,275	0.00	0.00	1.45	11.05
4	1	1393.26 **(3)**	3600.00	25,237,255	75,384,238	0.27	1.33	18.62	29.78	-	-	-	-	-	-	-	-
	2	1630.86 **(2)**	3600.00	26,867,022	66,977,579	0.39	1.91	17.26	29.39	-	-	-	-	-	-	-	-
	3	848.76	3439.00	13,211,531	48,326,244	0.71	2.58	22.05	38.34	-	-	-	-	-	-	-	-
6	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table 6.2.17. Effect of upper bound on BB performance (OFJSS)

					With upper bounds						Without upper bounds			
					CPU time		# of nodes		UB Gap %		CPU time		# of nodes	
n	*m*	*r*	*p*	*w*	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.
30	3	1	1	1	1.61	5.27	36,797	114,914	10.35	20.13	29.61	59.33	1,129,116	2,256,022
				2	2.16	7.42	49,389	182,493	10.56	19.88	20.91	46.48	816,497	2,012,373
				3	2.27	7.75	50,761	167,295	13.59	22.15	15.13	89.88	629,397	3,854,148
			2	1	0.11	0.25	4,779	6,802	0.46	3.44	11.35	20.88	420,498	807,923
				2	0.52	1.55	14,495	42,867	5.32	12.38	4.48	11.64	164,981	309,893
				3	0.14	0.23	4,893	6,765	6.52	13.68	2.73	5.23	96,945	195,963
		2	1	1	0.63	1.75	13,798	39,956	11.34	37.00	7.70	22.77	305,183	883,822
				2	1.13	5.66	27,117	144,285	19.93	47.66	18.49	53.09	688,392	2,109,133
				3	1.17	5.59	32,209	162,277	20.14	38.38	7.12	21.20	278,949	724,695
			2	1	0.13	0.47	5,039	13,020	1.60	9.88	5.85	28.66	205,132	919,026
				2	0.19	0.64	6,051	17,796	1.82	9.71	1.54	2.92	63,545	141,785
				3	0.11	0.25	3,931	7,696	10.58	57.89	0.83	1.94	33,276	82,745
40	3	1	1	1	56.22	173.16	1,029,191	3,474,927	9.75	16.67	2513.87**(5)**	3600.00	63,888,621	87,121,988
				2	44.17	124.92	790,039	2,241,258	9.16	17.95	2734.04**(4)**	3600.00	75,333,521	93,856,159
				3	35.44	136.39	620,946	2,505,819	14.40	30.56	1540.04**(3)**	3600.00	43,891,113	93,856,159
			2	1	0.28	0.72	11,557	19,826	0.33	1.82	168.17	650.97	6,274,882	26,333,958
				2	2.32	10.47	46,384	183,272	6.41	15.45	38.56	119.30	1,317,661	3,877,013
				3	0.97	2.59	22,125	49,641	4.96	13.86	32.56	110.41	1,113,724	3,710,450
		2	1	1	2.85	16.00	60,042	333,351	9.29	18.12	132.11	387.42	4,770,987	15,636,479
				2	2.58	7.77	52,434	159,672	8.49	15.63	174.64	810.64	6,058,142	30,368,702
				3	5.24	16.03	103,968	297,170	20.46	36.87	85.83	400.89	2,815,623	11,965,805
			2	1	0.18	0.34	8,939	12,523	9.51	79.74	25.34	118.36	795,257	3,850,522
				2	0.82	3.03	19,698	62,341	4.78	16.00	11.51	22.75	387,288	831,875
				3	0.48	0.97	12,648	20,249	6.92	18.09	6.55	23.02	242,286	784,132

Table 6.2.18. Effect of lower bound on BB performance (OFJSS)

					With LB						Without LB			
					CPU time		# of nodes		LB Gap %		CPU time		# of nodes	
n	*m*	*r*	*p*	*w*	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.
30	3	1	1	1	1.61	5.27	36,797	114,914	0.00	0.00	7.48	22.22	179,144	533,504
				2	2.16	7.42	49,389	182,493	0.23	1.16	26.15	196.49	493,915	3,575,710
				3	2.27	7.75	50,761	167,295	0.40	1.74	4.02	9.28	78,066	190,055
			2	1	0.11	0.25	4,779	6,802	0.15	1.15	0.12	0.25	4,802	6,802
				2	0.52	1.55	14,495	42,867	0.98	5.71	0.60	1.61	15,860	43,744
				3	0.14	0.23	4,893	6,765	0.49	2.50	0.16	0.25	5,053	7,235
		2	1	1	0.63	1.75	13,798	39,956	0.00	0.00	1.35	6.64	27,066	147,122
				2	1.13	5.66	27,117	144,285	0.00	0.00	2.99	16.48	71,104	441,328
				3	1.17	5.59	32,209	162,277	0.25	2.27	3.05	13.25	66,017	253,649
			2	1	0.13	0.47	5,039	13,020	0.00	0.00	0.15	0.50	5,169	13,020
				2	0.19	0.64	6,051	17,796	0.49	4.85	0.28	0.78	8,191	23,974
				3	0.11	0.25	3,931	7,696	0.31	3.08	0.14	0.36	4,384	11,296
40	3	1	1	1	56.22	173.16	1,029,191	3,474,927	0.22	1.08	123.05	574.27	2,440,439	12,456,631
				2	44.17	124.92	790,039	2,241,258	0.72	2.54	183.71	816.08	3,299,768	14,935,609
				3	35.44	136.39	620,946	2,505,819	0.42	2.56	153.18	908.81	2,513,500	14,628,233
			2	1	0.28	0.72	11,557	19,826	0.04	0.36	0.31	0.86	11,856	22,811
				2	2.32	10.47	46,384	183,272	0.19	1.89	2.94	12.66	53,903	206,785
				3	0.97	2.59	22,125	49,641	0.46	3.00	2.07	7.69	39,921	138,649
		2	1	1	2.85	16.00	60,042	333,351	0.59	4.05	4.99	23.14	78,118	336,120
				2	2.58	7.77	52,434	159,672	0.13	1.33	14.88	93.70	266,872	1,645,565
				3	5.24	16.03	103,968	297,170	0.13	1.31	13.10	53.44	257,744	1,081,908
			2	1	0.18	0.34	8,939	12,523	0.39	2.69	0.21	0.38	8,961	12,731
				2	0.82	3.03	19,698	62,341	0.46	2.00	0.98	3.34	20,596	64,996
				3	0.48	0.97	12,648	20,249	0.91	4.51	0.62	0.98	13,520	20,325

It can be seen from Table 6.2.18 that, in all instances, the average number of nodes, in turn CPU times, increase significantly when the lower bound is removed. The results also show that our lower bounding procedure is especially useful when there is a high variability in processing times. In this case, only a small number of jobs are allowed on each machine, and most of the problems can be solved before the end of Phase 1 by our efficient lower bounding procedure.

The effect of the reduction mechanisms and the precedence relations on the performance is examined next. In Table 6.2.19, we report the CPU times, the number of nodes, and the average number of jobs after reduction when we incorporate the properties of the optimal solution into our BB algorithm. We also present these values for the cases when groups of these properties are not employed.

In Table 6.2.19, the properties that are employed in Phase 1 form one group, while the properties that are the same for both the OFJSW and OFJSS problems form the second group. Finally, the properties related with the second phase of the BB algorithm form the last group. Our initial runs for the OFJSS problem have revealed that all properties are effective individually, so all are employed in the BB procedure.

The table reveals that each group of our reduction and precedence mechanisms for the OFJSS problem are quite effective in reducing the CPU times and the number of nodes. Actually, when we remove all properties from the branch and bound procedure, the CPU times and the number of nodes nearly triple in most of the problem instances.

The node at which optimal solution is found and the total number of nodes are given in Table 6.2.20, for n = 40 and 50. This table also validates the result that most of the optimal solutions are found in Phase 1. This indicates that a truncated version of our branch and bound algorithm that terminates at the end of Phase 1 can be a good alternative to optimal solutions for the larger-sized problems.

