ALLOCATION AND TOOLING DECISIONS IN FLEXIBLE MANUFACTURING SYSTEMS

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

BY

SELİN ÖZPEYNİRCİ

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

DECEMBER 2007

Approval of the thesis:

ALLOCATION AND TOOLING DECISIONS IN FLEXIBLE MANUFACTURING SYSTEMS

submitted by **SELİN ÖZPEYNİRCİ** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Industrial Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen ________________
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Nur Evin Özdemirel ________________
Head of Department, **Industrial Engineering**

Prof. Dr. Meral Azizoğlu ________________
Supervisor, **Industrial Engineering Dept., METU**

Examining Committee Members:

Prof. Dr. Ömer Kırca ________________
Industrial Engineering Dept., METU

Prof. Dr. Meral Azizoğlu ________________
Industrial Engineering Dept., METU

Prof. Dr. Selim Aktürk ________________
Industrial Engineering Dept., Bilkent University

Assist. Prof. Dr. Ferda Can Çetinkaya ________________
Industrial Engineering Dept., Çankaya University

Assist. Prof. Dr. Seçil Savaşaneril ________________
Industrial Engineering Dept., METU

Date: 28.12.2007

ABSTRACT

ALLOCATION AND TOOLING DECISIONS IN FLEXIBLE MANUFACTURING SYSTEMS

Özpeynirci, Selin
Ph.D., Department of Industrial Engineering
Supervisor: Prof. Dr. Meral Azizoğlu

December 2007, 111 pages

In this thesis, we consider a capacity allocation problem in flexible manufacturing systems. We assume limited time and tool magazine capacities on the Computer Numerically Controlled (CNC) machines. We have a set of operations that have to be assigned to the machines and each operation requires a set of tools to be processed. Our problem is to allocate the available capacity of the CNC machines to operations and their required tools. We consider two problems in this study: maximizing the total weight of operations where there are a limited number of tools of each type available and maximizing total weight minus total tooling cost where the tools can be used or purchased at a cost. We model the problems as Integer Linear Programs and show that they are NP-hard in the strong sense. For the total weight problem, we propose upper bounds, branch and bound algorithm for exact solutions and several heuristics for approximate solutions. For the bicriteria problem, we use Lagrangean relaxation technique to obtain lower and upper bounds. Our computational results have revealed that all solution approaches give satisfactory results in reasonable times.

Keywords: Flexible Manufacturing Systems, Capacity Allocation, Operation Assignment

ÖZ

ESNEK İMALAT SİSTEMLERİNDE PAYLAŞTIRMA VE MAKİNE UCU KARARLARI

Özpeynirci, Selin
Doktora, Endüstri Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Meral Azizoğlu

Aralık 2007, 111 sayfa

Bu tezde, esnek imalat sistemlerinde kapasite paylaştırma problemini ele aldık. Sayısal Denetimli (SD) makinelerde zaman ve makine ucu haznesi kapasitelerinde kısıtlamalar olduğunu varsaydık. Makinelere atanması gereken bir grup operasyonumuz var ve her operasyonun işlenebilmesi için bir grup makine ucu gerekmektedir. Problemimiz SD makinelerinin mevcut kapasitelerini operasyonlara ve gerektirdikleri makine uçlarına paylaştırmaktır. Bu çalışmada iki problem ele aldık: her makine ucu çeşidinden sınırlı sayıda mevcut olduğu durumda operasyonların ağırlıklarını maksimize etmek ve kullanılacak makine uçlarının belirli bir maliyetle temin edildiği durumda toplam operayon ağırlığı ile makine ucu maliyeti arasındaki farkı maksimize etmek. İki problemi de Tamsayılı Doğrusal Modeli olarak formüle ettik ve kuvvetle NP-zor olduğunu gösterdik. Toplam Ağırlık problemi için üst sınırlar, kesin çözüm için dal-sınır algoritması ve yaklaşık çözümler için sezgisel yöntemler önerdik. İki kriterli problemimiz için Lagrange gevşetim yöntemini kullanarak üst ve alt sınırlar elde ettik. Deney sonuçlarımız bütün çözüm yöntemlerimizin kısa zamanda iyi sonuçlar verdiğini göstermiştir.

Anahtar Kelimeler: Esnek İmalat Sistemleri, Kapasite Paylaştırma, Operasyon Atama

To

my parents

and

my love...

ACKNOWLEDGMENTS

I would like to thank my supervisor Prof. Dr. Meral Azizoğlu for her devoted guidance, advice, insight, comments and encouragement. I deem myself extremely fortunate to have such a great thesis supervisor as her. I always felt her support in all issues and I hope that we will be able to work together in the future.

I would like to express my thanks to Prof. Dr. Murat Köksalan for giving me the chance to work with him on many interesting and educating projects on different areas. He was a very understanding and precious guide in academic issues and beyond. Fortunately, we have a long list of future projects to work on together.

Thanks Prof. Dr. Ömer Kırca, Assist. Prof. Dr. Ferda Can Çetinkaya and Prof. Dr. Selim Aktürk for their valuable guidance and comments throughout the study.

My husband, my love, my best friend Özgür Özpeynirci was my endless support. We lived the difficulties of being a Ph.D. student together and we tried to understand, help, and encourage each other. Without him, this process would be much more difficult for me.

I strongly felt the support and faith of my parents Meral and Şener Bilgin in every stage of my life. I deem myself blessed to have their presence and love.

I am greatly indebted to Deniz Türsel Eliiyi who helped us in realizing our future plans. She patiently dealt with our concerns and was always a very supportive friend. I am very happy that I will work with her again and hopefully for a long time.

I also have felt the warm support of every person in METU IE. I am grateful to all my professors, friends and colleagues at the department for their kindness and cheerful presence. It was a great pleasure for me to be a part of this department both as a student and as an assistant.

I would like to express my gratitude to the Scientific and Technological Research Council of Turkey (TÜBİTAK) for giving me scholarship during my Ph.D. studies.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Flexible Manufacturing Systems (FMSs) are defined as integrated systems of computer numerically controlled (CNC) machines connected with automated material handling mechanisms. They combine the efficiency of a high-production transfer line and the flexibility of a job shop to best suit the batch production of mid-volume and mid-variety products. Due to these properties and highly intensive capital investment required for their implementation, flexible manufacturing has gained worldwide attention in recent years both in the manufacturing industry and in academic research. Several problems are addressed in Flexible Manufacturing System environments; some of which are part selection, system loading and operation assignment, machine loading and tool allocation. Stecke (1983) defines five major problems that should be solved in Flexible Manufacturing Systems: 1) selecting the part types for simultaneous machining, 2) grouping the machines that can process the same operations, 3) determining the production ratios for the part types, 4) determining the requirements for fixtures and pallets, and 5) allocating the tools and operations to the machines.

The main purpose of FMSs is to maintain flexibility and effective utilization of machine capacities through operation assignments and tool changeovers. Operation assignment decisions in Flexible Manufacturing Systems may be stated as assigning the operations to the NC machines subject to system-specific operational and technological constraints so as to achieve a goal or to optimize some performance criteria.

In Flexible Manufacturing Systems, tool management is another vital issue since a large number of tools are required for processing the operations and the CNC machines have limited number of tool slots. The operations can only be processed if their required tools are loaded on the machines. There are common restrictions, associated with operation assignment and tool loading decisions, such as operation-tool-machine compatibility, tool magazine capacity and available machining time. Selecting the operations to be processed by considering the tooling constraints makes the problem much more complicated.

In the literature, there are a large number of studies on Flexible Manufacturing Systems and various methodologies and systematic approaches have been proposed to solve the associated problems. The allocation problem in FMSs is stated as follows: given a fixed number of part types whose operations are to be processed on the CNC machines carrying tool magazines of limited capacity, determine the assignment of operations together with their tools. In the literature, Integer Programming (IP) has been the primary modeling approach for these problems. In a majority of the previous studies, the capacity allocation problem is analyzed for the operations and tools separately. However, little effort has been made to handle the simultaneous assignment of the operations and tools. These two assignment problems are interrelated in the sense that operations are selected according to the tools assigned and tools are placed according to the operations assigned. Our study is concerned with tool assignment to machines and time capacity allocation to operations in Flexible Manufacturing Systems. It differs from the previous research in the sense that we solve operation assignment and tool allocation problems simultaneously.

In this study, we first consider the problem of assigning operations together with their required tools to the machines in a FMS where the aim is to maximize the total weight of operation assignments. A set of operations with corresponding weights, indicating their relative importance, is given. The processing time of each operation is known. There are limitations on the number of tools of each type available in the system due to economic restrictions. Also, the number of tool slots on the tool magazine of the machines, and the capacities of the machines in terms of time units are other constraining factors. The primary decision is the selection among the operations and assignment of these to the machines. Once we select an operation, the whole inventory of that operation should be processed, i.e., operation splitting is not allowed. Moreover, the tools required for processing the operations should be loaded on the machines. In such an environment, we have to allocate the capacity of the machines to the operations and the required tools for processing those operations, so that the total weight is maximized.

We model the problem as an integer programming model and prove that it is NP-hard in the strong sense. We develop the properties of optimal solution and use them to enhance the efficiency of the upper bounding mechanisms and solution approaches. Our upper bounding procedures can be used to evaluate the performance of the approximate solutions and to curtail the size of the enumeration. A class of upper bounds uses linear programming relaxation of the problem that is strengthened by valid inequalities. We propose construction and improvement type lower bounding procedures to find powerful approximate solutions. Our constructive heuristic uses a simple greedy idea. The improvement type heuristics are of two types: best improving search and tabu search. We also propose a beam search procedure that is based on curtailed branch and bound idea, and obtain near optimal solutions in reasonable time for large size problem instances. For exact solutions, we present a branch and bound algorithm and find optimal solutions for moderate size problems.

In the second part of this study, we consider the minimization of total tool costs as well. In doing so, we study the trade-offs between the profit gained over operation assignments and cost incurred by tool usages. In this problem we assume that there are no limitations on the number of tools and we have to pay a cost for the usage or purchase of each tool. Other constraints of this bicriteria problem are the same as the total weight problem. We model this problem as an integer programming model. We propose Lagrangean relaxation based lower and upper bounds for near optimal solutions.

This thesis consists of 7 chapters, which are organized as follows: In Chapter 2, we give the definition and mathematical model of the total weight problem and present some previous studies in the literature. In Chapter 3, we define optimality properties and upper bounding procedures. We explain branch and bound algorithm for exact solutions in Chapter 4 and heuristics for approximate solutions in Chapter

5. In Chapter 6, we define the bicriteria problem and its solution approach. We conclude the study in Chapter 7.

CHAPTER 2

TOTAL WEIGHT PROBLEM

In this chapter, we first define our total weight problem with its underlying assumptions. We then give the mathematical model of the problem and discuss its computational complexity. Later, we present the previous studies on operation assignment and tool allocation problem.

2.1 Problem Definition

Consider n operations that are to be processed by a set of m CNC machines. An operation is assigned to at most one machine, and when assigned, it should be processed for P_i time units without preemption. We let w_i be the weight of operation i which may represent the profit brought or the assignment cost when it is negative.

The machines are flexible in the sense that they function according to the tools loaded on their tool magazines. Machine j has a tool magazine capacity of s_j tool slots and has a time capacity of C_j time units. P_i and C_j are measured in same units, say in the minutes.

The cutting tools are available in limited numbers due to the technological restrictions or budget limitations. There are t tool types and r_k tools of type k are available. Each tool uses one slot of the tool magazine. A set of tools $l(i)$ should be available on the tool magazine before processing operation i.

The problem is to allocate the time capacity of the machines to operations and their tool capacities to tools, so as to maximize the total weight of operations. In this study, we consider the batching problem. In batching problem, the operations to be processed are selected, their required tools are loaded on the tool magazines of the machines and the processing continues until all the selected operations are completed. Tools cannot be changed and new operations cannot be introduced. Then, a new batch is selected and processed.

Throughout this study, we make the following additional assumptions:

- Each machine can process each operation.
- An operation cannot be processed by more than one machine, i.e., operation splitting is not allowed.
- The tools are not shared between the machines, i.e., the loaded tools are not removed during the processing.
- The tool magazines of the machines are initially empty.
- All parameters, i.e., w_i, P_i, C_j, r_k, s_j, $l(i)$ are known with certainty, i.e., the system is deterministic.
- The set of operations, machines and tools are available at time zero and are not subject to change, i.e., the system is static.
- Each tool occupies one tool slot.
- The tools never breakdown and they have infinite life.
- The tools have negligible weight.
- The tool loading times are negligible.
- The machines and tools are continuously available.

2.2 Mathematical Model

In this section, we first define our indices, parameters and decision variables. Then, we give the mathematical representation of the problem. For the sake of compactness of the model presentation, we restate the indices and parameters.

The indices are:

i:	operation index
j:	machine index
k:	tool index

The parameters are:

n:	number of operations
m:	number of machines
t:	number of tool types
w_i:	weight of operation *i*
P_i:	processing time of operation *i*
C_j:	capacity of machine *j*
$l(i)$:	set of tools required to process operation *i*
s_j:	number of tool slots of machine *j*
r_k:	number of tool type *k* available

The decision variables are:

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

$$Z_{kj} = \begin{cases} 1 & \text{if tool } k \text{ is loaded on machine } j \\ 0 & \text{otherwise} \end{cases}$$

The objective function of our problem requires the maximization of the total weight, which can be expressed as:

$$\text{Maximize} \quad \sum_{i=1}^{n}\sum_{j=1}^{m} w_i X_{ij} \tag{2.1}$$

The constraints are:

- Operation *i* can be assigned to at most one machine.

$$\sum_{j=1}^{m} X_{ij} \le 1 \qquad \forall i \tag{2.2}$$

- The total processing time of the operations assigned to machine j can not exceed its time capacity.

$$\sum_{i=1}^{n} P_i X_{ij} \leq C_j \qquad \forall j \tag{2.3}$$

- The total number of tools mounted on machine j can not exceed its tool slot capacity.

$$\sum_{k=1}^{t} Z_{kj} \leq s_j \qquad \forall j \tag{2.4}$$

- The total number of tool type k loaded cannot exceed its availability.

$$\sum_{j=1}^{m} Z_{kj} \leq r_k \qquad \forall k \tag{2.5}$$

- An operation can be assigned to a machine only if its set of required tools are already loaded on that machine.

$$X_{ij} \leq Z_{kj} \qquad \forall i,j,k \in l(i) \tag{2.6}$$

- X_{ij}'s and Z_{kj}'s are set to 0 or 1.

$$X_{ij}, Z_{kj} \in \{0,1\} \qquad \forall i,j,k \tag{2.7}$$

Our problem reduces to the well-known multiple knapsack problem in the absence of the tooling constraints, (2.4)–(2.6). The multiple knapsack problem is strongly NP-hard (Martello and Toth, 1990). So is our problem, with an additional complexity brought by the tooling decisions.

2.3 Literature Review

In the literature, there are several studies that are pertinent to the total weight problem. In the absence of tooling constraints, the problem is the well-known multiple knapsack problem. In the absence of the tooling constraints but under the assumption of the limited number of operations per machine, the problem resembles the capacity allocation problem encountered in semiconductor manufacturing. Moreover, the part selection and machine loading problems in FMSs are relevant to our problem environment. We now discuss the previous studies on these related problems.

2.3.1 The Multiple Knapsack Problem

In the 0-1 Multiple Knapsack Problem, there are *n* items and *m* knapsacks with specified profit and weights of items and capacities of knapsacks given. The aim is to place the items into the knapsacks without exceeding their capacities so that the total weight of the selected items is maximized. Our problem is similar to 0-1 multiple knapsack problem without tooling constraints where operations can be considered as items and machines can be considered as knapsacks. If we assume that the tools are already loaded on the machines, our problem is to select the operations to be processed. 0-1 multiple knapsack problem is strongly NP-hard. Martello and Toth (1990) and Kellerer *et al*. (2004) give upper bounds, exact and approximate algorithms for this problem. Surrogate relaxation and Lagrangean relaxation are used to find upper bounds. Martello and Toth (1990) mention that dynamic programming is not practical for this problem and suggest the branch and bound algorithm to obtain exact solutions. They also propose some methods useful to reduce the problem size. Pisinger (1999) presents a recursive branch and bound algorithm that employs surrogate relaxation for the upper bound and reduction mechanisms. The lower bound is obtained by splitting items that appear in the solution of surrogate relaxation into knapsacks. This is done by solving subset sum problems. Kellerer *et al.* (2004) define variants of the problem such as the multiple knapsack problem with assignment constraints and the class-constrained multiple knapsack problem.

2.3.2 The Class-Constrained Multiple Knapsack Problem

The class-constrained multiple knapsack problems (CMKPs) are introduced by Shachnai and Tamir (2001a and 2001b). In the class-constrained multiple knapsack problem, items are identified not only by their weights and profits but also by their colors. Each knapsack has a limited number of compartments and items of different colors cannot be placed in the same compartment. Our problem can be reduced to the class-constrained multiple knapsack problem when we consider the tools used by the operations as their colors and the tool slots as the compartments of knapsacks (machines). The tool slot capacities correspond to the number of compartments. This similarity is valid for the case where all operations require only one tool, i.e., $|l(i)|=1$ for all i.

Shachnai and Tamir (2001a) assume all items have the same size (weight) and value (profit). They also assume that the total number of items is equal to the total sum of the knapsack volumes (capacities). But they show the validity of their results for the general problem settings. They study two NP-hard problems: the class-constrained multiple knapsack problem where they aim to maximize the total size of the packed items, and the fair placement problem (FPP) where the fraction of the items packed is to be maximized. In that study, they try to find a perfect placement that solves both problems optimally. These problems arise in storage management systems. They use greedy procedures to fill the knapsacks. They show that adding one compartment eliminates the gap between the optimal solution and the solution of any polynomial time algorithm. Their dual approximation algorithm finds an infeasible solution, and the degree of infeasibility measures the performance of the result.

Shachnai and Tamir (2001b) propose a polynomial time approximation scheme for the class-constrained multiple knapsack problem based on the algorithm of Chekuri and Khanna (2000).

Dawande *et al.* (2000) study the multiple knapsack problem with assignment restrictions in which the items can be assigned to only a set of knapsacks. They first consider the objective of maximizing the total assigned weight. They suggest several ways for finding feasible solutions: a greedy algorithm, successively solving single knapsack problems, and rounding the linear programming relaxation solution. Then they try to maximize total assigned weight and minimize utilized capacity simultaneously. They modify the successive knapsack algorithm for the bicriteria problem.

Marques and Arenales (2007) consider a knapsack problem with different compartments that have flexible capacities that are lower and upper bounded. A fixed cost should be paid to include each compartment depending on the type of the items that it contains. Also each compartment uses a part of the capacity of the knapsack. The problem is to determine the compartments and their suitable capacities so that the total utility minus the total cost of the compartments is minimized. The authors develop some heuristics for approximate solution of the problem.

2.3.3 The Capacity Allocation Problem

The studies in this section consider operation assignments, but ignore the tooling constraints.

Toktay and Uzsoy (1998) address a capacity allocation problem in a semiconductor wafer fabrication facility. They study allocating available machine capacity at a work center where different operations are to be processed. They set an upper limit on the number of the setups allowed for each machine over a shift. Also they put a restriction on the number of different machines that can process the same setup simultaneously because of the technological limitations. They formulate the problem as a maximum flow problem on a bipartite network with integer side constraints. They consider two objective functions: 1) maximizing throughput, i.e., the total amount of WIP processed at the work center during the shift and 2) minimizing the total deviation from predetermined production goals. They show that two objective functions are equivalent and the associated problems are NP-hard in the strong sense. They develop two heuristic procedures and report that their heuristics perform very well under different problem settings.