Table 6.2.19. Effect of reduction mechanisms on BB performance (OFJSS)

					With reduction & precedence					Without Theorems 4.2.1 through 4.2.3				Without Theorems 3.2.1 and 3.2.3				Without Theorems 4.2.6 through 4.2.8			
					CPU time		# of nodes		avg. # of	CPU time		# of nodes		CPU time		# of nodes		CPU time		# of nodes	
n	*m*	*r*	*p*	*w*	Avg.	Max.	Avg.	Max.	rem. jobs	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.
30	3	1	1	1	1.61	5.27	36,797	114,914	30	2.25	7.37	51,516	160,880	2.33	8.63	59,007	251,214	2.01	6.58	45,996	143,643
				2	2.16	7.42	49,389	182,493	30	3.02	10.39	69,144	255,490	6.18	30.32	161,900	771,828	2.70	9.28	61,736	228,116
				3	2.27	7.75	50,761	167,295	30	3.17	10.85	71,065	234,213	8.75	50.32	190,046	1,043,748	2.83	9.69	63,451	209,119
			2	1	0.11	0.25	4,779	6,802	30	0.15	0.35	6,691	9,523	0.08	0.18	3,517	5,443	0.14	0.31	5,974	8,503
				2	0.52	1.55	14,495	42,867	28	0.73	2.17	20,294	60,014	1.02	3.25	26,085	81,089	0.65	1.93	18,119	53,584
				3	0.14	0.23	4,893	6,765	29	0.19	0.33	6,850	9,471	0.14	0.24	4,920	7,642	0.17	0.29	6,116	8,456
		2	1	1	0.63	1.75	13,798	39,956	30	0.88	2.45	19,317	55,938	0.78	2.71	17,568	67,748	0.78	2.19	17,248	49,945
				2	1.13	5.66	27,117	144,285	29	1.59	7.92	37,964	201,999	1.31	6.84	34,298	186,360	1.42	7.07	33,897	180,356
				3	1.17	5.59	32,209	162,277	29	1.64	7.83	45,093	227,188	2.34	8.28	65,893	250,492	1.46	6.99	40,262	202,846
			2	1	0.13	0.47	5,039	13,020	30	0.18	0.66	7,055	18,228	0.17	0.94	6,125	27,675	0.16	0.59	6,299	16,275
				2	0.19	0.64	6,051	17,796	27	0.27	0.90	8,471	24,914	0.34	1.03	10,387	29,128	0.24	0.80	7,564	22,245
				3	0.11	0.25	3,931	7,696	27	0.15	0.35	5,503	10,774	0.14	0.43	4,919	11,743	0.13	0.31	4,913	9,620
40	3	1	1	1	56.22	173.16	1,029,191	3,474,927	40	78.71	242.42	1,440,868	4,864,898	101.82	288.27	1,926,368	5,304,391	70.28	216.45	1,286,489	4,343,659
				2	44.17	124.92	790,039	2,241,258	40	61.84	174.89	1,106,055	3,137,761	124.42	328.25	2,326,734	5,899,299	55.21	156.15	987,549	2,801,573
				3	35.44	136.39	620,946	2,505,819	40	49.62	190.95	869,324	3,508,147	164.13	631.86	3,017,662	11,464,508	44.30	170.49	776,182	3,132,274
			2	1	0.28	0.72	11,557	19,826	40	0.39	1.01	16,180	27,756	0.20	0.50	8,595	17,004	0.35	0.90	14,446	24,783
				2	2.32	10.47	46,384	183,272	36	3.25	14.66	64,937	256,581	4.15	11.73	84,937	220,623	2.90	13.09	57,980	229,090
				3	0.97	2.59	22,125	49,641	37	1.36	3.63	30,975	69,497	1.45	5.74	30,988	105,962	1.22	3.24	27,656	62,051
		2	1	1	2.85	16.00	60,042	333,351	40	3.99	22.40	84,059	466,691	7.86	56.15	134,842	932,963	3.56	20.00	75,052	416,689
				2	2.58	7.77	52,434	159,672	38	3.62	10.87	73,408	223,541	4.78	18.97	93,853	397,907	3.23	9.71	65,543	199,590
				3	5.24	16.03	103,968	297,170	36	7.33	22.44	145,555	416,038	14.65	56.58	275,968	1,028,563	6.54	20.04	129,959	371,463
			2	1	0.18	0.34	8,939	12,523	40	0.25	0.48	12,514	17,532	0.17	0.28	7,848	9,809	0.22	0.43	11,174	15,654
				2	0.82	3.03	19,698	62,341	32	1.15	4.24	27,577	87,277	1.72	6.45	36,849	121,264	1.03	3.79	24,622	77,926
				3	0.48	0.97	12,648	20,249	33	0.68	1.36	17,707	28,349	0.95	1.91	22,913	42,823	0.60	1.21	15,810	25,311

Table 6.2.20. Optimal solution node vs. total number of nodes (OFJSS)

				m=2				m=3				m=4			
				solution node		# of nodes		solution node		# of nodes		solution node		# of nodes	
n	r	p	w	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.
40	1	1	1	655	798	5,001	20,183	34,743	165,318	1,029,191	3,474,927	463,640	3,752,244	25,237,255	75,384,238
			2	449	720	12,069	34,402	111,356	844,911	790,039	2,241,258	284,000	1,751,493	26,867,022	66,977,579
			3	556	758	9,265	46,445	125,760	1,133,539	620,946	2,505,819	1,740,306	8,411,696	13,211,531	48,326,244
		2	1	283	601	762	823	7,978	10,943	11,557	19,826	85,949	119,308	107,069	133,988
			2	455	620	984	1,805	6,776	8,955	46,384	183,272	89,066	181,953	842,499	2,301,605
			3	459	717	959	3,279	10,693	35,602	22,125	49,641	264,507	1,078,666	977,439	4,319,289
	2	1	1	490	649	3,216	11,706	9,582	23,436	60,042	333,351	142,626	649,634	3,439,008	31,772,825
			2	381	597	1,151	1,943	15,335	93,254	52,434	159,672	68,989	92,154	1,895,264	9,919,175
			3	399	683	565	775	5,163	14,799	103,968	297,170	106,689	289,226	3,808,951	20,261,447
		2	1	424	758	786	1,073	6,329	10,784	8,939	12,523	78,925	103,140	99,990	121,506
			2	221	428	730	1,590	6,477	19,673	19,698	62,341	57,629	145,749	570,067	2,134,300
			3	331	537	453	853	5,182	9,940	12,648	20,249	196,241	996,002	473,837	2,586,220
50	1	1	1	1,027	1,204	18,690	92,268	235,330	1,783,288	4,092,988	12,323,658	-	-	-	-
			2	956	1,245	60,460	203,444	436,345	3,742,353	10,904,570	53,449,052	-	-	-	-
			3	950	1,251	29,665	174,275	758,429	7,126,107	3,221,592	10,911,642	-	-	-	-
		2	1	715	1,041	1,140	1,249	11,531	17,030	19,882	22,894	183,926	603,235	312,456	726,420
			2	554	884	1,048	1,841	28,936	70,041	151,068	822,146	239,709	763,765	8,074,958	28,199,665
			3	1,265	6,968	2,648	10,241	20,774	81,466	45,361	94,002	214,261	511,856	3,024,327	21,297,472
	2	1	1	611	1,247	15,801	124,154	16,779	18,902	155,308	365,213	146,490	252,800	4,452,261	9,490,588
			2	502	1,250	10,550	69,388	12,539	15,687	1,719,497	12,187,053	-	-	-	-
			3	603	842	4,037	14,640	16,575	49,960	349,753	1,765,573	689,943	4,421,434	16,813,386	52,303,154
		2	1	607	1,062	1,005	1,225	11,414	19,982	17,237	23,138	206,214	250,307	265,004	462,919
			2	262	515	1,068	2,221	8,617	31,146	61,113	305,619	150,663	624,240	1,492,678	9,544,784
			3	350	691	847	2,050	7,359	11,977	28,750	104,968	116,892	157,437	2,141,731	18,329,685

6.2.3. Results for the OFJSG Problem

This section presents the computational analysis for the OFJSG problem. The performance of our lower and upper bounds is investigated first. Table 6.2.21 reports the average and maximum root node lower and upper bound deviations as a percentage of optimal solutions for r =1 and r =2 up to 50 jobs. Values for larger problems left out, as most of the instances cannot be solved, and there is no difference in performance for solved instances. We use UBG = Min $\{UBG_1, UBG_2\}$ as an upper bound for the OFJSG problem at each node of the branch and bound tree. We disregard the MCNF-based bound, as our initial experiments have shown that the effort spent to calculate this bound is more than its savings due to node eliminations. The table also presents the number of times the root node lower bound gives the optimal solution in the "opt." columns. An empty entry in the table indicates that the algorithm could not return an optimal solution within the one-hour time limit for any problem instance.

One observation from the table is that, the performances of bounds, in particular the lower bounds, are consistently satisfactory. The performance of bounds for r = 1 are slightly better than those for r = 2. Essentially, when r = 1, the lower and upper bounding procedures yield respective average deviations of no more than 26% and 47% from the optimal solutions among all instances. When r = 2, the maximum respective average deviations become 28% and 54%.

In Tables 6.2.22 and 6.2.23, the CPU times (in seconds) and the number of nodes are presented. The numbers in parentheses are the number of instances that cannot be solved within one hour. It can be seen that r = 1 case is harder than r = 2 case despite the slightly better performance of the upper bound. This is due to the fact that r = 2 case has many more overlapping jobs in peak periods, and offers more potential reduction opportunities. When the processing times are concerned, p = 1 (low variability) case reduces the efficiency, due to the reduced differentiability power of the upper bound. The variability differences in weights do not seem to affect the performance.

Table 6.2.21. LB and UB deviations at root node (OFJSG)

				m =2					*m* =3					*m* =4				
				LB Gap (%)		UB Gap (%)			LB Gap		UB Gap			LB Gap		UB Gap		
r	*n*	*p*	*w*	Avg.	Max.	Avg.	Max.	opt.	Avg.	Max.	Avg.	Max.	opt.	Avg.	Max.	Avg.	Max.	opt.
1	20	1	1	6.12	19.13	15.43	25.26	2	20.38	29.41	6.02	14.47	0	19.06	28.76	3.18	6.25	1
			2	12.85	20.25	10.75	20.62	0	20.14	30.74	5.42	11.65	0	25.40	39.73	2.91	6.78	1
		2	1	6.52	16.30	26.27	50.70	2	13.01	24.19	22.97	34.86	0	21.73	29.25	12.09	16.43	0
			2	5.97	15.03	29.50	38.82	1	11.92	21.06	25.09	33.96	0	21.85	29.13	10.87	20.28	0
	30	1	1	6.51	11.67	17.09	28.38	0	18.76	25.45	8.72	12.56	0	25.12	31.39	3.94	7.50	0
			2	8.45	14.52	17.43	25.96	0	16.82	22.15	12.27	19.21	0	19.98	28.83	5.53	9.72	0
		2	1	8.29	15.97	39.61	49.41	0	7.33	15.33	30.36	34.69	0	13.62	20.00	25.66	32.34	0
			2	5.87	13.93	38.06	50.25	2	8.75	16.21	33.04	39.31	1	13.38	20.33	23.64	34.44	0
	40	1	1	5.70	10.05	21.31	26.94	0	13.65	19.42	13.22	19.46	0	19.39	26.62	6.77	10.04	0
			2	7.80	12.75	21.47	29.78	0	15.59	20.97	14.50	19.62	0	17.14	26.14	7.07	9.63	0
		2	1	5.11	11.67	35.82	48.74	0	8.48	14.12	39.27	52.80	0	10.64	18.78	34.75	41.15	0
			2	3.25	7.59	42.37	53.36	0	10.98	19.74	37.44	48.34	0	9.02	17.67	29.42	39.14	0
	50	1	1	3.86	6.22	28.44	35.19	1	14.42	54.55	18.88	24.48	0	16.84	53.12	15.72	65.85	0
			2	5.04	10.08	25.85	38.18	0	14.16	53.73	16.40	22.02	0	16.01	56.09	21.57	48.02	0
		2	1	3.90	11.59	41.32	52.86	3	6.05	12.25	39.51	50.87	0	-	-	-	-	-
			2	3.10	8.10	43.47	53.95	1	5.99	8.69	38.03	44.40	0	11.42	58.51	29.39	40.15	0
2	20	1	1	11.27	16.16	19.24	29.76	1	12.73	27.42	13.55	22.99	0	19.46	30.00	11.73	19.84	0
			2	7.25	15.31	21.53	46.59	1	18.56	30.39	14.34	22.97	0	22.29	26.71	11.46	19.34	0
		2	1	4.88	12.07	40.25	56.90	2	10.32	18.75	32.51	48.24	0	10.73	22.66	23.86	37.65	0
			2	9.56	18.32	35.36	62.26	1	12.04	23.76	29.07	40.39	0	11.51	16.37	25.64	33.24	0
	30	1	1	9.81	15.60	18.35	33.03	0	14.23	22.60	19.90	29.53	0	17.22	25.28	17.48	22.93	0
			2	13.78	25.05	20.10	28.41	0	13.27	23.93	21.39	31.82	0	16.86	26.67	18.35	27.35	0
		2	1	6.73	16.90	38.98	67.12	2	10.48	19.31	34.86	49.06	0	10.85	15.86	35.98	42.06	0
			2	4.31	10.51	41.89	53.74	2	11.77	20.47	35.07	48.65	0	13.56	20.92	31.47	39.23	0
	40	1	1	4.30	13.67	22.45	29.01	1	10.72	16.43	17.98	22.22	0	16.43	23.59	21.90	52.76	1
			2	8.10	18.06	22.81	28.86	0	13.67	21.33	20.71	28.67	0	15.70	29.82	18.66	44.59	0
		2	1	5.21	9.30	42.96	55.13	1	10.95	22.64	39.68	50.35	0	11.74	19.50	36.51	46.02	0
			2	4.11	8.00	47.30	79.93	0	8.22	21.37	44.14	63.79	0	10.74	20.58	35.24	49.08	0
	50	1	1	9.19	18.93	21.77	26.85	0	14.74	33.77	21.96	28.78	0	13.83	85.84	21.69	79.38	0
			2	8.39	15.63	22.91	29.36	1	12.86	16.82	18.04	25.69	0	-	-	-	-	-
		2	1	3.65	7.29	48.36	65.88	2	7.64	12.34	44.15	80.81	0	15.65	77.73	31.72	47.67	0
			2	5.42	12.42	40.84	56.90	1	8.77	17.55	44.34	55.38	0	27.99	67.37	23.31	43.24	0

Table 6.2.22. Branch and Bound performance for *r* =1 (OFJSG)

			CPU time		# of nodes		CPU time		# of nodes		CPU time		# of nodes	
n	***p***	***w***	**Avg.**	**Max.**	**Avg.**	**Max.**	**Avg.**	**Max.**	**Avg.**	**Max.**	**Avg.**	**Max.**	**Avg.**	**Max.**
20	1	1	0.04	0.13	940	3,704	0.22	1.23	4,248	24,655	0.17	0.61	2,691	10,602
		2	0.07	0.38	1,479	9,857	0.19	1.53	3,661	30,672	0.11	0.36	1,743	5,832
	2	1	0.02	0.03	275	713	0.25	1.13	4,076	18,182	0.51	1.55	7,401	22,248
		2	0.04	0.16	883	2,886	0.29	0.52	4,525	7,902	2.17	15.27	31,675	218,904
30	1	1	0.22	0.48	2,978	6,380	1.46	5.25	15,559	55,273	18.38	108.55	171,244	1,055,574
		2	0.19	0.59	2,257	7,644	63.35	346.00	654,300	3,441,047	69.07	527.78	580,780	4,267,107
	2	1	0.13	0.27	1,591	3,562	3.83	11.33	34,679	106,934	544.16 **(1)**	3600.00	3,840,728	24,689,398
		2	0.08	0.16	846	1,928	5.89	27.44	50,357	245,852	64.86	303.31	451,852	2,144,340
40	1	1	2.89	11.08	25,302	93,423	62.62	301.08	379,470	1,745,175	225.75	787.58	1,181,312	4,080,737
		2	2.09	8.53	17,139	66,355	792.50 **(1)**	3600.00	4,901,172	25,366,752	837.02 **(2)**	3600.00	6,320,753	30,351,662
	2	1	0.69	2.27	5,797	18,352	96.43	499.58	542,778	2,883,551	2620.49 **(7)**	3600.00	11,701,773	16,971,028
		2	0.60	1.27	4,876	11,916	92.49	382.34	471,698	1,805,149	1925.71 **(4)**	3600.00	8,436,918	16,971,028
50	1	1	11.00	24.97	66,361	175,020	1879.94 **(3)**	3600.00	10,296,227	28,362,726	1533.62 **(3)**	3600.00	6,030,178	16,239,153
		2	30.57	206.20	163,388	1,115,039	1707.61 **(4)**	3600.00	6,391,696	13,154,889	2515.42 **(6)**	3600.00	9,994,245	15,643,865
	2	1	2.61	11.28	16,294	73,955	449.29	2205.61	1,725,545	8,635,350	-	-	-	-
		2	1.62	3.69	9,218	19,252	112.70	439.05	407,251	1,511,997	3079.44 **(6)**	3600.00	10,160,678	13,952,706
60	1	1	50.43	113.95	221,598	537,676	-	-	-	-	-	-	-	-
		2	28.47	156.63	106,311	561,013	-	-	-	-	-	-	-	-
	2	1	3.83	9.53	19,006	45,475	1333.34 **(1)**	3600.00	3,916,837	10,457,577	-	-	-	-
		2	6.30	16.89	29,080	81,668	499.23	902.44	1,401,568	2,445,786	-	-	-	-

Table 6.2.22. (cont'd) Branch and Bound performance for r =1 (OFJSG)

			m=2				m=3			
			CPU time		# of nodes		CPU time		# of nodes	
n	p	w	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.
70	1	1	216.29	1485.08	760,337	5,129,243	-	-	-	-
		2	156.15	1169.97	509,495	3,790,899	-	-	-	-
	2	1	10.61	24.02	45,069	110,841	2316.76 **(3)**	3600.00	6,099,011	14,096,964
		2	6.73	15.39	26,073	54,432	2188.09 **(3)**	3600.00	5,374,362	14,096,964
80	1	1	252.10	882.08	696,350	2,430,919	-	-	-	-
		2	242.02	1163.13	616,244	2,840,010	-	-	-	-
	2	1	42.81	125.94	157,989	478,478	3126.11 **(8)**	3600.00	6,773,343	9,622,058
		2	17.09	50.14	53,261	143,864	3185.30 **(7)**	3600.00	6,354,132	8,924,368
90	1	1	589.94	1659.05	1,366,261	3,404,355	-	-	-	-
		2	528.42 **(1)**	3600.00	1,212,698	8,508,821	-	-	-	-
	2	1	69.84	120.77	213,803	394,830	-	-	-	-
		2	34.95	70.74	99,675	209,964	-	-	-	-
100	1	1	1668.20 **(2)**	3600.00	3,404,348	7,804,256	-	-	-	-
		2	816.40	2770.67	1,440,616	4,626,009	-	-	-	-
	2	1	90.17	173.03	237,489	478,392	-	-	-	-
		2	45.42	123.66	108,022	291,857	-	-	-	-