A similar study is due to Akçalı *et al.* (2005) who consider a work center with parallel machines so as to maximize the total throughput. They assume that an operation can be processed by only a subset of the machines. The number of tools available for an operation and the number of setups that can be performed on a machine are limited. They interpret the problem as a maximum flow problem with degree constraints, which is shown to be strongly NP-hard by Toktay and Uzsoy (1998). They develop several constructive heuristics and improve the heuristics by a local search approach. They implement their heuristics on several problem settings and observe their satisfactory performance.

Shanker and Srinivasulu (1989) study the loading problem in a flexible manufacturing environment. Their problem is to select a subset of jobs and assign them to machines so as to maximize the total workload. They formulate the problem as a mixed integer program and develop a two-stage branch and bound procedure. They also develop three heuristic procedures for the bicriteria problem of balancing the workload and maximizing the throughput. They apply their heuristics to the data generated by Shanker and Tzen (1985) and observe that their heuristics produce better results than that of Shanker and Tzen (1985). Shanker and Tzen (1985) consider a bicriteria problem of workload balancing among the machines and meeting the job due dates for a FMS with random job arrivals and stochastic operation times.

Swarnkar and Tiwari (2004) study the machine loading problem with two objectives: minimizing system unbalance and maximizing the throughput subject to the machine time and tool slot constraints. They propose a hybrid algorithm based on tabu search and simulated annealing and benefit from the advantages of both algorithms.

2.3.4 The Capacity Allocation and Tool Loading Problem

The studies discussed in this section deal with selecting the operations to be processed as well as loading the required tools to the machines.

Bilgin and Azizoğlu (2006) consider the operation and tool allocation problem in FMSs where operation splitting is allowed. They prove that their problem is strongly NP-hard and develop several heuristic procedures, one of which is based on Lagrangean relaxation that give near optimal solutions in a very short time. They show that linear programming relaxation dominates the Lagrangean relaxation upper bound.

D'Alfonso and Ventura (1995) study the tool assignment problem where the machines have limited tool slots in their tool magazines and tools require multiple slots. The objective is to minimize the number of daily production travels between the machines. They examine two algorithms: one is a Lagrangean relaxation approach with a subgradient optimization technique, the other is a graph theoretic heuristic. Their computational results reveal that the subgradient algorithm is superior to their heuristic algorithm in most of the problem settings.

Liang and Dutta (1993) propose an integrated approach for the simultaneous solution of the part selection and machine loading problems in Flexible Manufacturing Systems. They consider two objectives in a hierarchy. Their primary objective is to select and load a subset of parts such that the system output or the utilization of FMS productivity is maximized. Their secondary objective is obtaining the maximum system output with less input by either reducing processing cost or reducing makespan. They develop models for both objectives and show that as the size of the mixed integer program increases, the problem becomes very difficult to solve. They propose a solution method based on Lagrangean relaxation and develop a Lagrangean heuristic. Their computational results show that all problems reach an acceptable percentage error within a reasonable time.

Ventura *et al.* (1988) represent the tool loading problem in a Flexible Manufacturing System as an assignment model. The objective is to minimize the time span required to process all parts in a batch. They consider tool magazine capacity constraints, multiple slots for some tools and machine dependent tool processing times. They model two problems: the first one assumes that any machine can accommodate any tool and the tool processing times are independent of the machines, whereas the second model assumes machine dependent tool processing times. They develop several greedy heuristics for the first model, and six heuristic algorithms for the second model. They test the performance of their heuristics on two hypothetical cases, and choose the best performing algorithms for each model.

Chen *et al.* (1995) define the part selection problem in FMSs as the selection of the most cost effective set of parts to be processed simultaneously. Their objective is to maximize the total profit brought by the set of selected parts during the next production horizon. They develop a zero-one integer program and two heuristic algorithms. Their heuristic algorithms divide the part selection procedure into two stages where one stage deals strictly with the limitations on the machining time, the storage capacity and the Automated Guided Vehicle (AGV) time and the other stage uses three different strategies to choose a set of parts with respect to the tooling and fixture constraints. Their computational experiments with the heuristics reveal that satisfactory solutions can be found in a reasonably short time.

Ram *et al.* (1990) use a network representation with simple side constraints for modeling the machine loading and tool allocation problem in a FMS, where the objective is to minimize the total cost of operation assignments. They solve a sample problem using the branch and bound procedure and obtain the optimal solution in a short time.

Berrada and Stecke (1986) study the problem of assigning the tools, operations and the associated cutting tools required for the part types selected for simultaneous production. This assignment is constrained by each machine's tool magazine capacity as well as by the production capacities of the overall system and for each individual machine type. Their objective is to balance the machine workloads. They apply the branch and bound algorithm and discuss its performance under different problem cases and mention that the solutions can be found in a short time.

Sodhi *et al.* (1994) study the part selection problem to determine tool allocations and the production schedule for meeting the production plan. Their objective is to minimize the total cost. They assume constraints on the tool magazine capacity and the tool magazine changeover frequency. They propose a heuristic algorithm and observe its performance under different cases.

Hso and De Matta (1997) aim to detect the infeasibility of a loading problem. There are several decisions to be taken in FMSs that are interdependent. However, since these problems are generally solved separately, a decision made at a level may be infeasible at another one. They consider the problem of loading operations and tools simultaneously so that the total cost of processing the operations plus the penalty of not assigning the operations is minimized. They assume that for each operation, a set of tools is chosen among some alternatives. They use Lagrangean relaxation to obtain a lower bound and a Lagrangean heuristic to find feasible solutions. They divide the relaxed problem into subproblems and employ subgradient optimization to update the Lagrange multipliers. Using an upper and a lower bound, they show that, if certain conditions are satisfied, there is no feasible loading solution where all operations are assigned.

Denizel and Erengüç (1997) address part type selection and lot sizing problems in FMSs. They first present a single machine problem and then extend the model to the multiple machines. They propose a branch and bound algorithm for finding exact solutions.

Gray *et al.* (1993) discuss several issues related to tool management problem in FMSs. They mention the importance and complexity of tool management problem and introduce some decision problems in the tool, machine and system levels.

Grieco *et al.* (2001) provide a review on the loading problem in FMSs. They identify several characteristics of FMS environments and the corresponding models and approaches.

The first part of our study will consider the capacity allocation and loading problem discussed in Bilgin and Azizoğlu (2006) but with no operation splitting case. In the second part, we will consider a bicriteria problem that evaluates trade-offs between maximum total weight and minimum total cost.

CHAPTER 3

OPTIMALITY PROPERTIES AND UPPER BOUNDS FOR THE TOTAL WEIGHT PROBLEM

In this chapter, we first define optimality properties. Next, we present upper bounding approaches and discuss the results of the computational experiments.

3.1 Properties of Optimal Solution

In this section, we present some properties of the optimal solution. These properties arise from the dominance relations between operations and sets of operations.

We say that operation a dominates operation b if operations a and b satisfy the conditions of the below property.

Property 3.1 If $P_a \leq P_b$ and $w_a > w_b$ and $l(a) \subseteq l(b)$ then if operation b is assigned, operation a should also be assigned, i.e., $\sum_j X_{bj} = 1$ implies $\sum_j X_{aj} = 1$.

Proof Assume schedule S does not satisfy the conditions of the property, i.e. $P_a \leq P_b$, $w_a > w_b$ and $l(a) \subseteq l(b)$ and $\sum_j X_{bj} = 1, \sum_j X_{aj} = 0$.

Now let $\sum_j X_{bj} = 0$ and $\sum_j X_{aj} = 1$, keep other assignments as in S and get schedule $S¢$. $S¢$ is feasible as operation a can feasibly fit to the place vacated by operation *b* ($P_a \leq P_b$ and $l(a) \subseteq l(b)$). The total weight of S' is $w_a - w_b > 0$ units more than that of *S*. Hence, *S*, i.e., a schedule that does not satisfy the conditions of our property cannot be optimal. □

We implement the same idea to a set of operations and state them through the following two properties.

Property 3.2 Suppose there is a subset of operations, *G*, such that $w_i > \sum_{d \in G} w_d$, $P_i \leq \sum_{d \in G} P_d$, and the operations in *S* are already assigned to a particular machine. If operation *i* can fit to the machine without extra tool insertion, then operation *i* should be assigned.

Proof Suppose we have a solution with total weight W_1 where operations in set *G* are processed while operation *i* is not processed. If we remove operations in set *G* from the machine they are assigned to, say machine *j*, and insert operation *i* in place of them, this exchange will give a feasible solution. Let c_j' be the remaining time capacity of machine *j* in our current solution. Then $c_j' + \sum_{d \in G} P_d - P_i \geq 0$ since $P_i \leq \sum_{d \in G} P_d$, i.e., we do not exceed the time capacity of machine *j*. Also, we do not need to load additional tools for operation *i*. Hence, tooling constraints are also satisfied. After the exchange, total weight becomes

$W_2 = W_1 - \sum_{d \in G} w_d + w_i$ and clearly $W_2 > W_1$ since $w_i > \sum_{d \in G} w_d$.

Therefore, a solution that includes operations in set *G* but not operation *i*, cannot be optimal. □

A similar situation can be observed if a set of operations dominate a single operation.

Property 3.3 Suppose there is a subset of operations, *L*, such that $\sum_{d\in L} w_d > w_i$, $\sum_{d\in L} P_d \le P_i$ and $\bigcup_{d\in L} l(d) \subseteq l(i)$. If operation *i* is processed, then at least one operation in set *L* should be processed.

Proof Suppose there exists a solution where operation *i* is processed but none of the operations in set *L* is assigned to any machine. We can remove operation *i* from the machine and assign all operations in set *L* to the place vacated by operation *i*. This exchange will give a feasible solution since $\sum_{d\in L} P_d \le P_i$ and operations in set *L* do not require extra tool. The objective function will increase by $\sum_{d\in L} w_d - w_i > 0$. Therefore, a solution that includes operation *i* but not any operation in set *L* cannot be optimal. □

3.2 Upper Bounding Procedures

In this section, we present five upper bounds that rely on the relaxation of some constraints of the model. Note that any relaxation to our maximization problem provides an upper bound. Next, we discuss the results of our computational experiments.

3.2.1 Upper Bound 1, UB$_1$: Linear Programming (LP) Relaxation

UB$_1$ is obtained by restating the operation-tool assignment link constraints and relaxing the integrality constraints on X_{ij}s and Z_{kj}s. We assume

$$0 \le X_{ij} \le 1 \qquad \forall i, j$$
$$0 \le Z_{kj} \le 1 \qquad \forall k, j$$

and restate constraint (2.6) in the following form

$$\sum_{i\in a(k)} X_{ij} \le |a(k)|\, Z_{kj}, \forall k, j$$

where $a(k)$ is the set of operations requiring tool *k*.

We show through Theorem 3.1 that the optimal values of the tooling variables are available and do not constrain the operation assignment decisions.

Theorem 3.1 There exists an optimal LP-relaxed solution in which $Z_{kj}^{*} = \frac{\sum_{i \in a(k)} X_{ij}}{|a(k)|} \quad \forall k, j$ where $a(k)$ is the set of operations requiring tool k.

Proof The restated constraint

$$\sum_{i \in a(k)} X_{ij} \leq |a(k)| Z_{kj}, \forall k, j \tag{3.1}$$

follows that

$$Z_{kj} \geq \frac{\sum_{i \in a(k)} X_{ij}}{|a(k)|}, \forall k, j \tag{3.2}$$

Assume there is an optimal solution where

$$Z_{kj}^{'} > \frac{\sum_{i \in a(k)} X_{ij}}{|a(k)|}, \forall k, j \tag{3.3}$$

Let there be another solution Z_{kj}^{*} where Z_{kj}^{*}s are defined as in (3.4)

$$Z_{kj}^{*} = \frac{\sum_{i \in a(k)} X_{ij}}{|a(k)|}, \forall k, j \tag{3.4}$$

Accordingly, $Z_{kj}^{'} > Z_{kj}^{*}$. As $Z_{kj}^{'}$ is feasible, it satisfies the constraints (2.4) and (2.5). Z_{kj}^{*} also satisfies (2.4) and (2.5) since $\sum_{k=1}^{t} Z_{kj}^{*} \leq \sum_{k=1}^{t} Z_{kj}^{'}$ and $\sum_{j=1}^{n} Z_{kj}^{*} \leq \sum_{j=1}^{n} Z_{kj}^{'}$. This means that Z_{kj}^{*} is feasible as well. The objective function $\sum w_i X_{ij}$ is free of Z_{kj}, hence Z_{kj}^{*} is optimal as well. □

Theorem 3.1 implies that the tooling constraints are redundant in the LP-relaxed problem, hence it reduces to the classical continuous multiple knapsack problem. The optimal solution of the continuous multiple knapsack problem orders the operations in their nonincreasing order of w_i/P_i values and assigns them to one of the available machines (see Martello and Toth, 1990). This solution is equivalent to the surrogate relaxation of the multiple knapsack problem where a single machine with capacity $C = \sum_{j=1}^{m} C_j$ is assumed.

We hereafter let UB_1 denote the maximum weight of the LP relaxation solution.

3.2.2 Upper Bound 2, UB_2: Strengthened Linear Programming Relaxation

We can improve UB_1, by adding constraints that are valid for the original problem, but are not necessarily satisfied in the absence of integrality requirements on X_{ij}s and Z_{kj}s. We define several such constraints below.

1) In the LP relaxation, operations can be split so as to fill the remaining capacities of the machines. However, in the original problem, if the processing time of an operation i, P_i, exceeds the time capacity of a machine j, C_j, operation i cannot be assigned to machine j. Therefore, we add the constraint

$$X_{ij}=0 \text{ if } P_i > C_j \tag{3.5}$$

to the LP model and avoid assignment of operation i to machine j.

2) If the total time required to process two operations exceeds the time capacity of machine j, then those two operations cannot be assigned to machine j at the same time. This constraint expressed below may not be satisfied by the LP model, hence constitutes a valid cut.

$$X_{ij} + X_{gj} \leq 1 \qquad \forall i \text{ and } g \text{ where } P_i + P_g > C_j \tag{3.6}$$

3) If the set of tools required by operation i exceeds the tool magazine capacity of machine j, operation i cannot be assigned to machine j. Therefore, we add the constraint

$$X_{ij} = 0 \text{ if } |l(i)| > s_j \tag{3.7}$$

to the LP model and avoid assignment of operation i to machine j.

4) If the set of tools required by two operations *i* and *g* exceeds the tool magazine capacity of machine *j*, these two operations cannot be assigned to machine *j* together. We add this condition to the LP model through the following constraint:

$$X_{ij} + X_{gj} \leq 1 \qquad \forall i \text{ and } g \text{ where } |l(i) \cup l(g)| > s_j \tag{3.8}$$

The above idea may be extended to three operations *i*, *g* and *d* case as follows:

$$X_{ij} + X_{gj} + X_{dj} \leq 1 \qquad \forall i, g \text{ and } d \text{ where } |l(i) \cup l(g)| > s_j, |l(i) \cup l(d)| > s_j, |l(d) \cup l(g)| > s_j \tag{3.9}$$

5) In the original problem, even when the least loaded operations are assigned to a machine, the number of operations that the machine can process is limited.

We let [*r*] be the index of the operation having the *r*th smallest processing time. Assume n_j is defined such that

$$\sum_{r=1}^{n_j} P_{[r]} \leq C_j \quad \text{and} \quad \sum_{r=1}^{n_j+1} P_{[r]} > C_j \tag{3.10}$$

Hence n_j is a valid upper bound on the number of operations that can be processed on machine *j*, and the following constraint limits the number of operations on each machine:

$$\sum_{i=1}^{n} X_{ij} \leq n_j \qquad \forall j \tag{3.11}$$

The above constraint may also be written for two machines case. We define $n_{j1,j2}$ such that

$$\sum_{r=1}^{n_{j1,j2}} P_{[r]} \leq C_{j1} + C_{j2} < \sum_{r=1}^{n_{j1,j2}+1} P_{[r]} \tag{3.12}$$

Hence $n_{j1,j2}$ is a valid upper bound on the number of operations on machines j_1 and j_2. The following example illustrates n_j computations.

Example 3.1 Suppose we have five operations with the following data

i	1	2	3	4	5
P_i	2	5	4	10	8
$P_{[i]}$	2	4	5	8	10

There are two machines with time capacities C_1=7 and C_2=12. Note that

$$P_{[1]} + P_{[2]} < C_1 < P_{[1]} + P_{[2]} + P_{[3]} \Rightarrow n_1 = 2$$
$$P_{[1]} + P_{[2]} + P_{[3]} < C_2 < P_{[1]} + P_{[2]} + P_{[3]} + P_{[4]} \Rightarrow n_2 = 3$$

In other words, n_1=2 satisfies $\sum_{i=1}^{n_1} P_{[i]} \le C_1 < \sum_{i=1}^{n_1+1} P_{[i]}$ and n_2=3 satisfies $\sum_{i=1}^{n_2} P_{[i]} \le C_2 < \sum_{i=1}^{n_2+1} P_{[i]}$. Hence the number of operations that can be processed by machines 1 and 2 cannot exceed 2 and 3, respectively.

Similarly, the upper bound for two machines is

$$P_{[1]} + P_{[2]} + P_{[3]} + P_{[4]} = C_1 + C_2 < P_{[1]} + P_{[2]} + P_{[3]} + P_{[4]} + P_{[5]} \Rightarrow n_{1,2} = 4,$$

i.e., $n_{1,2}$=4 satisfies $\sum_{i=1}^{n_{12}} P_{[i]} \le C_1 + C_2 < \sum_{i=1}^{n_{12}+1} P_{[i]}$.

We generalize the above reasoning to a subset of machines, say R, and let n_R denote an upper bound on the number of operations that can be processed by the machines in R. Note that n_R satisfies

$$\sum_{r=1}^{n_R} P_{[r]} \le \sum_{j \in R} C_j < \sum_{r=1}^{n_R+1} P_{[r]} \qquad (3.13)$$

The generalized constraint should be written as below:

$$\sum_{j \in R} \sum_{i=1}^{n} X_{ij} \le n_R \qquad (3.14)$$

We further reduce n_j using the tool requirement set and tool magazine capacity information.

Consider two operations a and b such that $P_a \le P_b$ and $|l(a) \cup l(b)| > s_j$. The latter condition implies that both operations a and b cannot be processed on machine j, hence they should not appear in n_j computations simultaneously. To guarantee an upper bound, one can consider P_a, but not P_b, in computing n_j.

Accordingly, we compute $n¢_j$ as follows:

$$\sum_{\substack{r=1 \\ [r] \ne b}}^{n'_j} P_{[r]} \le C_j < \sum_{\substack{r=1 \\ [r] \ne b}}^{n'_j+1} P_{[r]} \qquad (3.15)$$

Hence $n¢_j$ is an upper bound on the number of operations that can be processed on machine j.

Example 3.2 Consider Example 3.1. Suppose $l(2)=\{1, 2, 3\}$, $l(3)=\{3, 4, 5\}$, $s_1 = s_2 = 3$. Operations 2 and 3 cannot be assigned to the same machine as $|l(2)\cup l(3)|=5$. While computing n_j we should consider operation 3, but not operation 2, since it has a higher P_i value.

$$P_{[1]} + P_{[2]} < C_1 < P_{[1]} + P_{[2]} + P_{[4]} \Rightarrow n_1' = 2$$
$$P_{[1]} + P_{[2]} < C_2 < P_{[1]} + P_{[2]} + P_{[4]} \Rightarrow n_2' = 2$$

Note that when we exclude operation 2, n'_2 decreases, which results in a tighter bound.

Now assume three operations a, b and c such that $P_a \le P_b \le P_c$, $|l(a)\cup l(b)|>s_j$, $|l(b)\cup l(c)|>s_j$ and $|l(a)\cup l(c)|>s_j$. These conditions altogether imply that only one of the operations in set $\{a, b, c\}$ can be performed on machine j. To guarantee an upper bound, P_a, but not P_b and P_c, is considered in n'_j computations. Accordingly, we compute n'_j as follows:

$$\sum_{\substack{r=1 \\ [r]\neq b,c}}^{n'_j} P_{[r]} \le C_j < \sum_{\substack{r=1 \\ [r]\neq b,c}}^{n'_j+1} P_{[r]} \qquad (3.16)$$

We generalize the above reasoning for a subset I such that $|l(a)\cup l(b)|>s_j$ for all operation pairs (a, b) in I. Note that only one operation in set I can be performed on machine j. To guarantee an upper bound, the operation having the smallest processing time, say operation I^1, is kept and the others are ignored. That is, we compute n'_j given I as follows:

$$\sum_{\substack{r=1 \\ [r]\notin I\setminus I^1}}^{n'_j} P_{[r]} \le C_j < \sum_{\substack{r=1 \\ [r]\notin I\setminus I^1}}^{n'_j+1} P_{[r]} \qquad (3.17)$$

Assume $I_1\cup I_2\cup\ldots\cup I_\alpha=\{1,\ldots,n\}$ and $I_q\cap I_p=\emptyset$ for all subset pairs (q, p). I_i may have a single operation. Let I_i^1 denote the operation having smallest processing time in set I_i. An upper bound on the number of operations is found by considering at most one operation from I_i. Only selecting I_i^1 from I_i guarantees such an upper bound. The resulting expression is as given below:

$$\sum_{\substack{r=1 \\ r|r\in \bigcup_{i=1}^{a} I_i^1}}^{n'_j} P_{[r]} \le C_j < \sum_{\substack{r=1 \\ r|r\in \bigcup_{i=1}^{a} I_i^1}}^{n'_j+1} P_{[r]} \qquad (3.18)$$

Once we obtain n'_j for each machine, we can write the following constraint:

$$\sum_{i=1}^{n} X_{ij} \le n'_j \quad \forall j \qquad (3.19)$$

We illustrate n'_j computations via the following example problem.

Example 3.3 Assume $l(1)$ = {2, 4} in Example 3.2. So $|l(1)\cup l(2)|=4>s_j$ and $|l(1)\cup l(3)|=4>s_j$. Any two operations in set {1, 2, 3} cannot be processed together. We ignore operations 2 and 3 and keep only the operation with smallest processing time, i.e., operation 1.

$$P_{[1]} < C_1 < P_{[1]} + P_{[4]} \Rightarrow n'_1 = 1$$
$$P_{[1]} + P_{[4]} < C_2 < P_{[1]} + P_{[4]} + P_{[5]} \Rightarrow n'_2 = 2$$

UB_2 is found by incorporating the constraints (3.5), (3.6), (3.8), (3.9), (3.14) and (3.19) into the LP relaxation of the original model.

3.2.3 Upper Bound 3, UB$_3$: Partial Linear Programming Relaxation

Consider the continuous relaxation of the operation assignment variables, X_{ij}s, while keeping the tool assignment variables, Z_{kj}s, integer. The resulting problem is the continuous capacity and tool allocation problem studied in Bilgin and Azizoğlu (2006). As shown in Bilgin and Azizoğlu (2006), the associated problem is strongly NP-hard. There are $m\times n$ operation assignment variables, X_{ij}s, and $m\times t$ tool assignment variables, Z_{kj}s. Therefore, it is easier to solve than our problem when $t<n$ and in many practical applications, the number of tools is much lower than the number of operations.

We let UB_3 denote the maximum weight of the problem with continuous operation assignment variables. Note that UB_3 dominates UB_1, as it additionally requires integral tool assignment variables.

3.2.4 Upper Bound 4, UB$_4$: Strengthened Partial Linear Programming Relaxation

One can also add some constraints that state some properties of the optimal solution and improve UB$_3$.

Constraint sets (3.5), (3.6), (3.8), (3.9), (3.14) and (3.19) may not be satisfied when only X_{ij}s are relaxed. Hence the incorporation of those constraints may improve the quality of UB$_3$. Moreover we use Property 3.1 defined in Section 3.1 and its result to generate a relation between operation and tool assignments. We introduce the constraint

$$X_{ij}=Z_{kj} \text{ where } k\in l(i) \tag{3.20}$$

if any of the following conditions hold:

i. $k\notin l(r)\ \forall r\neq i$

ii. $k\notin l(r)\ \forall r$ not dominated by operation i

UB$_4$ relaxes the integrality of X_{ij}s and incorporates constraints (3.5), (3.6), (3.8), (3.9), (3.14), (3.19) and (3.20) into the original model. Hence it dominates UB$_3$ which only relaxes the integrality of X_{ij}s.

3.2.5 Upper Bound 5, UB$_5$: Cardinality Based Upper Bound

We let N^* be the number of operations that can be processed over all machines in an optimal solution and UB(N^*) be an upper bound on N^*. UB(N^*) can be found, using the upper bounds on the number of operations that can be processed on each machine j, n'_j. Simply

$$N^*\leq\sum_{j=1}^{m}n_j^{'}=N_1\ .$$

We can also derive another upper bound on N^* by aggregate capacity considerations. Using the ideas followed in n'_j computations, we can say that N_2 that satisfies

$$\sum_{r=1}^{N_2}P_{[r]}\leq\sum_{j=1}^{m}C_j<\sum_{r=1}^{N_2+1}P_{[r]}$$

is a valid upper bound on N^*. Hence $N^* \leq \text{Min}\ \{N_1, N_2\} = \text{UB}(N^*)$.

We illustrate UB(N^*) computations through the following examples.

Example 3.4 Consider three operations and two machines. The tool magazines have unlimited capacities and there are no less than m tools of each tool type. Hence the tooling constraints are redundant.

i. P_i=2 for i=1, 2, 3 and C_j=3 for j=1, 2. Each machine can process one operation but not two. The upper bounds on the number of operations that can be processed by each machine are n_1=n_2=1. Therefore $N_1 = n_1 + n_2 = 2$.

If we consider the aggregate capacity C_1+C_2=6, we can assign all three operations, i.e., N_2=3. Hence, UB(N) = Min $\{N_1, N_2\} = N_1 = 2$.

ii. P_1=1, P_2=2, P_3=3 and C_1=3, C_2=1. The operations are already given in the increasing order of P_i values. The upper bounds for each machine are:

$$\sum_{i=1}^{2} P_i = 3 = C_1 < \sum_{i=1}^{3} P_i = 6 \Rightarrow n_1 = 2,$$

$$P_1 = 1 = C_2 < \sum_{i=1}^{2} P_i = 3 \Rightarrow n_2 = 1.$$

Then, we have $N_1 = n_1 + n_2 = 3$. On the other hand, when we consider the aggregate capacity,

$$\sum_{i=1}^{2} P_i = 3 < \sum_{j=1}^{2} C_j = 4 < \sum_{i=1}^{3} P_i = 6 \Rightarrow N_2 = 2.$$

This follows, UB(N^*) = Min $\{N_1, N_2\} = N_2 = 2$.

Having found UB(N^*) value, we can find an upper bound on the total weight. We let SN be the set of operations selected by the optimal solution.

Note that $\sum_{i \in SN} w_i \leq \sum_{i=1}^{|SN|} w_{[i]}$ where $w_{[i]}$ is the ith largest weight.

$$\sum_{i=1}^{|SN|} w_{[i]} \leq \sum_{i=1}^{UB(N^*)} w_{[i]} \quad \text{as } |SN| \leq \text{UB}(N^*).$$

Hence UB$_5$ $= \sum_{i=1}^{UB(N^*)} w_{[i]}$ is a valid upper bound on the optimal total weight.

To illustrate UB$_5$ computations, we assume the data given in Example 3.4 ii. Let w_1=1, w_2=2, w_3=3, hence $w_{[1]}$=3, $w_{[2]}$=2, $w_{[3]}$=1, UB(N^*) =2. This follows,

$$\text{UB}_5 = \sum_{i=1}^{UB(N^*)} w_{[i]} = w_{[1]} + w_{[2]} = 5.$$

UB$_1$ and UB$_5$ do not require any commercial software on mathematical programming; hence they can be effectively used in evaluating the partial solutions of an implicit enumeration technique. The other bounds are more powerful, however harder to obtain, as they require optimal model solutions by a commercial software. The bounds can be used to evaluate the performances of the heuristic procedures.

3.2.6 Computational Experiments

We design an experiment to test the performances of our upper bounding procedures. We set the number of operations, n, to 25, 50, 75, and 100 and the number of machines, m, to 3, 5, and 7 in our experiments. We set the number of tool types, t, to 8 and 16.

We generate two cases for the cardinality of the tool requirement sets. In Case 1, we set $|l(i)|$ to 1, in Case 2, we generate $|l(i)|$ from a uniform discrete distribution between 1 and t/4. The tools in set $l(i)$ are generated randomly. We generate r_k from a discrete uniform distribution in the interval [1, m–1] and s_j from a discrete uniform distribution in the interval [t/4, t/2]. The processing time of operation i, P_i, and the capacity of machine j, C_j, are generated in minutes and by a similar method used in Toktay and Uzsoy (1998). C_js are drawn from discrete random variables in the interval (0,720) and P_is are set to $\lfloor 720\times U(0,1) \times 0.8 \rfloor$. The weight of operation i, w_i, is discrete uniform between 25 and 150. For each combination of n, m, t and $|l(i)|$, we consider 10 problem instances. Hence we generate and solve a total of 480 (4×3×2×2×10) problem instances.

The models are generated in Visual Basic and solved by CPLEX 8.1 using a Pentium 4, 2.80 GHz computer with 520 MB RAM. We set an upper limit of 3600 seconds and we terminate the CPLEX run if the optimal cannot be found in this limit.

Table 3.1 reports the average and maximum CPU times (in seconds) for reaching the optimal solutions. Table 3.1 also provides the number of instances that can be solved to optimality. Although there are some exceptions, as the number of operations or the number of machines increase, both the CPU times and the number of instances that cannot be solved increase. We cannot see a significant effect of the number of tool types and number of tools in set *l*(*i*) on the CPU times since they do not directly affect the problem size.

Table 3.1 The CPU times (in seconds) of optimal solution and number of instances optimal is found in 3600 seconds

a) |l(i)|=1 case

		t=8			*t*=16		
n	*m*	Avg	Max	# Opt	Avg	Max	# Opt
25	3	0.16	1.34	10	0.08	0.39	10
	5	1.23	6.64	10	1.26	5.50	10
	7	10.49	100.16	10	12.24	106.34	10
50	3	0.60	3.88	10	0.46	2.73	10
	5	140.08	1359.61	10	361.17	1600.06	10
	7	96.38	354.06	6	85.42	338.69	4
75	3	3.18	12.63	10	5.94	55.67	10
	5	51.60	160.62	8	438.43	2485.47	7
	7	98.67	396.50	5	821.25	1569.22	2
100	3	2.36	21.48	10	45.14	319.55	10
	5	206.38	979.36	5	4.23	12.08	4
	7	-	-	0	281.61	1624.06	6

b) $|l(i)|^{3}1$ *case*

		t=8			t=16		
n	m	Avg	Max	# Opt	Avg	Max	# Opt
25	3	0.09	0.25	10	0.16	0.36	10
	5	0.97	5.92	10	5.04	23.41	10
	7	1.83	7.70	10	2.61	19.17	10
50	3	0.30	0.72	10	0.55	1.77	10
	5	5.66	37.66	10	34.15	101.16	10
	7	237.64	1413.55	8	259.13	948.86	10
75	3	0.23	0.50	10	2.18	7.66	10
	5	42.91	269.53	10	428.85	3200.78	8
	7	366.07	1234.41	5	916.79	2667.50	6
100	3	0.78	1.33	10	3.38	6.38	10
	5	90.23	294.44	10	326.84	875.08	7
	7	167.07	216.30	2	944.64	2837.81	6

We do not report any results for UB_5 as our preliminary tests indicate that UB_5 deviates from the optimal solution considerably. However, it is very quick and thus can be an attractive bound for implicit enumeration techniques.

Tables 3.2 and 3.3 show the average and maximum CPU times of each upper bound. In Table 3.2 we report on UB_2 and in Table 3.3 we report on UB_3 and UB_4. The solution times of UB_1 are negligible and therefore are not reported. Compared to UB_2, UB_3 and UB_4 need a significant time since they are outputs of integer programs. UB_4 is found more slowly than UB_3, due to the constraints added. Some instances for UB_3 and UB_4 could not be solved in our termination limit of 3600 seconds, as shown in the related tables. An increase in the CPU times as a result of an increase in the number of machines or operations, is observed for all upper bounds. Increase in the number of the tool types or the number of tools in set $l(i)$ increases the CPU times for UB_3 and UB_4 but does not affect UB_2.

Table 3.2 The CPU times (in seconds) of Upper Bound 2

		$\|l(i)\|=1$				$\|l(i)\| \geq 1$			
		t=8		t=16		t=8		t=16	
n	m	Avg	Max	Avg	Max	Avg	Max	Avg	Max
25	3	0.03	0.06	0.02	0.05	0.02	0.05	0.03	0.05
	5	0.04	0.05	0.04	0.06	0.05	0.06	0.06	0.09
	7	0.07	0.09	0.09	0.14	0.09	0.11	0.09	0.11
50	3	0.14	0.25	0.19	0.27	0.31	0.61	0.26	0.38
	5	0.40	0.55	0.42	0.53	0.49	0.61	0.56	0.72
	7	0.69	1.06	0.69	1.00	1.09	1.52	1.05	1.53
75	3	0.86	1.02	0.92	1.23	1.24	1.53	1.54	2.73
	5	1.61	2.19	1.70	2.27	3.28	4.98	2.62	3.75
	7	3.31	3.59	3.05	4.20	4.75	6.22	4.71	8.84
100	3	3.33	3.95	3.44	3.95	4.43	6.91	4.07	7.75
	5	5.31	8.00	4.98	6.17	10.87	23.89	9.33	17.25
	7	7.43	14.14	8.55	14.48	11.10	17.23	10.52	12.92

Table 3.3 The CPU times (in seconds) of Upper Bounds 3 and 4

a) Upper Bound 3

		$\lvert l(i)\rvert=1$				$\lvert l(i)\rvert\geq 1$			
		t=8		t=16		t=8		t=16	
n	m	Avg	Max	Avg	Max	Avg	Max	Avg	Max
25	3	0.02	0.11	0.01	0.02	0.04	0.09	0.17	0.36
	5	0.03	0.05	0.02	0.03	0.09	0.26	6.92	60.27
	7	0.09	0.42	0.03	0.08	0.38	1.08	0.96	3.94
50	3	0.03	0.05	0.02	0.03	0.07	0.13	0.25	0.41
	5	0.04	0.06	0.03	0.06	0.14	0.25	9.74	51.38
	7	0.05	0.09	0.04	0.08	0.31	0.56	56.97	204.89
75	3	0.06	0.17	0.01	0.11	0.09	0.17	0.66	1.70
	5	0.08	0.17	0.05	0.14	1.64	11.41	13.01	54.47
	7	0.06	0.19	0.03	0.06	1.00	3.61	191.85	931.41
100	3	0.05	0.13	0.03	0.08	0.20	0.28	1.02	2.30
	5	0.06	0.13	0.03	0.06	1.75	5.95	41.39	131.80
	7	0.09	0.22	0.06	0.09	2.44	8.02	58.82*	170.89*

* 8 instances are solved in 1 hr

b) Upper Bound 4

		$\lvert l(i)\rvert=1$				$\lvert l(i)\rvert\geq 1$			
		t=8		t=16		t=8		t=16	
n	m	Avg	Max	Avg	Max	Avg	Max	Avg	Max
25	3	0.09	0.23	0.07	0.19	0.12	0.31	0.33	0.86
	5	0.38	0.97	0.16	0.38	1.19	2.75	16.42	126.45
	7	1.75	6.66	24.51	237.25	5.74	26.77	199.54	1206.22
50	3	0.54	1.25	0.79	2.28	1.45	6.28	3.04	10.17
	5	1.22	2.05	0.82	1.91	3.54	7.36	260.27	2093.77
	7	3.47	10.39	1.91	3.55	48.39	161.17	902.60*	2592.86*
75	3	2.86	7.94	1.77	2.95	2.66	4.59	19.45	51.84
	5	3.65	7.11	2.66	3.42	112.36	599.39	504.03	2596.8
	7	11.95	41.19	6.44	13.23	204.33	990.56	712.91	1940.00
100	3	3.99	8.26	3.32	4.30	7.41	15.20	37.89	88.25
	5	7.14	12.28	8.36	15.69	184.95	381.3	1095.79**	2877.23**
	7	14.40	19.69	11.04	21.83	441.93**	1467.55**	961.81***	1692.45***

* 9 instances are solved in 1 hr

** 8 instances are solved in 1 hr

*** 4 instances are solved in 1 hr

Tables 3.4 and 3.5 show the average and maximum deviations from the optimal solution. In Table 3.4 we report on UB_1 and UB_2 and in Table 3.5 we report on UB_3 and UB_4. The deviations are given for the instances for which both the optimal solution and upper bound are available. When we analyze Tables 3.2–3.5, we see a trade-off between the solution time and solution quality. Obviously, UB_2 gives better solutions than UB_1 and UB_4 gives better solutions than UB_3. Also UB_3 and UB_4 perform better than UB_1. UB_4 performs significantly well, but at an expense of higher CPU times. The deviations from the optimal solution do not exceed 10% in almost all problems. Hence, it is a powerful estimator of the optimal solution. If we need a quick upper bound, UB_1 and UB_2 may be preferred. For all upper bounds, increasing $|l(i)|$ increases the deviations from the optimal solution. We could not observe a significant effect of the other problem parameters on the performances.