Table 6.2.23. Branch and Bound performance for r =2 (OFJSG)

			m =2				m =3				m =4			
			CPU time		# of nodes		CPU time		# of nodes		CPU time		# of nodes	
n	p	w	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.
20	1	1	0.03	0.06	512	1,186	0.22	1.44	4,568	31,045	0.91	2.42	15,104	37,509
		2	0.04	0.08	733	1,971	0.41	1.59	8,240	31,374	0.94	4.06	15,003	72,191
	2	1	0.02	0.05	337	544	0.20	0.48	3,250	7,177	3.21	12.06	49,031	189,835
		2	0.03	0.06	443	1,048	0.54	1.63	9,351	29,605	0.94	2.19	11,476	24,801
30	1	1	0.18	0.64	2,741	9,894	26.26	111.39	276,881	1,066,187	181.35	570.45	1,634,232	4,624,294
		2	0.88	6.19	11,578	78,986	19.85	68.66	213,014	764,007	1089.78 **(2)**	3600.00	9,480,185	35,108,258
	2	1	0.07	0.14	879	2,251	6.32	19.41	63,369	212,990	159.76	814.42	1,239,291	6,298,251
		2	0.09	0.16	1,030	1,848	5.41	19.09	49,698	178,182	88.02	294.47	617,939	1,847,235
40	1	1	0.79	2.13	7,559	18,185	348.73	1847.14	2,571,994	13,932,249	2322.88 **(6)**	3600.00	15,560,590	31,114,941
		2	1.22	3.48	12,404	37,136	550.84	1550.56	3,380,955	8,891,627	2243.94 **(4)**	3600.00	15,224,984	27,614,510
	2	1	0.25	0.64	2,637	8,704	34.09	64.56	230,334	453,861	2227.57 **(4)**	3600.00	12,582,391	25,545,648
		2	0.37	0.84	3,533	8,191	29.80	137.39	181,588	879,226	1366.39 **(2)**	3600.00	6,999,200	22,278,206
50	1	1	2.37	5.33	18,436	40,906	786.80 **(1)**	3600.00	4,633,268	21,093,815	3359.50 **(9)**	3600.00	18,422,810	22,138,532
		2	7.12	27.42	48,922	176,189	672.64	2520.80	3,324,960	11,742,049	-	-	-	-
	2	1	1.03	3.00	8,015	18,917	103.36	218.24	523,935	1,074,313	3431.52 **(9)**	3600.00	14,454,441	16,975,465
		2	0.91	2.33	6,577	16,973	92.12	232.36	421,495	1,110,023	3128.66 **(8)**	3600.00	11,549,167	15,340,701
60	1	1	16.68	100.34	95,926	562,356	2249.09 **(5)**	3600.00	8,565,768	17,837,234	-	-	-	-
		2	10.66	36.23	52,920	164,491	2398.52 **(4)**	3600.00	9,698,068	15,565,763	-	-	-	-
	2	1	2.21	8.06	12,080	41,017	860.15 **(1)**	3600.00	3,294,215	14,943,836	-	-	-	-
		2	1.18	2.56	7,545	14,456	442.28	2378.41	1,595,221	8,409,804	-	-	-	-

Table 6.2.23. (cont'd) Branch and Bound performance for r =2 (OFJSG)

			m =2				m =3			
			CPU time		# of nodes		CPU time		# of nodes	
n	p	w	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.
70	1	1	64.88	391.44	273,902	1,524,392	-	-	-	-
		2	11.83	50.67	58,979	282,801	-	-	-	-
	2	1	3.98	7.80	22,546	46,066	2073.00 **(3)**	3600.00	6,494,048	13,287,122
		2	3.55	8.78	18,956	52,498	1624.79 **(2)**	3600.00	5,028,617	13,025,980
80	1	1	87.81	424.25	371,796	1,606,844	-	-	-	-
		2	59.77	196.83	244,173	807,099	-	-	-	-
	2	1	12.75	35.88	55,711	165,701	2920.71 **(6)**	3600.00	7,427,352	10,685,416
		2	4.76	16.03	20,341	55,659	1831.52 **(4)**	3600.00	4,390,652	8,912,273
90	1	1	165.40	601.59	578,408	2,060,585	-	-	-	-
		2	48.72	207.44	187,069	786,215	-	-	-	-
	2	1	12.43	25.44	54,598	132,358	-	-	-	-
		2	11.60	31.45	45,387	127,441	-	-	-	-
100	1	1	677.80	3430.91	1,896,587	9,832,368	-	-	-	-
		2	253.65	1350.95	798,339	4,862,442	-	-	-	-
	2	1	55.30	210.30	186,069	649,224	-	-	-	-
		2	12.62	28.95	47,020	108,756	-	-	-	-

To see the effect of the number of machines on the efficiency of the algorithm more clearly, some additional runs are performed with 6 and 8 machines for 20 to 50 jobs, and r = 1 and p = 2. Table 6.2.24 presents the results of the associated runs. The exponential effect of the number of machines on the solution time and the number of nodes is apparent from the table. However, as the number of machines increase, our initial upper bound slightly improves. This is due to the fact that more jobs can be processed feasibly as the number of machines increases for a given n, therefore the relaxed and the optimal solutions become closer.

Next, we investigate the effect of the upper and lower bounding mechanisms and the dominance relations on the performance of our BB algorithm. For presenting the effects, (n, m) = (20,3) and (30,2) combinations are selected as all instances of these combinations can be solved within the one-hour CPU time limit, and the effects on these combinations are representative for all instances.

In Table 6.2.25, the CPU times, the number of nodes, and the upper bound deviations are presented, when *UBG* is utilized as the upper bound. The associated values are also presented for the case when a simple upper bound is used. The simple upper bound simply sets a single weight for a job as the maximum of its weights among all machines, and then takes the total weight of all not-yet-considered jobs as an upper bound. The table reveals that our upper bound is quite effective. Note that, with simple upper bound case, three of the instances cannot be solved for n =20, m =3, r =1, p =1, w =1 combination, however all instances are solved within one second when *UBG* is employed.

In Table 6.2.26, the effect of our lower bound on the efficiency of the branch and bound algorithm is reported. When an initial lower bounding procedure is not used, the lower bound is simply taken as zero. There is almost no change in the average number of nodes and the CPU times when the lower bound is removed. This is due to the fact that depth-first strategy returns a complete solution at early nodes, and this solution is likely to be a strong one due to the strong upper bounds.

Table 6.2.24. Effect of *m* on the BB performance (*r* =1, *p* = 2) (OFJSG)

		n =20								*n* =30							
		CPU time		# of nodes		LB Gap		UB Gap		CPU time		# of nodes		LB Gap		UB Gap	
m	***w***	**Avg.**	**Max.**	**Avg.**	**Max.**	**Avg.**	**Max.**	**Avg.**	**Max.**	**Avg.**	**Max.**	**Avg.**	**Max.**	**Avg.**	**Max.**	**Avg.**	**Max.**
2	1	0.02	0.03	275	713	6.52	16.30	26.27	50.70	0.13	0.27	1,591	3,562	8.29	15.97	39.61	49.41
	2	0.04	0.16	883	2,886	5.97	15.03	29.50	38.82	0.08	0.16	846	1,928	5.87	13.93	38.06	50.25
4	1	0.51	1.55	7,401	22,248	21.73	29.25	12.09	16.43	544.16 **(1)**	3600.00	3,840,728	24,689,398	13.62	20.00	25.66	32.34
	2	2.17	15.27	31,675	218,904	21.85	29.13	10.87	20.28	64.86	303.31	451,852	2,144,340	13.38	20.33	23.64	34.44
6	1	1.92	9.48	20,246	91,133	25.07	33.54	4.90	11.32	1274.04 **(1)**	3600.00	6,600,729	17,814,526	22.87	30.86	12.46	22.37
	2	9.64	77.91	106,455	873,249	25.73	34.83	5.98	13.96	953.37 **(2)**	3600.00	5,637,997	21,972,649	25.30	31.30	11.44	19.24
8	1	23.62	220.92	202,692	1,855,208	30.75	42.53	2.36	6.90	757.36 **(1)**	3600.00	3,391,150	15,276,652	28.95	35.04	5.69	9.56
	2	2.40	21.56	22,273	201,194	35.13	43.50	2.22	5.36	406.13	2629.73	1,506,249	9,002,695	27.01	33.36	4.46	6.96

		n =40								*n* =50							
		CPU time		# of nodes		LB Gap		UB Gap		CPU time		# of nodes		LB Gap		UB Gap	
m	***w***	**Avg.**	**Max.**	**Avg.**	**Max.**	**Avg.**	**Max.**	**Avg.**	**Max.**	**Avg.**	**Max.**	**Avg.**	**Max.**	**Avg.**	**Max.**	**Avg.**	**Max.**
2	1	0.69	2.27	5,797	18,352	5.11	11.67	35.82	48.74	2.61	11.28	16,294	73,955	3.90	11.59	41.32	52.86
	2	0.60	1.27	4,876	11,916	3.25	7.59	42.37	53.36	1.62	3.69	9,218	19,252	3.10	8.10	43.47	53.95
4	1	2620.49 **(7)**	3600.00	11,701,773	16,971,028	10.64	18.78	34.75	41.15	-	-	-	-	-	-	-	-
	2	1925.71 **(4)**	3600.00	8,436,918	16,971,028	9.02	17.67	29.42	39.14	3079.44 **(6)**	3600.00	10,160,678	13,952,706	11.42	58.51	29.39	40.15

Table 6.2.25. Effect of upper bound on BB performance (OFJSG)