Table 3.4 The deviations of Upper Bounds 1 and 2 from the optimal solution

a) Upper Bound 1

		$\|l(i)\|=1$				$\|l(i)\|\geq 1$			
		t=8		t=16		t=8		t=16	
n	m	Avg	Max	Avg	Max	Avg	Max	Avg	Max
25	3	6.99	15.50	4.85	17.35	14.63	30.81	12.29	27.49
	5	4.71	9.38	5.09	12.57	5.87	14.17	8.57	19.29
	7	4.65	10.82	2.4	5.93	9.87	19.17	6.51	13.39
50	3	2.55	4.29	3.38	12.07	9.63	20.97	14.12	34.12
	5	2.20	8.77	1.43	2.59	3.42	8.19	8.63	21.29
	7	2.72	7.82	2.21	5.15	3.70	8.04	5.00	9.23
75	3	2.29	4.15	1.61	3.70	9.42	19.00	19.74	30.89
	5	1.25	3.38	1.20	2.71	3.40	7.45	9.07	21.88
	7	1.08	2.02	1.00	1.27	2.62	4.38	5.28	12.43
100	3	2.54	4.61	1.34	2.58	15.87	31.66	18.84	34.93
	5	0.90	1.69	0.52	0.92	6.92	19.22	9.89	16.47
	7	-	-	0.85	1.37	2.57	2.94	6.86	13.52

b) Upper Bound 2

		$\lvert l(i)\rvert=1$				$\lvert l(i)\rvert\geq1$			
		t=8		t=16		t=8		t=16	
n	m	Avg	Max	Avg	Max	Avg	Max	Avg	Max
25	3	4.99	10.51	2.79	8.50	11.05	26.23	9.55	19.02
	5	3.89	8.56	2.72	6.60	5.41	13.26	6.93	18.22
	7	3.52	7.14	2.02	4.35	7.68	18.03	4.63	11.42
50	3	2.35	3.96	2.70	7.24	7.83	20.22	12.61	30.34
	5	1.81	6.18	1.19	2.24	3.12	7.87	8.36	21.22
	7	1.65	3.76	1.18	2.13	3.23	6.57	4.90	9.04
75	3	2.06	3.58	1.51	3.02	7.40	12.24	19.45	30.32
	5	1.23	3.38	1.04	2.32	3.20	7.42	6.92	21.57
	7	0.92	1.52	1.00	1.27	2.54	4.38	5.20	12.21
100	3	2.07	4.61	1.31	2.58	13.50	28.75	17.84	32.66
	5	0.83	1.69	0.48	0.92	6.84	18.6	9.81	16.46
	7	-	-	0.69	1.31	2.41	2.94	6.86	13.52

Table 3.5 The deviations of Upper Bounds 3 and 4 from the optimal solution

a) Upper Bound 3

		$\lvert l(i)\rvert=1$				$\lvert l(i)\rvert\geq 1$			
		t=8		t=16		t=8		t=16	
n	m	Avg	Max	Avg	Max	Avg	Max	Avg	Max
25	3	6.77	14.93	4.83	17.13	10.44	25.99	7.33	21.87
	5	4.59	8.73	5.09	12.57	5.69	12.84	6.69	11.84
	7	4.10	9.01	2.40	5.93	7.59	16.27	5.82	10.54
50	3	2.53	4.29	3.19	12.07	7.74	18.42	5.83	12.00
	5	2.09	7.61	1.43	2.59	3.41	8.19	3.42	5.34
	7	2.53	7.82	2.21	5.15	3.59	8.04	3.40	5.02
75	3	2.22	3.78	1.61	3.69	6.19	18.89	5.28	8.01
	5	0.96	1.83	1.20	2.71	2.22	3.88	4.85	16.73
	7	1.08	2.02	1.00	1.27	2.43	3.58	3.65	5.93
100	3	2.45	4.61	1.34	2.58	6.17	17.37	4.07	7.30
	5	0.82	1.30	0.52	0.92	3.31	8.98	3.84	6.10
	7	-	-	0.85	1.37	2.02	2.68	3.04	4.88

b) Upper Bound 4

		$\|l(i)\|=1$				$\|l(i)\|\geq 1$			
		t=8		t=16		t=8		t=16	
n	m	Avg	Max	Avg	Max	Avg	Max	Avg	Max
25	3	3.16	5.69	2.55	7.11	5.81	18.92	3.08	5.74
	5	2.68	5.50	1.84	4.45	2.80	7.64	3.02	8.88
	7	2.29	4.29	1.82	3.28	3.65	6.91	1.56	3.08
50	3	2.22	3.89	2.10	3.63	5.07	10.78	4.27	8.57
	5	1.68	5.14	1.18	2.21	2.59	6.33	2.00	3.74
	7	1.45	2.63	1.18	2.13	2.58	5.03	2.77	3.93
75	3	1.96	3.40	1.36	2.25	2.75	3.97	3.72	5.12
	5	0.92	1.83	1.01	2.17	1.73	3.40	2.31	6.20
	7	0.92	1.52	1.00	1.27	1.87	2.25	2.93	5.39
100	3	1.92	4.36	1.31	2.58	4.62	12.69	3.41	6.34
	5	0.76	1.30	0.48	0.92	2.65	5.57	3.12	5.44
	7	-	-	0.69	1.31	1.83	2.61	2.40	4.03

CHAPTER 4

BRANCH AND BOUND ALGORITHM FOR THE TOTAL WEIGHT PROBLEM

In this chapter, we first discuss how the properties derived so far can be used to limit the number of partial solutions of an implicit enumeration technique. We then present the branch and bound algorithm, that employ the properties and the upper bounds defined before, to find optimal solutions.

4.1 Application of the Optimality Properties

We define a partial solution, as a set of operations whose decisions are already set: either assigned to a particular machine or decided to be unassigned. When an operation is added to the partial solution, it is either assigned to a particular machine or set unassigned. A decision representing the addition of operation i on machine j to the partial solution is discarded if the addition leads to an infeasible solution or the upper bound after the addition is no bigger than the best known lower bound. The addition leads to an infeasible solution if the tool magazine or time capacity of machine j is violated or the tools to be added for operation i are more than the number available.

We use a procedure to fathom the nodes and strengthen the upper bounds. We let M_i be the set of machines that can process operation i and S_j be the set of operations that can be processed by machine j when we consider the time and tool magazine capacities of the machines. Sets M_i and S_j are updated during the execution of the branch and bound algorithm. We say that the assignment of operation i to machine j will not lead to any tooling problem if either of the following conditions holds:

- The tools required by operation i are already loaded on machine j
- The tool magazine of machine j can accommodate all tools required by the operations in set S_j and the tools required by operation i, but not already on machine j, are sufficient even when loaded on all machines that do not already reside them.

In deciding the assignment of operation i, to a particular machine j, we use the following elimination mechanisms:

1. If $|M_i| = 0$ then ignore operation i.
2. Assume $|M_i| = 1$, and operation i is the highest weighted operation among the undecided ones. Assign operation i to the machine in M_i if the assignment will not lead to any tooling problem and w_i is greater than the total weight of the operations that are shorter than P_i and that are assigned to the machine in M_i.
3. If $|S_j| = 0$ then ignore machine j.
4. If $|S_j| = 1$ and the operation in S_j will not lead to any tooling problem then assign the operation in S_j to machine j, and ignore machine j.

We next check the optimality properties as follows:

i) Suppose there are two operations a and b such that $P_a \leq P_b$ and $w_a > w_b$ and $l(a) \subseteq l(b)$. If operation a is not processed, we only consider the partial solution where operation b is not processed (otherwise property 3.1 is violated).

ii) Suppose a subset of operations, G, such that $w_i > \sum_{d \in G} w_d$, $P_i \leq \sum_{d \in G} P_d$ and the operations in G are assigned to the same machine. If operation i can fit to the machine without any tool addition, then we discard the partial solution at which operation i is not assigned to any machine (otherwise property 3.2 is violated).

iii) Suppose a subset of operations, L, such that $\sum_{d \in L} w_d > w_i$, $\sum_{d \in L} P_d \leq P_i$ and $\bigcup_{d \in L} l(d) \subseteq l(i)$. If no operation in set L is processed then we discard the partial solutions corresponding to the assignment of operation i to any machine (otherwise property 3.3 is violated).

We also strengthen the upper bound with the following procedure: Suppose there are two operations, say a and b that cannot be selected for processing at the same time, i.e., either a or b can be processed but not both. Such a situation can occur if

- a and b cannot fit to the same machine because sum of their processing times exceed the time capacity of all machines, and
- a and b require the same tool type of which there is only one tool available, i.e., a and b cannot be assigned to different machines.

Let us have an upper bound solution where a and b are both selected. We generate a dummy operation d with $P_d = \text{Min}\{P_a, P_b\}$ and $w_d = \text{Max}\{w_a, w_b\}$. Then we resolve the upper bound problems by introducing the dummy operation and ignoring operations a and b, hence obtain a tighter bound.

4.2 Branch and Bound Algorithm

In this section, we define the branch and bound algorithm developed to find exact solutions to our NP-hard problem. In our branch and bound tree, each level represents an operation. At level r, the operation having the rth largest w_i/P_i value is scheduled. There are m+1 nodes at each level. The first m nodes represent the assignment of the corresponding operation to each of the m machines. The (m+1)st node is the decision of not assigning the operation to any machine, i.e., not processing it. The following figure illustrates our branching tree.

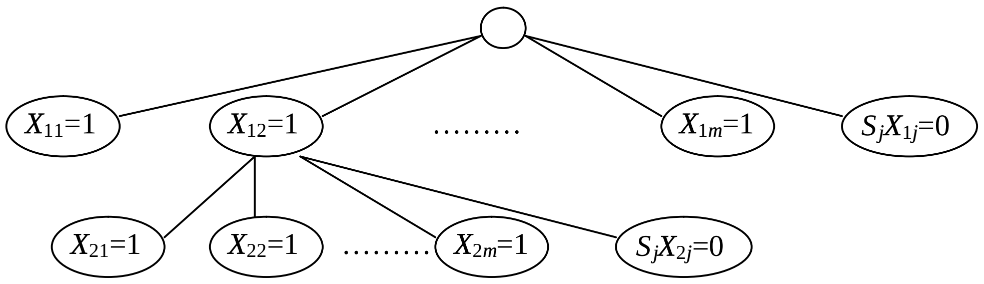

Figure 4.1 Branch and bound tree

For each node that cannot be fathomed by the reduction mechanisms, we compute UB_1 explained in Section 3.2.1. We eliminate the nodes that have upper bound values lower than the best known feasible solution. We use depth first search strategy, list the remaining nodes in their non increasing order of UB_1 values and select the first node of the list for further branching.

4.3 Computational Experiments

We design an experiment to test the performance of our branch and bound algorithm. We use the same set of data as described in Section 3.2.6, however we set the number of operations, *n*, to 25, 50, 75, 100, 125, 150, 200, 250, 300 and 500 and the number of machines, *m*, to 2, 3, 5 and 7.

4.3.1 Preliminary Experiments

We perform some preliminary runs to test the effects of the bounding mechanisms and dominance conditions on the performance of the branch and bound algorithm.

The results of our preliminary tests reveal that, the dominance conditions presented in Properties 3.2 and 3.3 decrease the number of nodes slightly, however increase the CPU times significantly. But the dominance condition presented in

Property 3.1 for a pair of operations leads to a reduction in CPU times as well. Hence, we only employ Property 3.1 in our application.

We start with an initial feasible solution of zero. We observe that a satisfactory lower bound is obtained very quickly due to our depth-first strategy. Hence we did not search for an initial lower bound.

We also compare the performances of the upper bounds, UB_1 and UB_5 defined in Sections 3.2.1 and 3.2.5, respectively. Our results have revealed that UB_1 outperforms UB_5, both in terms of number of nodes and CPU times. It is not even worth to use Min{UB_1, UB_5}, in place of UB_1. Table 4.1 reports the performance for alternative uses of upper bounds: UB_1, UB_5 and Min{UB_1, UB_5}.

Table 4.1 The average performance of branch and bound (*m*=3)

			UB_1		UB_5		Min{UB_1, UB_5}	
			# of nodes	CPU	# of nodes	CPU	# of nodes	CPU
n=25	|*l*(*i*)|=1	*t*=8	10,048	0.041	29,634	0.143	3,937	0.048
		t=16	8,874	0.047	26,305	0.142	3,617	0.042
	|*l*(*i*)|≥1	*t*=8	9,985	0.07	39,230	0.21	7,279	0.093
		t=16	2,169	0.023	12,823	0.134	1,381	0.023
n=50	|*l*(*i*)|=1	*t*=8	10,965	0.098	6,442,925	47.242	8,850	0.142
		t=16	30,673	0.257	5,444,243	43.781	20,819	0.323
	|*l*(*i*)|≥1	*t*=8	13,556	0.121	264,710	2.029	7,024	0.109
		t=16	22,491	0.281	1,019,496	8.458	16,342	0.342

As can be observed from the table, average number of nodes and average CPU times of UB_5 are much larger than that of UB_1 and these values significantly increase with the number of operations. When we consider Min{UB_1, UB_5}, the average number of nodes decrease, however the CPU times increase since we compute and evaluate two upper bounds at each node. As the problem size increases, the difference between the CPU times of UB_1 and Min{UB_1, UB_5} increases. Hence we conduct our experiments using UB_1.

4.3.2 Main Experiments

After deciding on the bounds and properties to be used, we carry out our main experiments. In Tables 4.2 through 4.5, we provide the average and maximum number of nodes and CPU times. We also give the number of instances that cannot be solved within 1 hour. Except some large size problems, almost all problems can be solved in 1 hour. As *m* and *n* increase, the number of problems that cannot be solved in 1 hour increases. For 7 machines case, instances with more than 25 operations could not be solved. For 5 machines, the upper bound on the number of operations is 75 for which optimal solutions are found in 1 hour. For 3 machines, this number increases to 300. When we have 2 machines, we can solve much larger size problems. Note that, if the problems cannot be solved in 1 hour, we assume its solution time is 3600 seconds and compute the average CPU times accordingly.

The effects of the number of operations, *n*, and the number of machines, *m*, on the problem difficulty can clearly be seen from the tables. As *m* increases, the solution time increases very rapidly due to the expansion of the branch and bound tree. The number of machines affects the breadth of the branch and bound tree, so adding even one machine increases the number of nodes too much. The effect of *n* is not as significant due to the fact that the feasibility issues control the size of the search. The number of operations that can be processed is limited; therefore increasing the number of operations increases the number of nodes and CPU times slower. When there are few machines, we can solve the problems with high number of operations.

The number of tools required to process an operation, $|l(i)|$, does not have a significant effect on the number of nodes or the CPU times. As we change $|l(i)|$, we cannot observe a consistent trend in the performance of the algorithm. For example, when n=125 and t=8, as we increase $|l(i)|$, the number of nodes and CPU time increase in 2 machines case, but decrease in 3 machines case.

An increase in the number of tool types, *t*, increases the number of nodes and the CPU times except a few problem instances.

Note that, there are significant differences between the average and maximum values in the tables. This shows the inconsistent behavior of the algorithm in different instances of the same parameter combination.

Table 4.2 Computational results for branch and bound algorithm with $|l(i)|=1$, $t=8$

			Number of nodes		*CPU times*	
n	*m*	# solved*	Avg.	Max.	Avg.	Max.
25	3	10	10,048	91,652	0.04	0.39
	5	10	326,542	1,190,571	1.85	6.42
	7	10	29,823,856	164,747,253	302.74	1442.70
50	3	10	10,965	54,126	0.10	0.44
	5	10	7,336,996	39,996,275	101.77	631.69
75	3	10	131,799	1,072,887	1.33	10.75
	5	6	14,199,893	53,056,872	1563.87	3600.00
100	2	10	3,978	12,266	0.05	0.13
	3	10	13,429	88,870	0.16	0.83
125	2	10	5,333	12,902	0.08	0.19
	3	10	2,072,062	10,246,772	29.57	138.31
150	2	10	8,970	58,888	0.19	1.28
	3	10	160,000	528,380	2.90	9.58
200	2	10	39,698	148,752	1.09	4.30
	3	10	278,055	977,733	8.66	36.33
250	2	10	12,644	57,086	0.30	1.31
	3	10	2,642,468	18,052,940	81.52	497.33
300	2	10	17,668	74,721	0.47	2.09
	3	10	628,099	2,693,256	31.53	152.17
500	2	10	1,718,727	10,626,526	93.12	511.81

* Number of instances solved in 1 hour (out of 10)

Table 4.3 Computational results for branch and bound algorithm with $|l(i)|=1$, $t=16$

			Number of nodes		*CPU times*	
n	*m*	# solved*	Avg.	Max.	Avg.	Max.
25	3	10	8,874	49,032	0.05	0.22
	5	10	17,815,255	172,585,021	167.68	1641.50
	7	9	59,893,576	187,612,195	798.94	3600.00
50	3	10	30,673	114,975	0.26	1.02
	5	8	40,865,867	148,618,769	1129.91	3600.00
75	3	10	82,384	495,443	0.78	4.41
	5	7	86,291,794	241,081,920	1866.01	3600.00
100	2	10	433	1,137	0.01	0.02
	3	10	1,105,918	7,279,157	11.70	77.89
125	2	10	40,999	318,027	0.43	3.27
	3	8	42,362,172	165,117,009	1154.78	3600.00
150	2	10	146,948	1,220,876	1.89	15.36
	3	9	7,947,691	46,607,806	553.51	3600.00
200	2	10	137,215	798,873	4.19	32.39
	3	9	31,719,779	99,794,967	1176.62	3600.00
250	2	10	159,080	909,261	4.78	32.17
	3	8	12,029,539	53,290,449	966.36	3600.00
300	2	10	231,184	1,234,191	11.50	87.88
	3	8	23,426,963	105,584,832	1230.46	3600.00
500	2	10	1,749,246	6,453,485	78.00	342.47

* Number of instances solved in 1 hour (out of 10)

Table 4.4 Computational results for branch and bound algorithm with $|l(i)|\geq 1$, t=8

			Number of nodes		*CPU times*	
n	*m*	# solved*	Avg.	Max.	Avg.	Max.
25	3	10	9,985	80,308	0.07	0.55
	5	10	289,070	2,292,326	2.01	15.95
	7	10	21,719,620	203,036,978	268.26	2549.00
50	3	10	13,556	73,584	0.12	0.56
	5	10	3,858,137	35,594,013	51.98	487.58
75	3	10	8,130	25,263	0.15	0.72
	5	10	11,065,399	94,240,415	222.63	1955.95
100	2	10	5,116	13,717	0.08	0.20
	3	10	39,331	155,122	0.72	3.16
125	2	10	11,375	49,912	0.19	0.95
	3	10	33,516	106,511	0.62	2.23
150	2	10	10,247	30,259	0.18	0.66
	3	10	508,398	2,435,791	13.65	62.44
200	2	10	9,032	35,364	0.24	1.23
	3	10	291,238	1,178,283	9.32	35.44
250	2	10	91,792	508,535	2.22	11.74
	3	10	805,906	4,230,691	34.73	179.16
300	2	10	123,800	430,491	5.21	21.17
	3	10	649,276	3,618,446	42.00	198.03
500	2	10	314,749	726,756	14.10	34.23

* Number of instances solved in 1 hour (out of 10)

Table 4.5 Computational results for branch and bound algorithm with $|l(i)|\geq 1$, t=16

			Number of nodes		*CPU times*	
n	*m*	# solved*	Avg.	Max.	Avg.	Max.
25	3	10	2,169	6,944	0.02	0.08
	5	10	4,654,009	23,806,351	50.56	258.22
	7	10	22,023,352	179,785,725	249.57	1999.08
50	3	10	22,491	109,776	0.28	1.44
	5	10	4,064,037	28,693,615	108.83	900.94
75	3	10	152,363	1,105,029	2.40	17.41
	5	8	31,909,419	116,365,477	1301.60	3600.00
100	2	10	24,789	71,023	0.43	1.39
	3	10	283,322	586,294	7.77	21.89
125	2	10	35,353	115,961	0.56	2.72
	3	10	331,050	2,840,359	8.18	72.17
150	2	10	50,175	180,143	1.28	6.63
	3	10	3,603,192	14,370,622	133.48	599.94
200	2	10	212,525	1,181,537	7.00	32.56
	3	10	9,648,356	59,354,182	520.74	3061.03
250	2	10	814,535	2,677,721	28.22	102.11
	3	10	15,914,289	36,038,503	1132.86	2546.56
300	2	10	2,629,268	12,956,325	122.55	495.98
	3	8	10,034,461	47,207,860	1201.97	3600.00
500	2	10	2,788,873	13,909,585	171.17	838.78

* Number of instances solved in 1 hour (out of 10)

In Table 4.6, the average and maximum nodes at which optimal solutions are found, are given. It can be seen that the optimal solution is found in the early stages of the algorithm. If we terminate the algorithm after a prespesified number of nodes evaluations or time limit, we may obtain high quality feasible solutions. For large size problems that cannot be solved in 1 hour, the incumbent, i.e., best known, solutions are likely to be close to the optimal ones.

Table 4.6 The average and maximum optimality nodes

a) |l(i)|=1

		t=8		*t*=16	
n	*m*	Avg.	Max.	Avg.	Max.
25	3	519	1,887	2,112	17,127
	5	108,460	953,934	955,966	9,502,628
	7	263,961	1,473,792	6,365,125	42,383,264
50	3	7,162	49,077	1,822	5,692
	5	997,628	2,607,835	209,603	914,543
75	3	24,838	121,192	6,505	46,307
	5	6,020,689	18,434,821	12,658,943	41,961,538
100	2	1,761	7,361	224	839
	3	2,763	6,721	20,842	89,028
125	2	3,167	8,722	3,966	14,258
	3	286,545	1,839,772	35,263	114,870
150	2	6,289	50,273	5,863	19,308
	3	103,413	476,402	1,142,682	7,147,426
200	2	29,029	98,078	11,646	37,265
	3	211,076	867,565	5,981,388	52,009,437
250	2	4,537	11,394	100,141	642,692
	3	407,649	2,416,761	855,364	6,441,127
300	2	13,531	73,835	33,844	141,668
	3	209,560	1,373,048	1,081,349	5,337,089
500	2	1,302,126	9,772,335	474,856	1,389,453

b) |l(i)|^3*1*

		t=8		*t*=16	
n	*m*	Avg.	Max.	Avg.	Max.
25	3	7,733	66,384	1,121	3,093
	5	82,132	673,706	710,010	6,413,441
	7	6,290,452	61,901,993	2,217,375	16,674,352
50	3	8,846	59,651	13,075	88,927
	5	2,530,388	23,375,869	1,649,261	9,139,517
75	3	4,739	21,374	88,101	675,466
	5	6,082,198	52,055,926	23,082,465	111,686,728
100	2	3,443	12,374	14,467	48,357
	3	25,800	91,149	177,262	433,568
125	2	5,482	22,398	21,538	76,241
	3	26,227	82,691	180,928	1,510,944
150	2	5,985	27,857	19,579	82,483
	3	324,741	1,440,901	973,185	5,252,899
200	2	4,369	15,785	126,310	682,392
	3	157,400	1,059,449	5,963,250	35,270,889
250	2	56,056	249,808	296,120	1,134,577
	3	680,229	4,154,648	7,729,755	21,696,563
300	2	65,293	160,968	448,651	2,708,093
	3	373,385	2,980,614	8,596,966	44,916,792
500	2	193,088	678,618	778,731	6,158,777

If we increase the time capacities of the machines, we can process more operations. We expect that, increasing the time capacities of the machines increases the number of nodes and hence the solution time of the branch and bound algorithm due to the increase in the number of feasible solutions. In order to see the effect of large machine time capacities, we multiply each C_j value by 10 for a small subset of problem instances and solve the problems in the set by our branch and bound algorithm. Table 4.7 shows the average and maximum CPU times and the number of nodes. As can be observed from the table, the CPU times and the number of nodes increase, as we expect. Moderate size problems can still be solved in reasonable times, however large size problems require more time.

Table 4.7 The computational results for the branch and bound algorithm with different machine time capacities (t=8, $|l(i)|$=1)

				CPU time		*Number of nodes*	
	n	*m*	# Opt*	Avg	Max	Avg	Max
Small Capacity	25	3	10	0.04	0.39	10,048	91,652
	100	3	10	0.16	0.83	13,429	88,870
	150	2	10	0.19	1.28	8,970	58,888
	500	2	10	93.12	511.81	1,718,727	10,626,526
Large Capacity	25	3	10	2.04	11.66	83,462	525,213
	100	3	8	1169.76	3600	11,511,106	35,290,372
	150	2	7	1148.12	3600	863,675	1,515,150
	500	2	3	2789.58	3600	1,758,501	3,642,145

* Number of problems that could be solved in one hour (out of 10)

In order to see how our reduction mechanisms work in eliminating the nodes, we kept the number of the nodes fathomed due to several properties. Throughout the branch and bound algorithm, we check the feasibility conditions and optimality properties in the following order and fathom the nodes if

1. the upper bound of a node is lower than the best known feasible solution,
2. the processing time of the operation exceeds the remaining time capacity of the machine,
3. the total number of additional tools required by the operations exceeds the remaining number of tools slots on the tool magazine of the machine,
4. the additional tools required by the operation are not available,
5. the operation is dominated by an unassigned operation,

Table 4.8 gives the average percentage of the nodes that are fathomed due to the properties given above among the eliminated nodes for a small subset of problem instances. We give the results for the original data used and for three cases where we multiply

- the time capacities of the machines by 10,
- the tool magazine capacities of the machines by 2,

- the numbers of tools by 2

other parameters remaining same.

As can be seen from Table 4.8, the nodes are generally fathomed due to the time capacity of the machines and upper bounds for the original data. However, the tool magazine capacities of the machines and the tool numbers have a significant effect in elimination of the nodes. Optimality Property 1 has a smaller effect compared to others. The sets M_i and S_j do not explicitly fathom the nodes; however they implicitly improve the performances of the upper bounds. When we increase the time capacities of the machines, the percentage of the nodes fathomed due to time capacity limit decreases, as expected. In this case, the tool magazine capacity and tool number constraints become more effective. Also when we increase the tool magazine capacities and the available tool numbers, the percentage of nodes fathomed due to these limits decreases. Note that, the reduction mechanisms are employed according to the order given above and a node is eliminated whenever a condition is satisfied and the remaining conditions are not checked.

Table 4.8 The average percentage of the nodes fathomed due to different properties (t=8, $|l(i)|$=1)

	n	*m*	*Upper Bound*	*Time Capacity*	*Tool Magazine Capacity*	*Tool Number*	*Optimality Property 1*
Original Data	25	3	46.68	40.42	5.67	6.90	0.33
	100	3	30.44	45.62	12.06	11.81	0.07
	150	2	31.93	29.08	27.41	10.62	0.98
	500	2	33.16	15.66	35.03	15.03	1.12
Large C_j	25	3	47.42	17.56	12.47	22.11	0.44
	100	3	33.44	28.41	8.92	28.99	0.24
	150	2	34.33	11.99	33.81	18.95	0.92
	500	2	30.18	17.48	36.79	13.23	2.32
Large s_j	25	3	50.69	41.58	0.21	7.18	0.35
	100	3	41.21	46.28	0.14	12.35	0.02
	150	2	46.94	37.83	2.31	12.90	0.02
	500	2	47.47	32.50	3.51	16.31	0.20
Large r_k	25	3	51.89	40.90	6.29	0.89	0.03
	100	3	39.79	43.66	15.02	1.51	0.03
	150	2	39.08	28.91	31.11	0.00	0.89
	500	2	32.58	15.92	50.44	0.00	1.06

The number of operations assigned in the optimal solution may give an idea about the difficulty of the problem. In Table 4.9, we give the average and maximum number of operations assigned to machines for a subset of problem instances. The number of operations assigned increases as the number of machines and the number of operations increase. Increasing the number of machines increases the total available capacity and gives more room for the operations to be processed. Increasing the number of operations increases the variability among the operations and we can select more operations to process among them. Also we can clearly see that increasing the time capacity of the machines increases the number of operations assigned, as expected. The number of tool types and the number of tools required to process an operation do not significantly affect the number of operations assigned.

Table 4.9 The number of operations assigned in the optimal solution (t=8, $|l(i)|$=1)

	n	m	$\|l(i)\|$	t	*Average*	*Maximum*
Small Capacity	25	3	1	8	7.9	13
				16	9.1	12
			≥1	8	8.6	13
				16	8.2	11
		5	1	8	10.9	13
				16	10.8	18
			≥1	8	11.1	15
				16	10.6	15
		7	1	8	13.0	17
				16	13.9	15
			≥1	8	12.9	15
				16	12.2	15
	100	3	1	8	14.3	19
	150	2	1	8	16.3	21
	500	2	1	8	27.5	37
Large Capacity	25	3	1	8	23.9	25
	100	3	1	8	43.6	54
	150	2	1	8	47.7	62
	500	2	1	8	65.7	108

If the variance of the operation weights is smaller, it may be more difficult to differentiate between the operations and this may increase the number of nodes and the CPU times of our branch and bound algorithm. To see the effect of the weights with smaller variance, we generated the weights of the operations from a discrete uniform distribution between [25, 50] for a subset of problem instances and solved the problems using our branch and bound algorithm. Table 4.10 gives the average and maximum number of nodes and CPU times.

Table 4.10 The computational results for the branch and bound algorithm with different weights (t=8, $|l(i)|$=1)

			CPU time		*Number of nodes*	
	n	*m*	Avg	Max	Avg	Max
Weights between [25,50]	25	3	0.06	0.22	11,091	47,079
	100	3	0.62	1.69	45,250	124,585
	150	2	0.42	2.25	19,075	76,238
	500	2	23.03	113.58	388,031	2,063,768
Weights between [25,150]	25	3	0.04	0.39	10,048	91,652
	100	3	0.16	0.83	13,429	88,870
	150	2	0.19	1.28	8,970	58,888
	500	2	93.12	511.81	1,718,727	10,626,526

Table 4.10 shows that, for n=25, 100 and 150, the CPU times and the number of nodes increase slightly when we generate the weights of the operations between [25, 50]. However, an opposite effect is observed for n=500. We may conclude that the variance of the weights does not significantly affect the performance of the branch and bound algorithm.

CHAPTER 5

APPROXIMATE SOLUTION APPROACHES FOR THE TOTAL WEIGHT PROBLEM

In this chapter, we present several procedures that find approximate solutions. These procedures are priority rule based heuristic, tabu search, best improving search and beam search algorithms. We also discuss the performances of these algorithms on test problems.

5.1 Priority Rule Based Heuristic

Priority rule based heuristic is a greedy algorithm that assigns the most promising operation at each step. As a quick and one pass approximation algorithm, it may not provide satisfactory solutions. However, it may be used to obtain initial solutions for improvement heuristics that we propose in the next two sections.

The following parameters are used to describe priority rule based heuristic.

AC_j = remaining time capacity of machine j

AT_j = remaining number of tool slots on the tool magazine of machine j

ar_k = available number of tools of type k

Below is the stepwise description of the heuristic.

Step 0. List the operations by Rule A and use Rule B as tie-breaker. Here, the maximum weight, minimum processing time, and maximum weight per unit processing time rules are employed as Rule A. When we use the maximum weight rule as Rule A, we use the minimum processing time rule as tie-breaker, i.e., Rule B. Otherwise we use the maximum weight rule for tie-breaking.

Set $AC_j = C_j$

$AT_j = s_j$

$ar_k = r_k$

Step 1. Select the first unassigned operation of the list that can be assigned to one of the machines. Let the operation be *i*.

If no such *i* exists, then stop.

Let a_{ji} be the set of tools to be added on the tool magazine of machine *j* for operation *i*.

Let *l* be the machine on which operation *i* requires the minimum number of tool additions, i.e., $a_{li} = \text{Min}_j \{a_{ji}\}$. Assign operation *i* to machine *l*.

Step 2. Update the parameters as:

$AC_l = AC_l - P_i$

$AT_l = AT_l - |a_{li}|$

$ar_k = ar_k - 1 \qquad \forall k \in a_{li}$

Go to Step 1.

5.2 Tabu Search Algorithm

In this section we first give a brief overview of the tabu search method and then discuss the specifics of our tabu search application.

5.2.1 An Overview of Tabu Search Algorithm

Tabu Search is a metaheuristic that guides the search procedure to explore the solution beyond local optimality. The method is introduced by Glover (1986) for solving the combinatorial optimization problems.

The tabu search algorithm starts at an initial feasible solution and in each iteration moves from one solution to another in the defined neighborhood. The move is usually realized to a solution having best, but not necessarily improving, objective function value. Some moves are forbidden and called *tabu* for preventing cycling and guiding the search towards the unexplored areas of the solution space.

A move, after made last time, is forbidden for a specified number of iterations, called its *tabu tenure*. If the tabu tenure is set to a high value, then a good non-tabu move can hardly be found. If it is set to a low value, then the same solutions may be obtained in the next few iterations. The tabu status of a move can be overridden in case it meets a specified condition, called *aspiration criterion*.

The method may employ two strategies, namely *intensification* and *diversification*, to define the move based on the history of the search. Intensification strategies give rise to the selection in the neighborhood of the current solution. Diversification strategies aim to drive the search into new neighborhoods by forbidding the moves that are very frequently changed during the search process.

After performing a number of moves in the neighborhood formed by changing the current solution via intensification strategy, the method jumps to a solution in the different neighborhood via diversification strategy. The method may terminate after applying the diversification strategy for a specified number of times. For each application, a specified number of intensification moves are made.

We refer the reader to Glover and Laguna (1997) for more detailed description and different versions of the tabu search algorithms.

5.2.2 Our Tabu Search Application

Our tabu search algorithm uses the heuristic procedure described in Section 5.1 to find an initial feasible solution.

Neighborhood Structure

The neighborhood of a solution is generated by exchanging a scheduled operation with an unscheduled one. We only accept the feasible solutions in the neighborhood, as maintaining an infeasible solution would be rather difficult due to the complex tooling relations.

Intensification Phase

In the intensification phase of the algorithm, i.e., the inner loop, we search the neighborhood of the current solution and move to a new non-tabu point that leads to the best improvement. This lets us exploit the search space near the current solution.

Tabu Attributes: An operation is made tabu if it is recently assigned to a machine or removed from a machine.

Tabu Tenures: Different tabu tenures are given to recently assigned and removed operations. If we set the tabu tenure of recently assigned operations too high, after a number of iterations we may not find any operation to remove. We give high tabu tenure to recently removed operations in order to give chance to other unassigned operations.

Tabu Status: A move is said to be tabu if at least one of the operations to be exchanged is tabu.

Aspiration Criterion: We remove the tabu status of a move if its total weight value is greater than the best known total weight value. The aim here is to favor solutions with high total weight values.

Once a move is realized, the associated operations are made tabu and all tabu tenures are updated.

Termination: Inner loop is terminated when a preset number of iterations (t_{max1}) are reached.

Diversification Phase

In the diversification phase of the algorithm, i.e. the outer loop, we update the weights of the operations by making use of frequency based memory, and restart the algorithm. Therefore, exploration of the search space is provided.

Frequency based memory

Frequency based memory keeps the number of iterations where an operation is assigned to a machine. This is done by using an array f[i] as follows: after each iteration, we check whether an operation is assigned to a machine or not; if it is, we increase f[i] by one. This is done by the following operation

```
for $i$=1 to $n$ do
    if Y($i$)>0
        Then f[$i$] = f[$i$]+1
    end
```

where Y(i) is the machine index where operation i is assigned to. Accordingly, Y(i) = 0 if operation i is not assigned to any machine. When we terminate the inner loop, we update the weights of the operations by subtracting the frequency of the operation from the weights, i.e. $w_i = w_i - f[i]$ for all i. Later, a new beginning solution is found and the inner loop is started again, with the new weights. The algorithm continues to run using the updated weights.

By the procedure above, the operations that have low weights in the new iterations are favored. We force the algorithm to start from a new point (jump to another region) and exploit a different part of the search space. The purpose here is not to be stuck at a local optimum. Outer loop is terminated when a prespesified number of iterations (t_{max2}) is reached.

5.3 Best Improving Search Algorithm

Best improving search algorithm uses the same neighborhood structure as the tabu search algorithm. At each iteration, the algorithm searches for feasible and improving exchanges among the assigned and unassigned operations. Then the exchange that results in the largest improvement in the objective function value is applied. Best improving search algorithm never moves to a non-improving solution and stops when a feasible or improving solution cannot be found. The number of iterations is limited with $t_{max1} \times t_{max2}$ in order to be comparable with the tabu search algorithm.

5.4 Beam Search Algorithm

Beam search algorithm is a heuristic branch and bound algorithm that returns a solution by searching only some promising partial solutions. At each level in the branch and bound tree, only a prespesified number of nodes are selected for branching, the rest of the tree is permanently discarded. Thus, the number of nodes evaluated is polynomial in the problem size.

At each level, the number of nodes kept for further branching, i.e., the number of promising nodes, is called ***beam width***, β. The evaluation function that determines the promising β nodes is called the ***beam evaluation function***.

Beam search technique sacrifices the guarantee of optimality for reduced solution times and memory requirements. The increase in β value and use of powerful elimination functions enhance the solution quality, however at an expense of higher solution times.

In the literature, many successful applications of the beam search technique are reported. Some recent noteworthy studies are due to Morton and Pentico (1993), Valente and Alves (2006), Della Croce *et al.* (2004) and Ghirardi and Potts (2005). Morton and Pentico (1993) give the details and several variations of the beam search technique. Valente and Alves (2006) apply different versions of the beam search algorithm to the single machine total weighted tardiness scheduling problem. Della Croce *et al.* (2004) apply a recovering beam search technique to some well-known combinatorial optimization problems. Ghirardi and Potts (2005) suggest recovering beam search approach for the makespan scheduling problem on unrelated parallel machines.

Next, we define the components of the beam search algorithm and explain the details of our implementation.

5.4.1 Beam Search Tree

Our beam search tree has the same structure with the branch and bound tree used in Section 4.2. Each level of the tree corresponds to an operation assignment related decision. At the first level, we consider the operation with the largest w_i/P_i value. At each level, there are m+1 nodes for the corresponding operation. The first m nodes represent the assignment of the operation to each of the m machines. The (m+1)st node represents the decision of not assigning the operation to any machine, i.e., not processing it. Accordingly, the first node sets the assignment variable X_{i1} to 1, the *m*th node sets X_{im}=1 and the (*m*+1)st node assumes $\sum_j X_{ij} = 0$. Figure 5.1 illustrates our beam search tree.

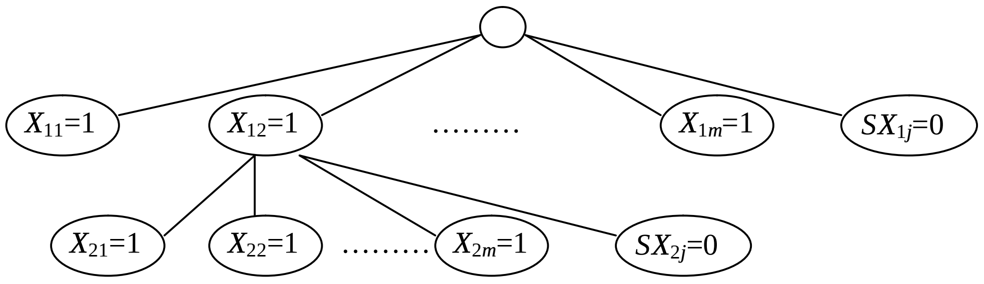

Figure 5.1 Beam search tree

5.4.2 Beam Evaluation Function

Beam evaluation function is used as a guide to determine the quality of the solution provided by a node. Our evaluation function considers the current operation assignments and an estimate for the future operation assignments. Such an estimate may be an upper bound or a lower bound or any weighted function of the upper and lower bounds. We define the weighted average evaluation function as $f = q$UB+(1-q)LB, where UB and LB are upper and lower bounds. Note that when q=0 and 1, the weighted average reduces to LB and UB, respectively.

We compute upper bounds 1 and 5 described in Sections 3.2.1 and 3.2.5 at each node and use Min{UB_1,UB_5} as UB in our beam search algorithm. We use the priority based heuristic presented in Section 5.1 as LB. We let Z_{inc} be the largest lower bound found so far. We fathom a node whenever its upper bound is no more than Z_{inc}. Moreover, we fathom a node whenever the upper bound solution is feasible.

We perform an initial experimentation to set appropriate values for q. We try q = 1, 0.75, 0.5, 0.25, 0 and found that q = 0, i.e., using the LB, as the beam evaluation function provides the best solutions.

5.4.3 Beam Width

We use two beam width assignment strategies, each of which is explained below.

5.4.3.1 Fixed Beam Width

In the classical beam search technique, the beam width is a prespesified parameter, it is initially set to b and never changed, hence it is fixed. As mentioned before, increasing the beam width improves the solution quality however increases the solution time. In practice, the most suitable b value is found empirically via preliminary experiments. We also performed a preliminary experiment to set the appropriate values for b. In our preliminary experiments, we tried $b = m$, $2m$, $3m$, $4m$, $5m$ and found that $4m$ provides the highest solution quality in reasonable solution times.

5.4.3.2 Variable Beam Width

A fixed beam width b may not fit well to all levels of the search tree. If there are so many promising nodes, one may prefer to keep more than b nodes for further consideration. On the other hand, if there are too few good solutions at some levels, one may want to eliminate more nodes. Recognizing these facts, the variable beam width strategy uses different b values at different levels of the search tree. The variable beam width strategy is applied by Valente and Alves (2006) to the total weighted tardiness problem. Their experimental results show the superiority of the variable beam width strategy over the fixed beam width strategy.

To determine the beam width to be used at each level, we compute the beam evaluation function for each node and find

$$F = aF_{\min}+(1-a)F_{\max}$$

where $F_{\min}$ and $F_{\max}$ are the minimum and maximum beam evaluation function values at that level, respectively. We let b' denote the number of nodes whose evaluation function values are no less than F.