					With upper bounds						Without upper bounds			
					CPU time		# of nodes		UB Gap		CPU time		# of nodes	
n	*m*	*r*	*p*	*w*	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.
20	3	1	1	1	0.22	1.23	4,248	24,655	6.02	14.47	881.07	1789.47	36,968,905	77,076,154
				2	0.19	1.53	3,661	30,672	5.42	11.65	1594.76**(3)**	3600.00	55,345,446	173,822,122
			2	1	0.25	1.13	4,076	18,182	22.97	34.86	0.80	3.91	60,220	250,711
				2	0.29	0.52	4,525	7,902	25.09	33.96	0.82	2.92	61,271	195,887
		2	1	1	0.22	1.44	4,568	31,045	13.55	22.99	5.99	30.48	342,845	1,460,293
				2	0.41	1.59	8,240	31,374	14.34	22.97	158.36	1123.69	6,893,215	46,186,799
			2	1	0.20	0.48	3,250	7,177	32.51	48.24	0.22	0.47	17,705	37,161
				2	0.54	1.63	9,351	29,605	29.07	40.39	1.55	5.20	118,808	487,369
30	2	1	1	1	0.22	0.48	2,978	6,380	17.09	28.38	2.03	12.64	112,341	670,986
				2	0.19	0.59	2,257	7,644	17.43	25.96	0.65	1.50	50,578	114,155
			2	1	0.13	0.27	1,591	3,562	39.61	49.41	0.05	0.08	3,272	5,900
				2	0.08	0.16	846	1,928	38.06	50.25	0.03	0.08	2,069	5,548
		2	1	1	0.18	0.64	2,741	9,894	18.35	33.03	0.81	2.23	48,612	123,885
				2	0.88	6.19	11,578	78,986	20.10	28.41	2.10	11.30	107,646	466,729
			2	1	0.07	0.14	879	2,251	38.98	67.12	0.03	0.05	1,874	3,438
				2	0.09	0.16	1,030	1,848	41.89	53.74	0.04	0.08	2,270	5,646

Table 6.2.26. Effect of lower bound on BB performance (OFJSG)

					With LB						Without LB			
					CPU time		# of nodes		LB Gap		CPU time		# of nodes	
n	*m*	*r*	*p*	*w*	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.
20	3	1	1	1	0.22	1.23	4,248	24,655	20.38	29.41	0.22	1.22	4,248	24,655
				2	0.19	1.53	3,661	30,672	20.14	30.74	0.19	1.53	3,661	30,672
			2	1	0.25	1.13	4,076	18,182	13.01	24.19	0.25	1.13	4,076	18,182
				2	0.29	0.52	4,525	7,902	11.92	21.06	0.31	0.58	4,530	7,937
		2	1	1	0.22	1.44	4,568	31,045	12.73	27.42	0.22	1.41	4,569	31,045
				2	0.41	1.59	8,240	31,374	18.56	30.39	0.42	1.58	8,241	31,374
			2	1	0.20	0.48	3,250	7,177	10.32	18.75	0.21	0.50	3,251	7,177
				2	0.54	1.63	9,351	29,605	12.04	23.76	0.54	1.61	9,352	29,605
30	2	1	1	1	0.22	0.48	2,978	6,380	6.51	11.67	0.22	0.50	2,978	6,380
				2	0.19	0.59	2,257	7,644	8.45	14.52	0.19	0.58	2,257	7,644
			2	1	0.13	0.27	1,591	3,562	8.29	15.97	0.13	0.27	1,593	3,563
				2	0.08	0.16	846	1,928	5.87	13.93	0.08	0.16	849	1,928
		2	1	1	0.18	0.64	2,741	9,894	9.81	15.60	0.17	0.63	2,741	9,894
				2	0.88	6.19	11,578	78,986	13.78	25.05	0.96	7.06	11,578	78,986
			2	1	0.07	0.14	879	2,251	6.73	16.90	0.07	0.14	883	2,251
				2	0.09	0.16	1,030	1,848	4.31	10.51	0.10	0.16	1,090	1,877

The effect of the reduction mechanisms due to the precedence relations on the performance is also examined. In Table 6.2.27, we report the CPU times, the number of nodes, and the average number of jobs after reduction when we incorporate the properties of the optimal solution into the BB algorithm. We compare these values with the cases that does not use each the properties.

Note that Theorems 5.3.3 and 5.3.4 are grouped together in the table since they are implemented together in the BB procedure. Our initial experimentation for the OFJSG problem has revealed that each one of the properties are effective in reducing computation times, so all are employed in the BB algorithm.

The table reveals that all our reduction mechanisms are quite effective on the branch and bound algorithm. The CPU times and the number of nodes increase when each of the reduction mechanisms is not used. The increase is more pronounced when the problem size gets larger. In fact, when none of the properties are employed, the increase in the CPU times and number of nodes are very significant.

The relation between the node at which optimal solution is found and the total number of nodes is also investigated through Table 6.2.28, for n = 40 and 50. One can easily observe that the optimal solution is generally found in very early stages of the algorithm. This indicates that the solutions found at termination limit are very likely to be optimal, and a truncated version of our branch and bound algorithm can be a good alternative to optimal solutions for the larger-size problems.

In order to impose a kind of eligibility structure into the problem, some extra runs are performed with agreeable weights, i.e. $w_{jr} \geq w_{ir}\ \forall r$, between any two jobs i and j. Namely, we define the weights such that, a job is more valuable than another on all machines, thereby facilitating problem size reduction. The weights are uniformly generated in the interval [0,40] as in the w = 2 case.

Table 6.2.27. Effect of reduction mechanisms on BB performance (OFJSG)

					With reduction & precedence					Without Theorem 5.3.1				Without Theorem 5.3.2				Without Theorems 5.3.3 and 5.3.4			
					CPU time		# of nodes		avg. # of	CPU time		# of nodes		CPU time		# of nodes		CPU time		# of nodes	
n	*m*	*r*	*p*	*w*	Avg.	Max.	Avg.	Max.	rem. jobs	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.
20	3	1	1	1	0.22	1.23	4,248	24,655	20	0.21	1.20	4,250	24,742	0.21	1.20	4,255	24,396	0.32	2.05	6,512	41,782
				2	0.19	1.53	3,661	30,672	20	0.18	1.52	3,674	30,672	0.19	1.53	3,663	30,672	0.28	2.27	5,546	46,638
			2	1	0.25	1.13	4,076	18,182	20	0.27	1.17	4,311	18,600	0.26	1.17	4,186	18,297	1.51	8.24	27,013	141,064
				2	0.29	0.52	4,525	7,902	20	0.33	0.53	5,110	8,259	0.32	0.58	4,957	8,247	2.15	4.63	38,680	82,614
		2	1	1	0.22	1.44	4,568	31,045	20	0.24	1.50	4,980	33,151	0.23	1.45	4,750	31,852	0.49	2.45	10,570	54,904
				2	0.41	1.59	8,240	31,374	20	0.43	1.56	8,517	30,863	0.42	1.53	8,290	30,297	1.08	5.20	23,501	122,150
			2	1	0.20	0.48	3,250	7,177	20	0.26	0.63	4,139	9,494	0.23	0.63	3,623	9,307	1.32	3.75	24,507	67,609
				2	0.54	1.63	9,351	29,605	19	0.67	1.77	10,768	30,977	0.61	1.86	10,143	32,271	3.01	10.72	52,948	177,679
30	2	1	1	1	0.22	0.48	2,978	6,380	30	0.24	0.50	3,192	6,315	0.23	0.49	3,066	6,315	5.64	34.11	77,393	421,642
				2	0.19	0.59	2,257	7,644	30	0.20	0.58	2,443	7,613	0.20	0.58	2,432	7,518	7.16	32.42	92,484	416,477
			2	1	0.13	0.27	1,591	3,562	29	0.19	0.41	2,258	5,510	0.19	0.41	2,161	5,240	3.43	10.56	44,813	129,281
				2	0.08	0.16	846	1,928	28	0.10	0.20	1,127	2,334	0.12	0.23	1,190	2,579	3.48	9.52	48,523	131,616
		2	1	1	0.18	0.64	2,741	9,894	28	0.24	0.81	3,675	13,044	0.29	1.30	4,209	18,514	1.80	9.13	28,288	135,204
				2	0.88	6.19	11,578	78,986	29	1.27	9.59	16,348	118,568	1.25	9.16	16,472	116,238	3.85	22.84	52,992	288,920
			2	1	0.07	0.14	879	2,251	27	0.11	0.20	1,490	3,379	0.14	0.31	1,660	3,890	0.91	3.69	13,832	47,460
				2	0.09	0.16	1,030	1,848	27	0.14	0.23	1,672	2,728	0.18	0.28	1,938	3,126	2.21	4.86	33,277	84,563