By using a variable beam width, too many or too few nodes may be evaluated. Too many evaluations increase the solution times and too few evaluations decrease the solution quality. To avoid both cases, we set bounds on the variable beam width and let $b_{\min}$ and $b_{\max}$ be the lower and upper bounds on the beam width, respectively. We set $b=b_{\min}$ if b' is less than $b_{\min}$ and $b=b_{\max}$ if b' is more than $b_{\max}$.

In our preliminary experiments, we tried a = 0.25, 0.5, 0.75, 0.9, 0.95 and observed that a = 0.9 gives the highest quality solutions. We set $b_{\min}$ as m, we use 250 nodes for $b_{\max}$, as computer memory capacity did not allow more and 250 evaluations did not cause a significant increase in the solution times.

5.4.4 One Step Ahead Look Procedure

In the classical beam search technique, the child nodes of every beam are evaluated, the most promising b nodes are selected, and the remaining nodes are permanently discarded. However, the permanently discarded nodes might lead to better solutions than the selected nodes and such a case could be observed if some future levels were examined.

In this study, we propose a new procedure that evaluates one level ahead performances of the nodes before selecting them. At each level, we compute the beam evaluation function for grandchild nodes of each node. We sort the grandchild nodes in non-increasing order of the function values and select the parents of the most promising grandchild nodes. We continue until b different nodes are found.

We refer to this method as one step ahead look procedure. The one step ahead look procedure provides higher quality solutions as the future decisions are considered to some extent. However, such considerations are done at an expense of increased solution times.

We illustrate the above procedure through an example whose branch and bound tree is depicted in Figure 5.2.

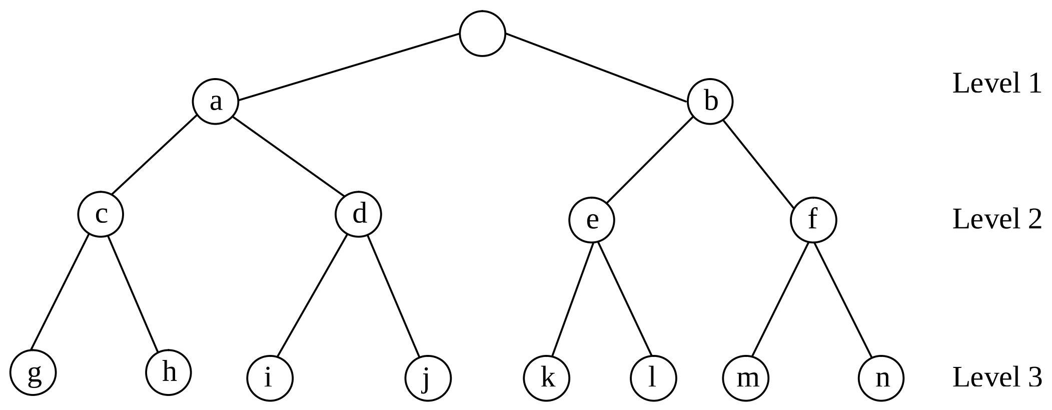

Figure 5.2 An example for one step ahead look procedure

Suppose that the beam width is 2 and nodes *a* and *b* are the beam nodes at level 1 and we are about to decide the beam nodes at level 2. In the classical beam search algorithm, we compute the beam evaluation functions of nodes *c*, *d*, *e* and *f*. Suppose the order of the function values is $F_c > F_d > F_e > F_f$. This results in the choice of nodes *c* and *d* as beams and the elimination of nodes *e* and *f*.

In one step ahead look procedure, we compute the beam evaluation functions of the nodes at level 3 to select the beam nodes of level 2. Suppose node *g* has the highest evaluation function value. So, its parent node, i.e., node *c*, is selected as beam node. Let node *h* have the second highest evaluation function value. Since its parent node is already selected, we check for the third highest evaluation function value. Suppose it belongs to node *k*. Then, the second beam node is its parent, which is node *e*. Note that, the one step ahead look procedure results with different beam selections than the classical procedure. The beam nodes at level 3 are found by evaluating the nodes at level 4 and so on.

5.5 Computational Experiments

We design an experiment to test and compare the performances of our approximate solution approaches. We use the same data set defined in Section 3.2.6 with the number of operations, n=25, 100, 300 and 500 and the number of machines, m=2, 3, 5 and 7.

5.5.1 Preliminary Experiments for Beam Search Algorithm

We first performed a group of experiments to determine the best value of b for the fixed beam width case. Table 5.1 gives the average deviations from the optimal solution for beam widths m, 2m, 3m, 4m, 5m when the beam evaluation function is 0.5LB+0.5UB. We calculate the deviation of an instance as

$$\%\text{ dev=}\frac{\text{Optimal Total Weight} - \text{Total Weight of Beam Search}}{\text{Optimal Total Weight}}\times 100\,.$$

The solutions for the beam width values 3m, 4m and 5m clearly outperform the solutions with m and 2m beam widths. Note from Table 5.1 that b = 4m generally gives the best solutions. Thus, we set the beam width to 4m in our main experiments. We observe similar performances when we change the weights in our beam evaluation function.

Table 5.1 The average performance of beam search algorithm for different *b* values, *f*=0.5(LB+UB)

a)|l(i)|=1

			b=*m*		*b*=2*m*		*b*=3*m*		*b*=4*m*		*b*=5*m*	
t	*n*	*m*	% Dev.	CPU Time	% Dev.	CPU Time	% Dev.	CPU Time	% Dev.	CPU Time	% Dev.	CPU Time
8	25	3	0.89	0.02	0.70	0.01	0.70	0.02	0.56	0.02	0.40	0.01
		5	3.31	0.01	2.49	0.00	2.39	0.01	1.68	0.01	1.68	0.01
		7	2.31	0.02	1.85	0.01	1.72	0.02	1.72	0.02	1.72	0.03
	100	2	1.33	0.02	1.34	0.02	0.78	0.00	0.82	0.01	0.92	0.03
		3	1.68	0.02	1.62	0.02	0.67	0.01	0.78	0.01	0.78	0.02
	300	2	1.12	0.03	0.91	0.03	0.86	0.04	0.83	0.05	0.52	0.03
		3	1.14	0.04	0.81	0.05	0.77	0.06	0.86	0.08	0.78	0.07
	500	2	2.61	0.04	1.28	0.03	1.09	0.04	1.07	0.05	1.16	0.06
16	25	3	1.33	0.01	0.86	0.00	1.21	0.02	1.40	0.02	1.64	0.01
		5	3.58	0.02	1.24	0.01	1.76	0.01	1.69	0.02	1.72	0.02
		7	3.37	0.02	4.02	0.02	2.97	0.03	2.97	0.03	2.97	0.05
	100	2	0.99	0.03	1.04	0.02	0.83	0.01	0.70	0.00	0.63	0.03
		3	1.07	0.02	0.75	0.02	0.75	0.01	0.41	0.02	0.45	0.02
	300	2	1.33	0.03	0.77	0.04	0.61	0.04	0.58	0.05	0.99	0.04
		3	1.52	0.06	0.53	0.05	0.49	0.07	0.62	0.09	0.97	0.08
	500	2	1.08	0.05	0.69	0.03	0.65	0.04	0.78	0.05	1.01	0.07

$b)|l(i)|^{3}1$

			b=*m*		*b*=2*m*		*b*=3*m*		*b*=4*m*		*b*=5*m*	
t	*n*	*m*	% Dev.	CPU Time	% Dev.	CPU Time	% Dev.	CPU Time	% Dev.	CPU Time	% Dev.	CPU Time
8	25	3	1.61	0.03	1.39	0.00	1.16	0.02	1.45	0.01	1.88	0.00
		5	2.91	0.02	2.46	0.01	2.09	0.02	1.33	0.02	1.53	0.02
		7	2.51	0.02	2.33	0.01	2.33	0.02	2.09	0.02	2.09	0.03
	100	2	0.98	0.03	0.64	0.02	0.44	0.01	0.44	0.00	0.52	0.02
		3	1.99	0.02	0.71	0.01	0.78	0.01	0.47	0.01	0.48	0.03
	300	2	2.97	0.03	2.19	0.03	1.93	0.03	1.29	0.05	2.44	0.04
		3	1.44	0.05	0.52	0.05	1.11	0.06	1.07	0.08	1.11	0.07
	500	2	4.90	0.03	3.27	0.03	1.78	0.04	2.58	0.05	3.09	0.07
16	25	3	2.60	0.02	2.01	0.01	2.01	0.02	2.01	0.01	1.88	0.01
		5	3.52	0.02	2.15	0.01	1.80	0.01	1.40	0.01	1.40	0.02
		7	3.27	0.02	2.81	0.01	3.68	0.02	3.23	0.02	3.55	0.04
	100	2	5.83	0.02	2.51	0.01	2.84	0.01	2.78	0.01	3.48	0.03
		3	2.64	0.03	1.43	0.02	1.30	0.02	1.38	0.04	1.17	0.03
	300	2	4.68	0.04	3.70	0.04	2.30	0.05	2.60	0.07	4.04	0.06
		3	6.17	0.06	4.81	0.07	4.78	0.09	3.40	0.13	3.67	0.13
	500	2	4.47	0.05	2.38	0.03	1.45	0.04	1.39	0.05	2.70	0.08

We then fix $b = 4m$ and carry out the experiments for determining the best beam evaluation function. We set $q = 1, 0.75, 0.5, 0.25, 0$ and give the average deviations from the optimal solution in Table 5.2. It can be observed from the table that the deviations are smallest when $q = 0$, i.e., using only lower bound as the beam evaluation function. This may show that lower bound used is a better approximation for the optimal solution than the upper bound. Even when only lower bounds are used for beam evaluation function, we calculate upper bounds in order to fathom the non-promising nodes and the nodes whose optimal solutions are already available.

Table 5.2 The average performance of beam search algorithm for different beam evaluation functions, b=4m

a)|l(i)|=1

			q=1		*q*=0.75		*q*=0.5		*q*=0.25		*q*=0	
t	*n*	*m*	% Dev.	CPU Time	% Dev.	CPU Time	% Dev.	CPU Time	% Dev.	CPU Time	% Dev.	CPU Time
8	25	3	1.40	0.01	0.56	0.02	0.56	0.02	0.56	0.01	0.37	0.02
		5	2.29	0.02	1.54	0.01	1.68	0.01	1.68	0.03	2.13	0.03
		7	3.20	0.02	1.74	0.03	1.72	0.02	1.67	0.03	2.25	0.03
	100	2	1.28	0.01	0.92	0.02	0.82	0.01	1.06	0.03	1.10	0.04
		3	1.85	0.02	0.96	0.01	0.78	0.01	0.78	0.02	0.23	0.04
	300	2	1.27	0.02	0.58	0.04	0.83	0.05	0.91	0.04	0.46	0.16
		3	1.49	0.04	0.80	0.08	0.86	0.08	0.92	0.08	0.67	0.26
	500	2	1.39	0.04	1.08	0.04	1.07	0.05	1.19	0.05	1.01	0.36
16	25	3	1.21	0.01	0.81	0.01	1.40	0.02	1.40	0.00	0.51	0.02
		5	1.85	0.02	1.69	0.02	1.69	0.02	1.26	0.03	1.22	0.02
		7	3.65	0.03	2.95	0.04	2.97	0.03	2.97	0.04	5.10	0.03
	100	2	0.74	0.01	0.74	0.02	0.70	0.00	0.70	0.02	0.02	0.04
		3	1.39	0.01	0.65	0.02	0.41	0.02	0.41	0.02	0.40	0.05
	300	2	1.91	0.02	0.58	0.04	0.58	0.05	0.60	0.04	0.67	0.17
		3	1.52	0.08	0.64	0.09	0.62	0.09	0.62	0.10	0.43	0.28
	500	2	1.90	0.06	0.75	0.06	0.78	0.05	0.80	0.05	0.60	0.38

$b)|l(i)|^{3}1$

			q=1		*q*=0.75		*q*=0.5		*q*=0.25		*q*=0	
t	*n*	*m*	% Dev.	CPU Time	% Dev.	CPU Time	% Dev.	CPU Time	% Dev.	CPU Time	% Dev.	CPU Time
8	25	3	1.07	0.01	1.63	0.01	1.45	0.01	1.45	0.00	0.89	0.03
		5	2.62	0.03	1.37	0.01	1.33	0.02	1.67	0.02	1.32	0.02
		7	2.65	0.02	2.13	0.03	2.09	0.02	2.32	0.03	1.98	0.03
	100	2	0.60	0.01	0.48	0.02	0.44	0.00	0.20	0.02	0.09	0.03
		3	2.05	0.01	0.96	0.03	0.47	0.01	0.47	0.02	0.80	0.05
	300	2	2.25	0.02	1.96	0.04	1.29	0.05	0.89	0.04	0.86	0.15
		3	1.87	0.03	1.27	0.07	1.07	0.08	0.93	0.07	0.95	0.26
	500	2	2.11	0.05	1.83	0.05	2.58	0.05	2.55	0.06	2.95	0.36
16	25	3	2.31	0.01	2.01	0.01	2.01	0.01	2.01	0.01	2.01	0.02
		5	3.87	0.02	1.71	0.02	1.40	0.01	1.62	0.03	1.42	0.03
		7	6.23	0.03	3.23	0.03	3.23	0.02	2.60	0.03	3.13	0.04
	100	2	4.52	0.01	3.85	0.02	2.78	0.01	2.72	0.02	2.01	0.04
		3	3.70	0.02	1.62	0.03	1.38	0.04	1.38	0.03	1.38	0.06
	300	2	3.10	0.03	2.97	0.06	2.60	0.07	2.69	0.06	2.91	0.17
		3	8.32	0.08	4.71	0.11	3.40	0.13	2.89	0.13	2.89	0.29
	500	2	3.34	0.04	1.63	0.05	1.39	0.05	2.04	0.06	2.09	0.38

We next investigate the effect of using one step ahead look procedure in reducing the average deviations and solution times. We compare the algorithms that use and that does not use one step ahead look procedure. We report the results for b=4m and f=LB. The results for the other combinations of b and f were similar, hence are not reported. As can be observed from Table 5.3, for all problem combinations, the one step look ahead procedure reduces the average deviations considerably, with a slight increase in the CPU times.

Table 5.3 The average performance of beam search algorithm with and without one step ahead look procedure

a)|l(i)|=1

			Without One Step Ahead Look		With One Step Ahead Look	
t	*n*	*m*	% Dev.	CPU Time	% Dev.	CPU Time
8	25	3	0.37	0.02	0.09	0.01
		5	2.13	0.03	1.50	0.03
		7	2.25	0.03	1.44	0.10
	100	2	1.10	0.04	0.47	0.03
		3	0.23	0.04	0.23	0.06
	300	2	0.46	0.16	0.26	0.19
		3	0.67	0.26	0.60	0.45
	500	2	1.01	0.36	0.87	0.51
16	25	3	0.51	0.02	0.49	0.01
		5	1.22	0.02	1.10	0.04
		7	5.10	0.03	3.00	0.15
	100	2	0.02	0.04	0.18	0.03
		3	0.40	0.05	0.45	0.08
	300	2	0.67	0.17	0.50	0.22
		3	0.43	0.28	0.32	0.50
	500	2	0.60	0.38	0.25	0.56

$b)|l(i)|^{3}1$

			Without One Step Ahead Look		With One Step Ahead Look	
t	*n*	*m*	% Dev.	CPU Time	% Dev.	CPU Time
8	25	3	0.89	0.03	0.57	0.01
		5	1.32	0.02	1.02	0.03
		7	1.98	0.03	1.60	0.09
	100	2	0.09	0.03	0.15	0.03
		3	0.80	0.05	0.47	0.06
	300	2	0.86	0.15	1.01	0.20
		3	0.95	0.26	0.70	0.41
	500	2	2.95	0.36	2.04	0.52
16	25	3	2.01	0.02	1.35	0.02
		5	1.42	0.03	0.59	0.05
		7	3.13	0.04	2.20	0.10
	100	2	2.01	0.04	1.71	0.04
		3	1.38	0.06	0.92	0.09
	300	2	2.91	0.17	2.25	0.21
		3	2.89	0.29	2.03	0.46
	500	2	2.09	0.38	1.88	0.55

To see the effect of using variable beam width, we perform an experiment for varying values of a. We set a to 0.25, 0.5, 0.75, 0.9 and 0.95, and report the average performances in Table 5.4. As can be clearly observed from the table, an increase in a value improves the performances at a small expense of increased CPU times. This is an expected result because when we assign a higher weight to $F_{\min}$, the number of nodes that have evaluation function larger than F increases and so the beam width. A larger beam width lets us keep more promising nodes and obtain better results. When we increase a from 0.9 to 0.95, we could not observe any significant decrease in average deviations. Hence we decide to set a = 0.9. The upper bound for the beam width is controlled by the memory limitations of the computer used. For 25, 50 and 100 operations, the limit is 250 nodes. For 300 and 500 operations, the limit decreases to 125 nodes. As those limits were of manageable size, we did not impose an additional upper bound value.

Table 5.4 The average performance of beam search algorithm for variable beam width for different *a* values with one step ahead look procedure

a)|l(i)|=1

			a=0.25		*a*=0.50		*a*=0.75		*a*=0.90		*a*=0.95	
t	*n*	*m*	% Dev.	CPU Time	% Dev.	CPU Time	% Dev.	CPU Time	% Dev.	CPU Time	% Dev.	CPU Time
8	25	3	0.00	0.03	0.00	0.03	0.00	0.13	0.00	0.22	0.00	0.23
		5	1.09	0.13	0.71	0.13	0.71	0.40	0.69	0.44	0.69	0.43
		7	0.78	0.42	0.71	0.42	0.71	0.64	0.71	0.69	0.71	0.68
	100	2	0.40	0.04	0.00	0.04	0.00	0.72	0.00	0.91	0.00	1.02
		3	0.23	0.05	0.00	0.05	0.00	0.60	0.00	0.90	0.00	0.84
	300	2	0.95	0.07	0.26	0.07	0.02	0.83	0.00	1.17	0.00	1.16
		3	0.68	0.16	0.22	0.16	0.22	1.58	0.22	2.09	0.22	1.87
	500	2	0.95	0.12	0.47	0.12	0.21	1.38	0.10	2.06	0.10	1.83
16	25	3	0.00	0.05	0.00	0.05	0.00	0.21	0.00	0.26	0.00	0.25
		5	0.59	0.30	0.51	0.30	0.34	0.49	0.34	0.55	0.34	0.53
		7	2.39	0.67	2.39	0.67	2.39	0.97	2.39	1.01	2.39	0.99
	100	2	0.14	0.05	0.00	0.14	0.00	0.31	0.00	0.39	0.00	0.43
		3	0.13	0.32	0.15	0.32	0.05	1.18	0.05	1.28	0.05	1.27
	300	2	0.81	0.38	0.22	0.38	0.09	1.62	0.09	2.04	0.09	1.94
		3	0.15	0.96	0.12	0.96	0.12	2.48	0.12	2.99	0.12	2.64
	500	2	0.89	0.28	0.15	0.28	0.04	2.15	0.04	2.91	0.04	2.58

b)|l(i)|³1

			a=0.25		a=0.50		a=0.75		a=0.90		a=0.95	
t	n	m	% Dev.	CPU Time	% Dev.	CPU Time	% Dev.	CPU Time	% Dev.	CPU Time	% Dev.	CPU Time
8	25	3	1.20	0.02	0.00	0.02	0.00	0.15	0.00	0.25	0.00	0.26
		5	0.76	0.08	0.00	0.08	0.00	0.35	0.00	0.42	0.00	0.41
		7	1.37	0.31	0.97	0.31	0.71	0.55	0.71	0.62	0.71	0.59
	100	2	0.66	0.02	0.00	0.02	0.00	0.51	0.00	0.77	0.00	0.91
		3	0.64	0.05	0.11	0.05	0.00	0.67	0.00	1.11	0.00	1.08
	300	2	2.87	0.05	1.69	0.05	0.89	0.85	0.15	1.35	0.09	1.35
		3	1.65	0.12	0.40	0.12	0.00	1.13	0.00	1.61	0.00	1.52
	500	2	2.76	0.13	1.50	0.13	0.19	0.98	0.11	1.81	0.11	1.81
16	25	3	1.93	0.02	0.46	0.02	0.00	0.17	0.00	0.26	0.00	0.27
		5	0.91	0.05	0.54	0.05	0.55	0.43	0.55	0.51	0.55	0.49
		7	2.32	0.26	0.29	0.26	0.29	0.62	0.29	0.77	0.29	0.68
	100	2	3.09	0.03	1.61	0.03	1.19	0.68	0.00	0.97	0.00	1.15
		3	1.76	0.04	0.64	0.04	0.33	1.15	0.08	1.37	0.08	1.58
	300	2	4.16	0.07	2.53	0.07	0.40	1.04	0.17	1.66	0.10	1.74
		3	4.22	0.11	1.21	0.11	0.58	1.41	0.51	2.11	0.45	2.06
	500	2	3.33	0.13	2.44	0.13	0.40	1.47	0.32	2.16	0.32	2.07

To summarize, the results of our preliminary experiment with several parameter combinations have clearly revealed that, the best results are obtained when the variable beam width with a = 0.9 is used, lower bound is used as an evaluation function and one step ahead look procedure is employed. The results associated with the best combination is highlighted in Tables 5.5 and 5.6, in Section 5.5.2. It can be observed from Tables 5.1 through 5.4 that almost all combinations are solved in less than one second.

In Section 4.3.2, we observe an increase in the solution times of branch and bound when the time capacities of the machines are increased. As beam search algorithm is based on the idea of branch and bound, one can expect an increase in the difficulty of beam search solutions with an increase in the time capacity. To verify this expectation, we multiply the machine time capacities by 10 for a small subset of problem instances. Table 5.5 gives the average and maximum deviations from the optimal solution and the CPU times. The deviations from the optimal solution are still very small. The CPU times increase slightly due to the increase in the number of operations assigned, hence the depth of the beam search tree. The table also gives the number of problem instances with known optimal solutions for each combination.

Table 5.5 The performance of beam search algorithm for large machine time capacities

					Deviation		CPU time	
$\|l(i)\|$	t	n	m	# Opt	Avg	Max	Avg	Max
1	8	25	3	10	0.21	1.73	0.94	1.22
		100	3	10	0.10	0.33	4.65	6.83
		500	2	10	0.47	1.42	12.37	18.91
	16	25	3	10	0.00	0.00	1.38	1.75
		100	3	5	0.20	0.78	8.45	13.42
		500	2	10	0.29	0.76	20.19	25.00
≥1	8	25	3	10	0.21	2.12	0.73	0.80
		100	3	10	0.44	2.52	5.94	10.02
		500	2	10	0.41	1.85	9.53	12.91
	16	25	3	10	0.15	1.47	0.81	1.00
		100	3	8	2.11	7.12	10.87	16.95
		500	2	10	0.82	2.49	14.26	31.61

If the weights of the operations are small, the total weight, i.e., objective function value, of the solutions will be small. This may affect the deviations of the solutions from the optimal objective function values. In order to see the effect of smaller weights, we generate the weights of the operations between [25, 50] for a subset of problem instances and apply beam search algorithm. Table 5.6 gives the average and maximum deviations from the optimal solution and the CPU times for these instances.

Table 5.6 The performance of beam search algorithm for small weights

				Deviation		CPU time	
$\|l(i)\|$	t	n	m	Avg	Max	Avg	Max
1	8	25	3	0.04	0.43	0.26	0.44
		100	3	0.07	0.67	1.05	1.63
		500	2	0.08	0.24	1.94	2.64
	16	25	3	0.06	0.33	0.19	0.30
		100	3	0.22	1.03	1.19	2.25
		500	2	0.08	0.62	2.56	4.05
≥1	8	25	3	0.00	0.00	0.17	0.31
		100	3	0.06	0.56	0.94	1.39
		500	2	0.07	0.68	1.66	2.44
	16	25	3	0.00	0.00	0.22	0.34
		100	3	0.00	0.00	0.84	1.11
		500	2	0.28	1.50	2.21	2.78

The results in Table 5.6 show that decreasing the weights does not affect the deviations and CPU times of the beam search algorithm and the small deviations are due to the good performance of the solution approach. The deviations are still very close to optimal and the algorithm runs in very small time.

5.5.2 The Comparison of Best Improving, Tabu Search and Beam Search Algorithms

In this section, we compare the performances of three approximate solution approaches proposed: best improving search, tabu search and beam search algorithms. To find a starting feasible solution for all algorithms, we compute the priority based heuristic solution defined in Section 5.1. We use the best solution among three priority rules as the initial solution.

In Tables 5.7 and 5.8, we report the performances of the instances for which the optimal solutions are known and are not known, respectively. Hence in Table 5.7, we calculate the deviations relative to the optimal solutions, and in Table 5.8, the deviations are relative to each other.

Table 5.7 gives the average and maximum deviations and the number of optimal solutions found among 10 problems for the three algorithms. We define the deviation of an instance as

$$\%\ \text{dev}=\frac{\text{Optimal Total Weight}-\text{Total Weight of Heuristic Solution}}{\text{Optimal Total Weight}}\times 100\,.$$

The beam search algorithm clearly shows a superior performance over the tabu search and best improving search algorithms. Beam search algorithm finds the optimal solution in 78% of the problem instances, while tabu search and best improving search algorithms find the optimal solution in 40% and 5% of the problem instances, respectively. The average deviation of the beam search solution from the optimal solution is less than 1% in all instances except one. Tabu search algorithm performs close to the beam search algorithm in small size problems especially in terms of the average deviation. However the deviations between their performances increase considerably with an increase in the problem size. The worst case performance of beam search algorithm is no more than 3% in majority of the problem combinations. On the other hand, the maximum deviations of the tabu search and best improving search algorithms are too high in majority of the problem instances. Note that, increasing the number of tools required by the operations increases the deviations of the tabu search and best improving search algorithms from the optimal solution. However, other parameters do not have a significant effect on the performances of the algorithms.

Table 5.7 The performances of the beam search, tabu search and best improving search algorithms

a)|l(i)|=1

			Beam Search			Tabu Search			Best Improving		
t	*n*	*m*	Avg	Max	# opt	Avg	Max	# opt	Avg	Max	# opt
8	25	3	0.00	0.00	10	2.28	10.54	5	7.37	13.40	2
		5	0.69	2.45	4	0.55	2.45	5	9.03	13.45	0
		7	0.71	2.69	6	0.69	1.93	5	10.10	22.14	0
	100	2	0.00	0.00	10	1.69	6.95	4	5.80	19.65	0
		3	0.00	0.00	10	1.37	9.57	6	10.39	22.73	0
	300	2	0.00	0.04	9	1.68	4.25	1	8.40	20.90	0
		3	0.22	0.52	5	2.51	4.73	1	7.93	14.66	0
	500	2	0.10	0.54	6	3.86	10.73	0	7.09	15.46	0
16	25	3	0.00	0.00	10	0.13	1.30	9	6.31	10.52	1
		5	0.34	1.79	7	0.20	1.79	8	7.46	20.72	1
		7	2.39	10.39	2	0.76	1.90	3	10.01	14.67	0
	100	2	0.00	0.00	10	0.25	1.50	8	3.19	9.91	3
		3	0.05	0.53	9	0.17	1.00	8	4.12	9.93	1
	300	2	0.09	0.42	6	1.42	3.45	1	8.66	22.03	0
		3*	0.12	0.66	5	0.71	2.25	3	6.33	10.22	0
	500	2	0.04	0.22	8	1.14	3.99	2	7.61	21.33	0

* Optimal solution of one problem could not be found in 1 hour.

b)$|l(i)|^3 1$

			Beam Search			Tabu Search			Best Improving		
t	*n*	*m*	Avg	Max	# opt	Avg	Max	# opt	Avg	Max	# opt
8	25	3	0.00	0.00	10	1.43	6.33	7	6.73	12.81	2
		5	0.00	0.00	10	0.28	2.27	8	11.09	17.46	0
		7	0.71	3.35	6	0.74	3.35	6	13.78	24.18	0
	100	2	0.00	0.00	10	6.14	28.06	4	8.90	28.06	2
		3	0.00	0.00	10	2.35	8.43	3	12.10	17.12	0
	300	2	0.15	0.89	8	8.10	25.96	2	14.35	26.15	0
		3	0.00	0.00	10	5.80	13.68	1	12.08	23.68	0
	500	2	0.11	1.08	9	10.40	15.79	0	14.97	24.26	0
16	25	3	0.00	0.00	10	3.15	10.47	3	12.78	25.55	1
		5	0.55	2.46	7	0.77	3.59	6	11.74	26.29	0
		7	0.29	1.56	7	0.33	2.16	8	13.54	22.78	0
	100	2	0.00	0.00	10	2.99	10.85	4	13.76	30.36	0
		3	0.08	0.57	8	3.43	10.97	2	14.47	30.22	0
	300	2	0.17	0.73	7	2.31	7.41	4	11.13	19.60	2
		3	0.51	2.33	6	4.63	9.42	0	17.11	28.50	0
	500	2	0.32	2.08	5	7.70	18.39	0	12.30	18.94	0

Table 5.8 gives the average and maximum deviations of the beam search and tabu search solutions from the solutions of the best improving search algorithm for the problems that we could not find the optimal solutions in 1 hour. We define the deviation of an instance as

$$\%\text{ dev}=\frac{\text{Solution value by Beam Search} - \text{Solution Value by Best Improving}}{\text{Solution Value by Best Improving}}\times 100$$

for beam search and as

$$\%\text{ dev}=\frac{\text{Solution value by Tabu Search} - \text{Solution Value by Best Improving}}{\text{Solution Value by Best Improving}}\times 100$$

for tabu search.

The table shows that the tabu search and beam search algorithms perform similar and much better than best improving search algorithm.

Among three algorithms, our beam search algorithm gives the best solution in 405 of the 440 problems, while the tabu search and best improving search algorithms give the best solutions in 175 and 15 instances, respectively.

Table 5.8 The deviations of the beam search and tabu search solutions from the best improving search solutions

				Tabu Search		Beam Search	
$\|l(i)\|$	t	n	m	Avg	Max	Avg	Max
1	8	100	5	6.45	12.79	6.84	12.74
			7	7.40	14.40	7.79	14.08
		500	3	7.86	30.65	10.27	31.45
	16	100	5	9.09	16.37	9.33	16.54
			7	7.31	15.59	7.70	16.93
		500	3	5.31	7.74	6.19	9.38
≥1	8	100	5	10.99	31.82	12.57	32.45
			7	7.02	13.00	8.70	16.31
		500	3	8.40	19.25	15.49	27.06
	16	100	5	14.47	20.57	16.74	25.51
			7	15.34	36.65	18.00	39.47
		500	3	10.71	37.42	16.77	38.51

Table 5.9 gives the CPU times of the algorithms. All algorithms find solutions in very small times even for large size problem instances. The average and maximum CPU times for the beam search and tabu search algorithms are close. The best improving search algorithm stops when no improving move is available, hence returns solutions relatively quicker.

Table 5.9 The CPU times of the beam search, tabu search and best improving search algorithms

a)|l(i)|=1

			Beam Search		Tabu Search		Best Improving	
t	*n*	*m*	Avg	Max	Avg	Max	Avg	Max
8	25	3	0.22	0.44	0.03	0.05	0.03	0.06
		5	0.44	0.74	0.03	0.03	0.04	0.08
		7	0.69	1.07	0.04	0.05	0.04	0.06
	100	2	0.91	1.22	0.32	0.53	0.03	0.62
		3	0.9	1.26	0.26	0.39	0.1	0.31
		5	2.72	3.95	0.32	0.36	0.25	0.34
		7	4.54	5.78	0.34	0.39	0.31	0.41
	300	2	1.17	1.74	1.7	2.48	0.03	0.06
		3	2.09	3.42	1.84	2.25	0.69	2.19
	500	2	2.06	2.88	4.64	6.44	0.41	3.83
		3	2.76	4.19	4.72	5.61	1.75	4.73
16	25	3	0.26	0.41	0.04	0.05	0.03	0.06
		5	0.55	1.66	0.04	0.06	0.04	0.08
		7	1.01	1.57	0.04	0.05	0.04	0.11
	100	2	0.39	0.84	0.2	0.28	0.12	0.23
		3	1.28	2.37	0.26	0.31	0.16	0.33
		5	3.46	5.75	0.32	0.41	0.31	0.67
		7	4.23	6.47	0.32	0.52	0.29	0.38
	300	2	2.04	5.87	1.72	2.97	0.29	1.38
		3*	2.99	5.04	1.72	1.98	1.18	2.44
	500	2	2.91	6.2	4.08	4.5	0.38	3.5
		3	4.28	7.58	4.33	5.08	2.1	4.5

* Optimal solution of one problem could not be found in 1 hour.

b)|l(i)|^3*1*

			Beam Search		Tabu Search		Best Improving	
t	*n*	*m*	Avg	Max	Avg	Max	Avg	Max
8	25	3	0.25	0.43	0.05	0.08	0.02	0.03
		5	0.42	0.72	0.05	0.06	0.05	0.09
		7	0.62	1.28	0.05	0.08	0.06	0.09
	100	2	0.77	1.18	0.48	0.72	0.08	0.55
		3	1.11	1.48	0.45	0.66	0.09	0.41
		5	2.08	2.83	0.54	0.73	0.39	0.83
		7	3.89	5.3	0.44	0.52	0.49	0.72
	300	2	1.35	1.76	2.79	4.52	0.03	0.05
		3	1.61	2.85	2.38	4	0.34	1.7
	500	2	1.81	2.18	5.7	6.97	0.6	5.66
		3	2.3	4.67	6.71	10.38	0.03	0.08
16	25	3	0.26	0.37	0.08	0.19	0.04	0.17
		5	0.51	1.14	0.07	0.17	0.09	0.17
		7	0.77	1.58	0.05	0.11	0.07	0.11
	100	2	0.97	1.61	0.59	0.95	0.07	0.44
		3	1.37	1.91	0.84	1.55	0.06	0.38
		5	2.79	4.08	0.84	1.13	0.9	1.98
		7	3.92	8.11	0.66	1.55	0.59	1.52
	300	2	1.66	2.38	3.72	5.78	0.31	2.92
		3	2.11	2.85	4.33	7.08	0.04	0.05
	500	2	2.16	3.21	5.45	6.95	0.03	0.05
		3	3.15	4.67	6.03	7.91	1.76	6.08

CHAPTER 6

BICRITERIA PROBLEM

In this chapter, we consider a bicriteria problem that trade-offs between total weight of the assigned operations and total usage and purchase cost of the assigned tools. We first define the problem and give its mathematical model. Next, we present some previous studies in the literature that are pertinent to minimizing machining or tooling costs. We then explain our Lagrangean relaxation based solution approach to find lower and upper bounds on the optimal objective function values. Finally, we discuss the results of the computational experiments designed to test the performance of our approach. Finally, we discuss the treatment of our bicriteria problem in different multicriteria contexts.

6.1 Problem Definition and Mathematical Model

Recall that, till now, we consider a single criterion problem so as to maximize the total weight. We assume that each tool is available at limited quantities. This tool availability constraint may represent a budget allocated for each particular tool type. However, in many practical instances, the tools may not be available and have to be purchased. In some cases, the tools may already be purchased and available at unlimited quantities, but their usages may incur costs. These usage costs may be due to wear downs, damages due to the frequent loading and unloadings.

Recognizing these practical issues, we model the environment so that the tooling costs are not constraining issues, but have to be penalized. Hence we consider the tooling cost as a component of the objective function representing the total tool purchase or usage cost. Once tooling cost appears in the objective function in some form along with total weight, then the associated problem is a bicriteria optimization problem. Our bicriteria problem trade-offs between the total weight of operation assignments and total cost of used tools.

Our problem with exactly one tool type requirement for each operation is analogous to the constrained compartmentalized problem surveyed in Section 2.3.2. A compartment in a knapsack resembles a tool type on a machine and the fixed cost of adding a compartment resembles the cost of using a tool type. Our aim is to maximize total weight minus the tooling costs, whereas the constrained compartmentalized problem aims to maximize the total utility minus the costs of using compartments.

In this case, we assume that each tool of type k can be purchased at a cost c_k. c_k represents the cost of using a tool if the tools are available at unlimited quantities, however their usages incur costs. The mathematical model becomes:

Maximize $$\sum_{i=1}^{n}\sum_{j=1}^{m} w_i X_{ij} - \sum_{k=1}^{t}\sum_{j=1}^{m} c_k Z_{kj} \qquad (6.1)$$

$$\sum_{j=1}^{m} X_{ij} \le 1 \qquad \forall i \qquad (6.2)$$

$$\sum_{i=1}^{n} P_i X_{ij} \le C_j \qquad \forall j \qquad (6.3)$$

$$\sum_{k=1}^{t} Z_{kj} \le s_j \qquad \forall j \qquad (6.4)$$

$$X_{ij} \le \frac{1}{|l(i)|} \sum_{k \in |l(i)|} Z_{kj} \qquad \forall i,j \qquad (6.5)$$

$$X_{ij}, Z_{kj} \in \{0,1\} \qquad \forall i,j,k \qquad (6.6)$$

Note that in (6.1), we maximize the net profit which is the difference between the profit brought over all assignments and the cost due to using or purchasing tools. Constraint set (6.2) ensures that an operation is assigned to at most one machine. Constraint sets (6.3) and (6.4) prevent exceeding the time and tool slot capacities of the machines, respectively. Constraint set (6.5) guarantees that an operation is assigned to a machine only if its required tools are loaded on the machine. Constraint set (6.6) defines the binary variables.

The problem is NP-hard in the strong sense, as it reduces to the multiple knapsack problem in the absence of the tooling requirements.

6.2 Literature Review

In the FMS literature, there are several problems that consider the tool and machine costs while assigning operations to the CNC machines.

Çatay *et al.* (2005) study the capacity allocation problem with machine duplications in semiconductor manufacturing. They model the problem of assigning individual operations to the predetermined machine groups where machine duplication is allowed as a variation of a generalized assignment problem. Their objective is to find the assignment that minimizes the total monthly operating cost of the machines and the monthly procurement cost of the additional machines capitalized. They use Lagrangean relaxation and Lagrangean decomposition techniques for obtaining lower bounds on the optimal solution and propose a heuristic procedure. Their experiments with different problem settings reveal the tightness of the lower and upper bounds.

Çatay *et al.* (2003) consider an allocation problem for wafer fabrication industry where they decide on the number of the tools. They assume that the demand for wafers over multiple time periods is known. They develop a mixed integer programming model that minimizes the machine tool operating costs, new tool acquisition costs and inventory holding costs. They propose a Lagrangean relaxation based heuristic algorithm to find efficient tool procurement plans.

Another study on the machine loading and tool allocation problem considering costs in FMSs is due to Sarin and Chen (1987). They aim to assign all operations of the parts and the associated tools to the machines. Once these decisions are made, the tools stay on their assigned machines and the parts route through the machines where necessary tools and programs are already loaded. They model the problem so as to minimize the total machining costs including cutting tools and machine usages. They discuss the changes in decisions influenced by the changes in problem parameters. They impose a constraint on the lower utilization of each machine so as to minimize the difference in machine utilizations and see that machining costs increase. They also discuss the application of Lagrangean relaxation to their problem.

Kim *et al.* (2003) aim to minimize the tool purchasing cost under deadlines on the completion times of the operations. They try to determine the number of tool copies for each tool type in a Flexible Manufacturing System so that the parts are completed by the maximum available makespan. They propose several heuristic algorithms that improve an initial feasible solution iteratively.