Table 6.2.28. Optimal solution node vs. total number of nodes (OFJSG)

				m=2				m=3				m=4			
				solution node		# of nodes		solution node		# of nodes		solution node		# of nodes	
n	r	p	w	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.
30	1	1	1	248	1,124	2,978	6,380	1,992	10,886	15,559	55,273	13,414	130,057	171,244	1,055,574
			2	524	1,737	2,257	7,644	6,412	58,822	654,300	3,441,047	187,341	1,834,620	580,780	4,267,107
		2	1	221	657	1,591	3,562	5,707	31,599	34,679	106,934	402,015	3,134,362	3,840,728	24,689,398
			2	160	449	846	1,928	8,886	45,193	50,357	245,852	156,138	438,244	451,852	2,144,340
	2	1	1	609	1,908	2,741	9,894	58,203	300,993	276,881	1,066,187	57,096	292,200	1,634,232	4,624,294
			2	484	1,948	11,578	78,986	35,495	140,860	213,014	764,007	1,044,103	4,849,478	9,480,185	35,108,258
		2	1	143	543	879	2,251	22,600	126,988	63,369	212,990	126,328	516,976	1,239,291	6,298,251
			2	171	681	1,030	1,848	6,119	19,950	49,698	178,182	150,431	408,032	617,939	1,847,235
40	1	1	1	6,346	47,714	25,302	93,423	32,706	194,146	379,470	1,745,175	50,749	297,814	1,181,312	4,080,737
			2	2,601	11,879	17,139	66,355	378,880	1,301,335	4,901,172	25,366,752	2,946,985	29,385,100	6,320,753	30,351,662
		2	1	917	6,440	5,797	18,352	83,256	220,636	542,778	2,883,551	2,700,490	12,100,432	11,701,773	16,971,028
			2	603	2,577	4,876	11,916	159,919	765,604	471,698	1,805,149	1,423,618	9,031,893	8,436,918	16,971,028
	2	1	1	1,196	4,728	7,559	18,185	375,573	2,331,898	2,571,994	13,932,249	2,691,124	15,140,465	15,560,590	31,114,941
			2	946	5,764	12,404	37,136	256,469	1,066,253	3,380,955	8,891,627	1,336,006	4,810,559	15,224,984	27,614,510
		2	1	378	846	2,637	8,704	76,539	280,420	230,334	453,861	1,718,432	5,080,760	12,582,391	25,545,648
			2	545	2,822	3,533	8,191	19,277	57,077	181,588	879,226	3,127,554	18,234,931	6,999,200	22,278,206

Table 6.2.29 compares the performance of the algorithm between the $w = 2$ case and the agreeable weights on two job and machine combinations, when r =1. Expectedly, the numbers of nodes, in turn the solution times are smaller when the weights are agreeable. The upper bound performances also get better when a job is consistently more valuable. This is due to the fact that, when solving the assignment problem of Algorithm 5.3.1 for an interval, we can consider only the most-weighted $O(m)$ jobs instead of considering all available jobs. Hence, the overall complexity of the algorithm becomes $O(nm^3)$ instead of $O(n^4)$. We did not observe any significant difference between lower bound performances.

Table 6.2.29. Performance with agreeable weights (r =1) (OFJSG)

w_{jk} ~U(0,40)									
			CPU time		# of nodes		LB Gap %		UB Gap %	
n	m	p	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.
40	2	1	2.09	8.53	17,139	66,355	7.80	12.75	21.47	29.78
		2	0.60	1.27	4,876	11,916	3.25	7.59	42.37	53.36
	3	1	792.50 **(1)**	3600.00	4,901,172	25,366,752	15.59	20.97	14.50	19.62
		2	92.49	382.34	471,698	1,805,149	10.98	19.74	37.44	48.34
50	2	1	30.57	206.20	163,388	1,115,039	5.04	10.08	25.85	38.18
		2	1.62	3.69	9,218	19,252	3.10	8.10	43.47	53.95
	3	1	1707.61 **(4)**	3600.00	6,391,696	13,154,889	14.16	53.73	16.40	22.02
		2	112.70	439.05	407,251	1,511,997	5.99	8.69	38.03	44.40

w_{jk} ~U(0,40), new weight structure									
			CPU time		# of nodes		LB Gap %		UB Gap %	
n	m	p	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.
40	2	1	1.52	7.44	8,954	55,478	8.25	13.55	17.52	22.23
		2	0.98	1.55	9,254	12,554	3.25	8.63	35.62	48.23
	3	1	312.70	1439.05	2,565,895	5,621,254	14.22	19.58	13.20	18.22
		2	52.49	354.73	125,454	1,255,221	11.02	19.54	33.28	45.78
50	2	1	21.45	185.64	112,452	985,898	7.02	10.55	22.55	33.22
		2	1.54	4.80	11,225	20,125	2.90	7.65	39.98	52.21
	3	1	789.65 **(2)**	3600.00	1,458,745	9,652,457	13.23	42.25	16.20	22.02
		2	65.25	198.32	110,447	1,145,784	8.56	12.25	33.80	41.20

Finally, we investigate the efficiency of our Decomposition Theorem (Theorem 5.3.4). Table 6.2.30 shows the average and maximum number of subproblems for all instances. It can be observed that the respective average number of decompositions are at least 5 and 2 among all problems, for the combinations p = 1

and p = 2. The number of subproblems decreases as the number of jobs increases, due to increasing number of job overlaps. The figures indicate that a developed solution strategy may benefit significantly from decompositions due to the exponential nature of the algorithm, especially for large size problems.

Table 6.2.30. Number of decompositions in instances of the OFJSG problem

			r = 1						r = 2					
			m =2		m =3		m =4		m =2		m =3		m =4	
			Number of subproblems		Number of subproblems		Number of subproblems		Number of subproblems		Number of subproblems		Number of subproblems	
n	p	w	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.
20	1	1	12	15	12	14	11	15	9	11	8	11	9	11
		2	12	15	11	13	11	15	8	10	10	12	9	10
	2	1	4	7	4	6	4	6	4	6	4	6	4	6
		2	5	6	4	4	4	7	5	6	4	5	4	5
30	1	1	12	17	12	16	12	17	11	13	10	12	9	11
		2	11	13	12	14	12	16	10	12	9	11	10	12
	2	1	3	4	3	4	3	4	4	6	4	5	4	4
		2	3	5	3	4	3	3	4	6	4	7	4	6
40	1	1	12	15	11	13	12	14	9	11	11	14	11	15
		2	11	15	11	15	11	14	10	13	11	13	9	13
	2	1	3	5	3	5	3	4	3	5	3	5	3	4
		2	3	6	2	3	2	4	4	6	3	5	3	4
50	1	1	10	13	9	12	10	12	10	13	10	12	11	14
		2	10	13	11	12	11	13	11	15	11	14	11	13
	2	1	2	4	2	3	3	3	4	5	3	4	3	4
		2	2	4	2	4	2	3	3	4	3	6	3	5
60	1	1	9	12	9	11	9	11	11	14	11	13	11	13
		2	10	13	8	10	9	10	11	13	10	13	10	13
	2	1	2	2	2	3	2	3	3	5	3	5	3	4
		2	2	3	2	2	2	3	3	5	4	6	3	3
70	1	1	9	13	8	11	8	10	10	12	9	11	10	13
		2	8	10	8	11	7	9	11	13	11	13	11	13
	2	1	2	3	2	4	2	2	3	5	2	3	2	3
		2	2	3	2	2	2	2	3	4	3	4	3	4
80	1	1	7	10	6	8	7	9	11	13	10	13	10	12
		2	7	10	6	8	6	9	11	12	10	11	11	14
	2	1	2	2	2	2	2	2	2	4	2	3	3	4
		2	2	3	2	3	2	2	3	4	3	4	2	3
90	1	1	7	8	6	7	6	8	9	13	10	11	10	12
		2	6	7	6	7	6	8	11	15	11	14	10	12
	2	1	2	2	2	3	2	2	3	4	2	4	2	3
		2	2	3	2	3	2	2	3	5	3	4	2	3
100	1	1	5	7	5	7	6	8	9	13	10	13	9	10
		2	5	8	6	6	5	7	9	12	10	12	10	12
	2	1	2	3	2	2	2	2	3	4	3	4	2	4
		2	2	4	2	2	2	2	3	4	2	3	2	3

The effect of decompositions on the performance of the BB algorithm is explored on a small set of moderate size problems with 40 and 50 jobs, 2 and 3 machines. In Table 6.2.31, the CPU times and the number of nodes are presented for $r = 2$ when the decomposition rule is used and not used. The average numbers of decompositions are also presented. When the decomposition rule is used the decomposed subproblems are solved independently, and the table presents the total solution times and number of nodes.

The figures indicate that the decomposition rule proves very valuable. Note that when the theorem is employed, the average solution times are under one second for $p = 1$ case, where there are higher number of decompositions (see Table 6.2.30). Hence, it seems that the $p = 1$ setting becomes easier than the $p = 2$ setting in this case, in contrast to the case that does not use the theorem. The table also reveals that larger size problems are very likely to be solved in reasonable times when the decomposition rule is utilized.

Table 6.2.31. The effect of decompositions on BB performance ($r = 2$) (OFJSG)

				Without decomposition				With decomposition				
				CPU time		# of nodes		avg. # of	CPU time		# of nodes	
n	*m*	*p*	*w*	Avg.	Max.	Avg.	Max.	subprob.	Avg.	Max.	Avg.	Max.
40	2	1	1	0.79	2.13	7,559	18,185	9	0.01	0.03	256	395
			2	1.22	3.48	12,404	37,136	10	0.01	0.02	199	334
		2	1	0.25	0.64	2,637	8,704	3	0.09	0.22	929	2,007
			2	0.37	0.84	3,533	8,191	4	0.09	0.47	972	3,396
	3	1	1	348.73	1847.14	2,571,994	13,932,249	11	0.08	0.14	1,764	3,311
			2	550.84	1550.56	3,380,955	8,891,627	11	0.10	0.17	2,078	3,496
		2	1	34.09	64.56	230,334	453,861	3	26.78	83.55	114,501	322,836
			2	29.80	137.39	181,588	879,226	3	2.42	12.25	12,882	48,290
50	2	1	1	2.37	5.33	18,436	40,906	10	0.02	0.03	357	620
			2	7.12	27.42	48,922	176,189	11	0.02	0.03	355	574
		2	1	1.03	3.00	8,015	18,917	4	0.11	0.58	1,181	5,681
			2	0.91	2.33	6,577	16,973	3	0.42	1.30	3,095	8,875
	3	1	1	786.80 **(1)**	3600.00	4,633,268	21,093,815	10	0.26	0.42	5,290	8,951
			2	672.64	2520.80	3,324,960	11,742,049	11	0.15	0.53	3,212	10,220
		2	1	103.36	218.24	523,935	1,074,313	3	57.59	215.72	286,997	1,074,313
			2	92.12	232.36	421,495	1,110,023	3	16.94	64.52	86,913	272,366

CHAPTER 7

CONCLUSIONS

In this chapter, we first give a brief summary of our study, and state important conclusions and contributions. Then, we present two more topics from FJS literature that may be worth studying. Finally, we present other research directions.

7.1. SUMMARY AND CONCLUSIONS

In this study, we have considered the Operational Fixed Job Scheduling problem. The problem involves many tasks, each with specified ready times and deadlines, that are to be processed on a number of identical parallel resources. Our objective is maximizing weighted number of processed jobs with a given number of processors.

The problem is typical for reservation systems and has many real life applications, such as classroom assignment, transportation systems, and shift scheduling.

Researchers have studied these variations only for the tactical version of the FJS problem. Despite its practical importance the research on the OFJS problem is rather limited, and to the best of our knowledge, this study is the first to attempt the above problems. We aim to fill the gap in the OFJS literature.

We first established a complexity relation between operational and tactical fixed job scheduling problems. Using this relation, we resolved the complexity status of an OFJS problem, given the complexity status of the tactical version. We have analyzed the problem under three environments. The first environment includes working time constraints, where the actual time that the machine operates is

bounded; that is, no machine is allowed to operate for more than a given total working time of T units. The second environment includes spread time constraints, where an upper bound S is imposed on the total time between the start and the finish times of the operations on any machine. We established a relation between the optimal solutions of the problems with working time and spread time constraints. The third environment has machine dependent job weights (general weights). A special case of this problem becomes the OFJS problem with eligibility constraints, where each machine is eligible to process a subset of jobs, but not all jobs.

We have shown that all three problems are NP-hard in the strong sense. For each problem environment, we have identified some polynomially solvable special cases. We also proposed branch and bound algorithms that employ powerful reduction mechanisms and efficient lower and upper bounds.

The computational results revealed that, for all problem environments, the algorithms return optimal solutions for problem instances with up to 100 jobs in reasonable solution times. Moreover, the optimal solutions are found in very early levels of branching, and this leads us to conclude that the truncated versions of the algorithms are very likely to perform well on larger problem instances. The results for the OFJSW and OFJSS problems have shown that, other than problem size, the most important parameters that affect the difficulty of the problem are the processing times and weight distribution. Higher variability in both of these parameters brings higher computational times. When the results for the OFJSG problem are concerned, when low variability in processing times reduces the efficiency, due to the reduced upper bound performances resulting from higher number of nonoverlapping job combinations. The variability differences in weights do not seem to affect the performance for this problem. We also find the decomposition theorem very effective, and hope to solve the problems of much higher sizes by using the theorem.

7.2. FURTHER TOPICS IN FIXED JOB SCHEDULING

In this section, we discuss some topics in FJS that are open for further research.

7.2.1. The FJS Problem with Availability Constraints

When any of the machines is available only for a specified time interval rather than being continuously available, the resulting problem is FJS with availability constraints. The availability constraints usually appear as shifts of consecutive hours for each machine, or sets of machines. For example, in the airline company example defined in Chapter 1, there are different shifts for the engineers, since the arrivals and departures of aircrafts take place continuously. Teams of engineers are available at the airport during specific time intervals (shifts) of some consecutive hours. In such a situation, a job can be carried out in a shift by a machine if the time between the ready time and the deadline of the job forms a subinterval of the machine's shift. Different shifts may have different costs, as well.

The following additional parameters are necessary for the formulation of the problem:

- Z: Total number of shifts,
- z: Shift index, $z = 1,\ldots,Z$,
- b_z: Beginning time of shift z, $z = 1,\ldots,Z$,
- e_z: Ending time of shift z, $z = 1,\ldots,Z$,
- M_z: Number of machines available in shift z, $z = 1,\ldots,Z$,
- $z(k)$: The shift in which machine k is available, $k = 1,\ldots,m$.

Note that, if $M_z > n$, then scheduling of jobs in shift z is easy, as each job can have its own machine. Hence, it is assumed that $M_z \leq n$, $\forall z$. A job j can be processed by a machine k, if and only if the time between the ready time and the deadline of the job, i.e. (r_j, d_j), forms a subinterval of the shift of machine k, which is denoted by $(b_{z(k)}, e_{z(k)})$. Mathematically;

$x_{jk} = 0 \iff r_j < b_{z(k)}$ or $d_j > e_{z(k)}$.

The FJS problem with eligibility constraints, and the FJS problem with availability constraints are closely related. To see the relation, recall the definition of Arkin and Silverberg (1987) for the OFJS problem with eligibility constraints. V_j represents the subset of machines by which job j can be processed. The same representation can be used for formulation of the problem with availability constraints as follows:

$V_j : \{k: b_{z(k)} \leq r_j \text{ and } d_j \leq e_{z(k)}\}$.

Hence, the OFJS problem with availability constraints can be seen as a special case of the OFJS problem with eligibility constraints. Therefore, the $O(n^{m+1})$ time algorithm of Arkin and Silverberg (1987) can solve the OFJS problem with availability constraints, as well.

The OFJS problem with availability constraints is studied by Kolen, Lenstra and Papadimitriu (1986). They show that the feasibility problem is NP-complete. Kolen and Kroon (1993) also study the OFJS problem with availability constraints. The authors show that the OFJS problem with availability constraints is polynomially solvable if and only if there are no overlapping shifts that have different beginning and ending times.

As it was reviewed in Chapter 5, Jansen (1994) provides an approximation algorithm for the TFJS problem with eligibility and availability constraints. There are two shifts in the problem, and machine costs differ among shifts. His algorithm has a complexity of $O(F.Z.n^2)$, where F is the number of machine classes, and Z is the number of shifts.

Bouzina and Emmons (1995) conjecture that the OFJS problem with availability constraints and maximum number objective is NP-hard. The authors provide a polynomial time algorithm for the preemptive version of the problem with number and weight maximization objectives. Moreover, they propose a polynomial time

algorithm for the preemptive case of maximum number problem having a hierarchical eligibility structure (represented by matrix L_1) in addition to the availability constraints.

There seems to be a relation between the OFJS problems with eligibility and availability constraints. Hence, the results derived for the problem with eligibility constraints can be exploited for the problem with availability constraints. The problem with availability constraints seems also related with the OFJSS problem (the OFJS problem with spread time constraints). Note that, the OFJSS problem is a more general case where the beginning times of the shifts of the machines are not predetermined, but are decision variables.

7.2.2. The FJS Problem with Uniform Machines

So far, we have analyzed the FJS problem where all machines are identical in terms of processing speeds. As opposed to this situation, the machine speed factors may determine the processing times of the jobs, in which case the problem is referred to as FJS problem with uniform machines. As an example, the relay satellites in the EOS problem defined in Chapter 1 may have different transmission speeds, and the overall speed of transmission to the ground may depend on both the observing satellite and the relay satellite that it links to. The following additional parameter, and change in processing time definition is necessary for formulating the problem:

- s_k: Speed factor associated with machine k, $k = 1,\ldots,m$,
- p_{jk}: Processing time of job j on machine k, $j = 1,\ldots,n$, $k = 1,\ldots,m$.

The processing time of a job j on machine k is expressed as the product of p_j and a speed factor s_k:

$$p_{jk} = p_j * s_k , \quad \forall j, k.$$

As to the best of our knowledge, there are only two studies on the FJS problem with uniform machines. Nakajima *et al.* (1982) study the TFJS problem with two types

of uniform machines: inexpensive slow machines (type 1) and expensive fast machines (type 2). n jobs, each having a window of processing $[r_j, r_j + p_j*s_k]$ on machine k, are to be processed on these machines. Different costs are defined for these machines, and the problem is to find the number of machines of each type to process all jobs so that the total cost is minimized. The authors show that the problem is NP-hard in the strong sense. They provide an $O(n)$ time algorithm for the special case where all ready times are equal. They also develop an $O(n \log n)$ time algorithm for the special case where the processing times of all jobs on the fast machines are equal to one.

Bekki (2003) studies the operational version of the above problem. He shows that the problem is NP-hard in the strong sense. An $O(m \log n)$ time algorithm is provided for the special case where all ready times are equal. The author also develops a polynomial time algorithm for the special case where the processing times of all jobs on the fast machine are equal to one. An $O(mn)$ time algorithm is introduced for the special preemptive case where $p_j = w_j$, $\forall j$, as well. For the general problem, a Branch and Bound algorithm is proposed, which employs precedence relations, and powerful lower and upper bounding procedures. His computational analysis reveals that the procedure finds solutions for large-size problem instances in reasonable times.

There is even a more general case of the FJS problem with unrelated machines, where the speed factor is defined relative to the jobs and machines, and s_{jk} denotes the speed factor of machine k for job j. The processing time of job j on machine k is then p_j*s_{jk}.

Most of problems analyzed in the previous chapters are NP-hard even when the machines are identical. Exact algorithms may be generated for some special cases for the uniform machine environments. Detecting these special cases and proposing solution approaches for the general case can open new research avenues.

7.2.3. Other Research Directions

We have studied working time constraints and spread time constraints separately. What may be the case in a real life situation is that these operating time constraints may appear together, like in the Bus Driver Scheduling Problem, or airline crew scheduling. For example, in the airline crew scheduling, the time that a pilot spends on the plane (spread time) is limited as well as the time he actually flies the plane (working time). Hence, studying the FJS problem under both operating time constraints may be worthwhile, and fill a gap in the literature.

Another application may involve eligibility constraints besides the operating time constraints. In fact, the OFJS problem with spread time constraints is a special case of the problem with eligibility constraints. Once the first jobs are set, we can redefine the job weights as follows, and use the general weight procedures to handle the spread time constraints.

$$w_{jk} = \begin{cases} 0 & \text{if job } j \text{ cannot be processed due to spread time constraints,} \\ w_j & \text{otherwise.} \end{cases}$$

For working time constraints, we can evaluate the partial schedules using the results of the general weight problem. In doing so, the weights can be defined as:

$$w_{jk} = \begin{cases} 0 & \text{if job } j \text{ cannot be processed due to } T\text{-constraints,} \\ w_j & \text{otherwise.} \end{cases}$$

Relations between the problem environments may be another issue of research. We have already shown in Chapter 2 that the optimal solution of the problem with a working time constraint T is an upper bound on the optimal solution of the problem with a spread time constraint S, provided that $S = T$. Exploring the relationships between the problem with eligibility constraints and the problems with operating time constraints (note that one such relation is given by the above weight definition) may also be meaningful.

Another research topic may be the Interval Selection Problem, which is a generalization of the FJS problem, where each job has several alternative ready times. Hence, the ready time that the job will start its processing is an additional decision variable. Jobs may also have different weights associated with different alternative intervals.

Another generalization is the Variable Job Scheduling Problem that is introduced in Chapter 1. In this problem, the processing time p_j of job j is smaller than or equal to (d_j - r_j). Hence, there is a flexibility in the start time of the accepted job; it is a decision variable that takes a value between r_j and (d_j - p_j). All problem environments analyzed in the previous chapters for the FJS problem can be defined, and the solution procedures can be modified for the variable case, as well.

As mentioned, the computational experiments revealed that, the optimal solutions for all three problems are found in very early stages of the total search. This implies that, the truncated versions of the developed branch and bound procedures are likely to give satisfactory solutions for larger problem sizes. These algorithms may be necessary especially for problem instances with higher number of machines, as the difficulty is very dependent on the number of machines. The worst-case behaviors of our lower bounds need to be explored, as well.

When larger problem instances are concerned, more efficient approximation algorithms based on intelligent search methods like metaheuristics may also be developed for any of our problems. For instance, a Tabu Search approach that utilizes problem specific information as optimal solution properties in neighborhood definition may be considered. An initial solution may be generated with one of the developed lower bounds, and the moves may be defined as swaps of jobs between machines and/or insertion/deletion of a job from the schedule. In such a case, the recent move and its reverse move should be stored in a tabu list for a number of iterations called the tabu duration. Parameters of the algorithm like tabu duration or stopping conditions are to be explored through initial experimentation.

REFERENCES

Arkin A.M., Silverberg E.L., 1987. Scheduling Jobs with Fixed Start and End Times, *Discrete Applied Mathematics*, 18, 1-8.

Bekki Ö.B., 2003. "Fixed Job Scheduling on Uniform Parallel Machines", MS Thesis, Department of Industrial Engineering, METU.

Bouzina K.I., Emmons H., 1995. "Interval Scheduling with Shift and 2-Level Hierarchy Constraints", presentation in: *Informs Meeting,* Los Angeles.

Bouzina K.I., Emmons H., 1996. Interval Scheduling on Identical Machines, *Journal of Global Optimization*, 9, 379-393.

Dantzig G.L., Fulkerson D.R., 1954. Minimizing the Number of Tankers to Meet a Fixed Schedule, *Naval Research Logistics Quarterly,* 1, 217-222.

Dondeti V.R., Emmons H., 1992. Fixed Job Scheduling with Two Types of Processors, *Operations Research*, 40, S76-S85.

Dondeti V.R., Emmons H., 1993. Algorithms for Preemptive Scheduling of Different Classes of Processors to do Jobs with Fixed Times, *European Journal of Operational Research*, 70, 316-326.

Dilworth R.P., 1950. A Decomposition Principle for Partially Ordered Sets, *Annals of Mathematics*, 51, 161-166.

Fischetti M., Martello S., Toth P., 1987. The Fixed Job Schedule Problem with Spread-Time Constraints, *Operations Research,* 35, 849-858.

Fischetti M., Martello S., Toth P., 1989. The Fixed Job Schedule Problem with Working-Time Constraints, *Operations Research*, 37, 395-403.

Fischetti M., Martello S., Toth P., 1992. Approximation Algorithms for Fixed Job Schedule Problems, *Operations Research*, 40, S96-S108.

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

Gertsbakh I., Stern H.I., 1978. Minimal Resources for Fixed and Variable Job Schedules, *Operations Research*, 26, 68-85.

Gupta U.L., Lee D.T., Leung J.Y.-T., 1979. An Optimal Solution to the Channel Assignment Problem, *IEEE Transactions on Computers*, 28, 807-810.

Hashimoto A., Stevens J.E., 1971. "Wire Routing by Optimizing Channel Assignments within Large Apertures", in: *Proceedings of the 8th Design Automation Workshop*, 155-169.

Jansen K., 1994. An Approximation Algorithm for the License and Shift Class Design Problem, *European Journal of Operational Research*, 73, 127-131.

Kolen A.J.W., Lenstra J.K., Papadimitriu C.H., 1986. Interval Scheduling Problems, *Manuscript*, Centre for Mathematics and Computer Science, C.W.I., Kruislaan 413, 1098 SJ Amsterdam.

Kolen A.J.W., Kroon L.G., 1991. On the Computational Complexity of (Maximum) Class Scheduling, *European Journal of Operational Research*, 54, 23-38.

Kolen A.J.W., Kroon L.G., 1992. License Class Design: Complexity and Algorithms, *European Journal of Operational Research*, 63, 432-444.

Kolen A.J.W., Kroon L.G., 1993. On the Computational Complexity of (Maximum) Shift Class Scheduling, *European Journal of Operational Research*, 64, 138-151.

Kroon L.G., 1990. "Job Scheduling and Capacity Planning in Aircraft Maintenance", PhD Thesis, Rotterdam School of Management, Erasmus University, The Netherlands.

Kroon L.G., Salomon M., Van Wassenhove L.N., 1995. Exact and Approximation Algorithms for the Operational Fixed Interval Scheduling Problem, *European Journal of Operational Research*, 82, 190-205.

Kroon L.G., Salomon M., Van Wassenhove L.N., 1997. Exact and Approximation Algorithms for the Tactical Fixed Interval Scheduling Problem, *Operations Research*, 4, 624-638.

Nakajima K., Hakimi S.L., Lenstra J.K., 1982. Complexity Results for Scheduling Tasks in Fixed Intervals on Two Types of Machines, *SIAM Journal on Computing*, 11, 512-520.

Orlin J.B., 1993. A Faster Strongly Polynomial Minimum Cost Flow Algorithm, *Operations Research*, 41, 338-350.

Santos E.J., Zhong X., 2001. Genetic Algorithms and Reinforcement Learning for the Tactical Fixed Interval Scheduling Problem, *International Journal on Artificial Intelligence Tools*, 10, 23-38.

Wolfe W.J., Sorensen S.E., 2000. Three Scheduling Algorithms Applied to the Earth Observing Systems Domain, *Management Science*, 46, 148-168.

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Türsel Eliiyi, Deniz
Nationality: Turkish (TC)
Date and Place of Birth: 27 April 1977 , İzmir
Marital Status: Married
Phone: +90 312 210 47 94
Fax: +90 312 210 12 68
email: deniz@ie.metu.edu.tr

EDUCATION

Degree	Institution	Year of Graduation
MS	METU Industrial Engineering	2001
BS	METU Industrial Engineering	1998
High School	60. Yıl Anadolu High School, İzmir	1994

WORK EXPERIENCE

Year	Place	Enrollment
1998-present	METU Department of Industrial Engineering	Research Assistant
1997 August	ERICSSON	Intern Engineering Student
1996 July	VESTELKOM	Intern Engineering Student

FOREIGN LANGUAGES

Advanced English, Basic German, Basic Spanish

PUBLICATIONS

1. Türsel Eliiyi, D., Azizoğlu, M., "Fixed Job Scheduling Problem: A Literature Review", *Proceedings of the 24th National Congress on Operational Research and Industrial Engineering*, Adana, Turkey (2004).

2. Türsel Eliiyi, D., Kandiller, L., "A Decision Support System for Cell Formation Problem", *Proceedings of 30th International Conference on Computers & Industrial Engineering*, Tinos Island, Greece (2002).

HOBBIES

Movies, trekking, site-seeing, jazz vocal performance.