6.3 Solution Approach

In this section, we propose a Lagrangean relaxation technique to obtain lower and upper bounds on the optimal objective function value. We first discuss a Lagrangean relaxation method and a subgradient optimization that we will use to solve the Lagrangean problem. Then, we describe the application of the Lagrangean relaxation procedure to our problem.

6.3.1 Lagrangean Relaxation and Subgradient Optimization

Suppose we have the following integer programming problem:

$Z = \text{Max } cx$

$Ax \leq b$

$Dx \leq e$

$x \geq 0$ and integer

Assume that when the constraint set $Ax \leq b$ is removed, the problem will be an easy-to-solve one relative to the original problem. Therefore, we place this constraint set into the objective function with the vector of Lagrangean multipliers $u=(u_1,u_2,\dots,u_n)$ and solve the following problem.

$Z_D(u)=\text{Max } cx+u(b-Ax)$

$Dx\leq e$

$x\geq 0$ and integer

Note that $u(b-Ax)\geq 0$, provided that $u\geq 0$. Hence $Z_D(u)\geq cx^*+u(b-Ax^*)$ and $Z_D(u)$ is an upper bound for any positive u vector.

One of the important issues that should be addressed is the determination of the vector u. The value of u highly affects the efficiency of the solution. In most cases, finding u that makes Lagrangean solution close to the original solution is very hard. Additionally, the following issues need to be considered:

1. Selection of the constraint set to be relaxed so that the resulting problem will be an easy-to-solve one, relative to the original problem.
2. Selection of a solution approach for the Lagrangean problem.

We now discuss the above issues in relation to our problem.

1. *Selection of the relaxed constraint set.*

As we discussed, the constraint set to be relaxed should be the one that complicates the problem. So one should expect that the relaxed constraint set leaves a polynomially solvable problem or gives a decomposable structure to the problem. In the latter case, the decomposed problems could be solved independently and possibly relatively easier. In our case, once the constraint set that links the operation assignments to the tool assignments is removed, we have two independent decomposed problems: one for operation assignments, one for tooling assignments.

2. *Selection of a solution approach for the Lagrangean problem.*

There are several methods for solving the Lagrangean dual, and the following three methods are the most popular ones:

1. Subgradient optimization
2. Column generation
3. Multiplier adjustment

In this study, we use subgradient optimization method to update Lagrangean multipliers and solve the Lagrangean dual problem. We prefer subgradient optimization as its superior performance for many problems are reported in the literature. The subgradient method starts with initial Lagrangean multiplier u^0 and then at each iteration the sequence of Lagrangean multipliers $\{u^k\}$ is generated by the rule:

$$u^{k+1} = \text{Max}\{0, u^k - t_k(b - Ax^k)\}$$

where

x^k = an optimal solution to $LR_u{}^k$, i.e., the Lagrangean problem with dual variables set to u^k

t_k = a positive scalar step size

Held, Wolfe and Crowder (1974) developed the following result about the convergence of the subgradient method:

If $k \to \infty$, $t_k \to 0$ and $\sum_{i=1}^{k} t_i \to \infty$ then $Z_D(u^k)$ converges to its optimal value Z_D. A formula for t_k that has been proven to be effective in practice is:

$$t_k = \frac{l_k(Z_D(u^k) - Z^*)}{\sum_{i=1}^{m}(b_i - \sum_{j=1}^{n} a_{ij}x_j{}^k)^2}$$

where

Z*: The objective value of the best known feasible solution to the original problem. It is generally obtained through a heuristic.

$Z_D(u^k)$: The objective function value of the Lagrangean problem with multipliers set to u^k.

λ_k: A scalar between 0 and 2.

Frequently, λ_k is initially taken as 2 and reduced by a factor of 2 whenever $Z_D(u^k)$ fails to decrease in a specified number of iterations.

There are several stopping rules for the subgradient method (Beasley, 1995), some of which are listed below.

1. $Z_D(u^k) = Z^*$. In this case, the method returns the optimal solution.
2. The procedure iterates a specified number of times.

3. λ_k becomes too small.
4. $u^{k+1}-u^k \leq \varepsilon$, where ε is a prespecified small number.

Fisher (1981 and 1985) gives a thorough review of Lagrangean relaxation technique and discusses some application areas. In the literature, there are several successful applications of the technique, including, but not limited to, facility location, scheduling and generalized assignment problems.

We next discuss the application of Lagrangean relaxation technique to our NP-hard problem.

6.3.2 Lagrangean Relaxation of Our Problem

Consider the constraint set (6.5) below

$$X_{ij} \leq \frac{1}{|l(i)|} \sum_{k \in |l(i)|} Z_{kj} \qquad \forall i,j$$

Now, consider the relaxation of the constraint set (6.5) and obtain the following Lagrangean Relaxation (LR) model.

$$\text{(LR)} \quad \text{Maximize } \sum_{i=1}^{n}\sum_{j=1}^{m} w_i X_{ij} - \sum_{k=1}^{t}\sum_{j=1}^{m} c_k Z_{kj} + \sum_{i=1}^{n}\sum_{j=1}^{m} u_{ij}\left(\sum_{k \in l(i)} \frac{1}{|l(i)|} Z_{kj} - X_{ij}\right) \quad (6.7)$$

subject to (6.2), (6.3), (6.4) and (6.6)

$u_{ij} \geq 0 \quad \forall i,j$

where u_{ij} is the Lagrange multiplier associated to the constraint (i, j) in (6.5).

Constraint sets (6.2) and (6.3) are only related to X_{ij}s, and set (6.4) is only related to Z_{kj}s.

The objective function of LR can be rewritten as in (6.8).

$$\text{Maximize} \quad \sum_{i=1}^{n}\sum_{j=1}^{m}(w_i - u_{ij})X_{ij} + \sum_{j=1}^{m}\left(\sum_{i=1}^{n} u_{ij} \sum_{k \in l(i)} \frac{1}{|l(i)|} Z_{kj} - \sum_{k=1}^{t} c_k Z_{kj}\right) \qquad (6.8)$$

Note that the objective function in (6.8) has two separate terms, one related with X_{ij}s and the other related with Z_{kj}s. Therefore, LR can be decomposed into two independent subproblems, each of which is discussed below.

(SLR-1) Maximize $\sum_{i=1}^{n}\sum_{j=1}^{m}(w_i - u_{ij})X_{ij}$ (6.9)

subject to

$$\sum_{j=1}^{m} X_{ij} \le 1 \qquad \forall i \tag{6.10}$$

$$\sum_{i=1}^{n} P_i X_{ij} \le C_j \qquad \forall j \tag{6.11}$$

$$X_{ij} \in \{0,1\} \qquad \forall i, j \tag{6.12}$$

(SLR-2) Maximize $\sum_{j=1}^{m}\left(\sum_{i=1}^{n} u_{ij} \sum_{k\in l(i)} \frac{1}{|l(i)|} Z_{kj} - \sum_{k=1}^{t} c_k Z_{kj}\right)$ (6.13)

subject to

$$\sum_{k=1}^{t} Z_{kj} \le s_j \qquad \forall j \tag{6.14}$$

$$Z_{kj} \in \{0,1\} \qquad \forall k, j \tag{6.15}$$

SLR-1 is a generalized assignment problem, which is known to be strongly NP-hard. As it is very difficult to find the optimal solution to SLR-1 and we need to solve it at each iteration of the Lagrangean relaxation procedure, we prefer to use an upper bound for approximate solutions in place of optimal solutions. We employ the ideas used in deriving UB_1 and UB_5 defined in Sections 3.2.1 and 3.2.5, respectively, and obtain SUB_1 and SUB_2 below.

SUB_1: We remove the integrality constraints on X_{ij} variables. We list the operation-machine pairs in non-increasing order of $(w_i - u_{ij})/P_i$ values. Starting from the first pair in the list, we assign the operation to its machine pair. If the processing time of the operation exceeds the available time capacity of the machine, we split the operation. We continue until no further assignment is possible.

SUB_2: We let $[r]$ be the index of the operation having the r^{th} smallest processing time. Assume n_j is defined such that

$$\sum_{r=1}^{n_j} P_{[r]} \le C_j \quad \text{and} \quad \sum_{r=1}^{n_j+1} P_{[r]} > C_j$$

Hence n_j is a valid upper bound on the number of operations that can be processed on machine j.

We list the operation-machine pairs in non-increasing order of (w_i–u_{ij}) values. Starting from the first pair in the list, we assign the operation to the machine ignoring the time capacity of the machine. We decrease n_j by one. We continue until $n_j = 0$, for all j.

We take the minimum of SUB_1 and SUB_2 as an upper bound for the optimal solution value of subproblem 1, Z_{SLR1}.

SLR-2 is an easy problem whose optimal solution can be found through the following procedure: For machine j, we list the tools in non-increasing order of their objective function coefficients expressed in (6.13). We select the first s_j tools and set Z_{kj}=1 if the coefficient is positive. The associated objective function value gives the solution of subproblem 2, Z_{SLR2}.

We assume that every tool occupies exactly one tool slot. Future research may consider the case where some tools use more than one tool slot. Suppose tool type k requires d_k tool slots. Then the tool magazine capacity constraint becomes

$$\sum_{k=1}^{t} d_k Z_{kj} \le s_j \quad \forall j$$

The above constraint states that the tool slots required by the tools loaded on machine j cannot exceed the total slots available on the tool magazine of machine j, s_j.

In such a case, SLR-2 is a generalized assignment problem which can be handled like SLR-1.

We show that when u_{kj}s are given, the decomposed problems can be solved in polynomial time. Next, we discuss the way we generate the u_{kj} values.

The Lagrangean relaxation procedure applied to our problem is as follows:

Step 0. Set u_{ij}^0=0, Z*=0 or an initial feasible solution, λ_0=2.

Step 1. Solve SLR1 and SLR2, compute $Z_{\text{LR}}=Z_{\text{SLR1}}+Z_{\text{SLR2}}$

Step 2. Obtain a feasible solution by applying a Lagrangean heuristic.

Step 3. Update Z* if the heuristic in Step 2 produces a better solution.

Step 4. Calculate the step size t_p

$$t_p = \frac{l_p(Z_D(u^p) - Z^*)}{\sum_{i=1}^{n}\sum_{j=1}^{m}\left(\sum_{k\in l(i)}\frac{1}{|l(i)|}Z_{kj} - X_{ij}\right)^2}$$

Step 5. Update Lagrangean multipliers

$$u_{ij}^{p+1} = Max\left\{0, u_{ij}^{p} - t_p\left(\sum_{k\in l(i)}\frac{1}{|l(i)|}Z_{kj} - X_{ij}\right)\right\}$$

Return to Step 2.

If Lagrangean solution does not decrease in a specified number of iterations, halve the scalar l_p.

In initial setting and updating the values of t_p and l_p, we refer to the results that are proven to be effective in the literature.

After some preliminary experimentation, we decided to run the algorithm for 10,000 iterations and halve l_p when the Lagrangean solution does not improve in 500 iterations.

We now discuss our Lagrangean heuristic used in Step 2 of the above procedure. We develop a myopic heuristic procedure that selects the operation and machine pair having the maximum contribution to the objective function among the ones that can be assigned without exceeding tool magazine capacity. Below is the stepwise description of our Lagrangean heuristic.

Step 1. Compute the contribution of each operation-machine pair. The contribution is the weight of the operation minus the total cost of the additional tools that should be loaded on the machine to process the operation.

$$wh_{ij} = \begin{cases} w_i - \sum_{k|k\in l(i)\wedge Z_{kj}=0} c_k & \text{if operation } i \text{ is not assigned} \\ 0 & \text{if operation } i \text{ is already assigned to a machine} \end{cases}$$

Step 2. Find the operation-machine pair with the maximum contribution. Call the pair as operation *i′* and the machine *j′*. Check whether

- the processing time of operation i' is less than the time capacity of machine j', and
- the tool magazine of machine j' has enough number of tool slots for the required tools of operation i'.

If both conditions are satisfied, assign operation i' then the additional tools required to process operation i' to machine j', and update $Z_{kj'}$ for all $k \in l(i')$, $P_{i'}$, $C_{j'}$, and $s_{j'}$.

Return to Step 1. If no more assignment is possible, then stop.

6.3.3 Initial Solution

For finding an initial feasible solution to our problem, we modify a heuristic procedure proposed in Bilgin and Azizoğlu (2006) for a capacity allocation problem with operation splitting. We assign the operations and necessary tools based on a maximum weighted matching algorithm. The aim of a matching algorithm is to find an assignment of operations to machines with the maximum total weight. The network representation of our matching algorithm is given in Figure 6.1.

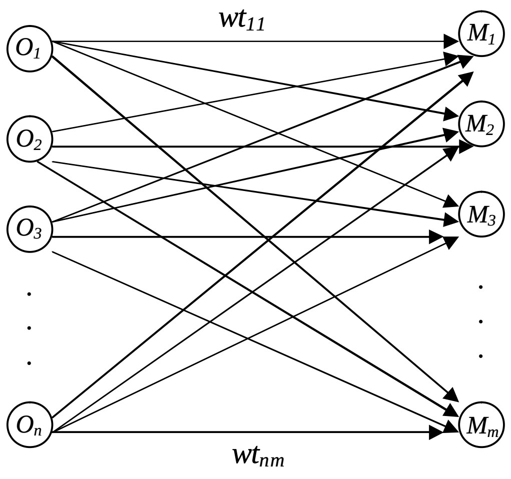

Figure 6.1 Network representation of the maximum weighted matching problem

In the above network, the operations are represented by the nodes O_1, O_2, …, O_n and the machines are represented by the nodes M_1, M_2, …, M_m. The weight of each arc from node *i* to node *j* is wt_{ij}, where wt_{ij} is the contribution of assigning operation *i* to machine *j*, into the objective function. We try to send flow from the operation nodes to the machine nodes simultaneously in each iteration.

The steps of the matching-based heuristic is as follows:

Step 1. Construct a bipartite graph containing operation and machine nodes. Calculate the weight of an arc from operation *i* to machine *j* as

$$wt_{ij} = \begin{cases} w_i - \sum_{k \mid k \in I(i) \wedge Z_{kj}=0} c_k & \text{if assigning operation } i \text{ to machine } j \text{ is feasible} \\ 0 & \text{otherwise} \end{cases}$$

Assigning operation *i* to machine *j* is infeasible if any of the below cases occurs

- The processing time of operation *i* is greater than the available capacity of machine *j*.
- The number of additional tools required by operation *i* is greater than the available number of tool slots on machine *j*.

Step 2. In order to obtain a maximum weighted matching of operations and machines, solve the following linear program:

The assignment variable is defined as below:

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

The objective function expressed in (6.16) maximizes the total weight.

$$\text{Maximize} \quad \sum_{i=1}^{n} \sum_{j=1}^{m} wt_{ij} X_{ij} \tag{6.16}$$

The constraint set (6.17) ensures that at most one operation is assigned to each machine.

$$\sum_{i=1}^{n} X_{ij} \le 1 \qquad \forall j \tag{6.17}$$

The constraint set (6.18) ensures that at most one machine is allocated to each operation.

$$\sum_{j=1}^{m} X_{ij} \leq 1 \qquad \forall i \qquad (6.18)$$

The nonnegativity of the decision variables is stated in (6.19).

$$0 \leq X_{ij} \leq 1 \text{ and integer} \quad \forall i, j \qquad (6.19)$$

The model expressed by (6.16) through (6.19) is a weighted bipartite matching, i.e., assignment, model whose constraint matrix is totally unimodular. This implies the optimal solution to the Linear Programming relaxation of the model gives all integer variables (see Nemhauser and Wolsey, 1988) and therefore the constraint set (6.19) can be replaced by $X_{ij} \geq 0$.

Step 3. Assign all the required tools by the operations to the corresponding machines. Update time and tool slot capacities of the machines.

Step 4. Stop if no more assignment is possible. Otherwise, go to Step 1.

We apply an improvement procedure at the end of the matching based heuristic. We check whether we can improve the matching solution by exchanging an assigned operation with an unassigned one. At each step, we find such feasible exchanges and apply the one that provides the highest improvement. We continue until no more feasible or improving exchange is found.

6.3.4 Computational Experiments

We design an experiment in order to test the performance of our Lagrangean relaxation approach. We set the number of operations, n, to 25, 50, 100, 150, 200, 300 and 500 and number of machines, m, to 3, 5, and 7 in our experiments. We set the number of tool types, t, to 8 and 16.

We use a similar method for generating the test data as in Marquez and Arenales (2007). We assume that the time capacities of the machines, C_j, are equal to 1188 time units. We generate the processing times of the operations, P_i, from the discrete uniform distribution in the interval [20, 444]. We set the weights of the operations, $w_i = P_i + s_i$ where s_i is an integer number in the interval $[0, P_i]$. We generate s_j from a discrete uniform distribution in the interval $[t/4, t/2]$ as in the total weight problem. We generate the tool usage/purchase costs, c_k, from the discrete uniform distribution in the interval [1, 100]. As the c_k values are not too high, they can represent the tool usage costs. In this case, the tools are available in the system and the costs are charged once they are used. On the other hand, the higher c_k values may represent the tool purchase case.

We assume that each operation requires a single tool as in Marques and Arenales (2007). For modifying the solution procedures for $|l(i)|\geq 1$ case for the bicriteria problem, we may consider problem specifications in the Lagrangean problem and connect two subproblems in order to obtain tighter bounds.

We generate and solve 10 problem instances for each parameter combination.

We could find the optimal solutions of the bicriteria problem only for cases with 3 machines and up to 100 operations using CPLEX 8.1 software. Table 6.1 gives the average and maximum deviations of the lower and upper bounds for these problems. The results show that both lower and upper bounds are close to the optimal solution value. The deviations of the lower bounds are generally smaller than those of upper bounds. The number of operations and tool types do not significantly affect the solution quality.

Table 6.1 The deviations of the upper and lower bounds from the optimal solution (m=3)

		Lower Bound		Upper Bound	
t	n	Avg	Max	Avg	Max
8	25	0.58	1.32	2.32	3.55
	50	1.02	1.98	1.59	2.61
	100	0.89	1.55	1.23	1.74
16	25	1.05	2.54	1.65	2.72
	50	0.75	2.03	1.02	1.59
	100*	0.62	1.22	1.07	1.85

* The optimal solution of one problem could not be found

Table 6.2 shows the average and maximum percent deviations of the lower bound from the upper bound and the average and maximum CPU times (in seconds) of the algorithm. We define the deviations as the difference between upper and lower bounds as a ratio of the upper bound. It can be seen from the table that, except for n=25 case, the deviations only increase with the number of machines. For the fixed number of machines, the deviations are almost same in all problem instances for all parameter combinations. This shows the robustness of the solutions. The deviations between the lower and upper bounds are too small. Hence, we conclude that both bounds are very close to the optimal solutions even for large sized problem instances. The CPU times increase with an increase in the number of machines or the number of operations. The average and maximum deviations and CPU times are very close to each other for all problem combinations. This shows the consistent behavior of our approach over all problem instances.

Table 6.2 The percent deviations between the upper and lower bounds and CPU times

		t=8				*t*=16			
		Deviations		*CPU times*		*Deviations*		*CPU times*	
n	*m*	Avg	Max	Avg	Max	Avg	Max	Avg	Max
25	3	2.83	4.09	2.87	3.46	2.64	3.67	3.32	3.77
	5	3.91	7.66	5.79	6.09	2.99	6.39	6.41	6.60
	7	2.13	4.31	7.09	7.26	0.90	3.21	7.99	8.18
50	3	2.57	4.47	8.13	8.48	1.74	3.56	8.98	10.68
	5	4.90	6.90	20.62	21.79	3.79	4.40	22.49	25.17
	7	6.40	8.15	30.79	32.07	6.44	7.94	32.44	34.19
100	3	2.09	3.14	22.80	23.33	1.67	2.37	23.76	24.23
	5	4.25	5.48	65.45	72.99	3.79	4.87	71.90	80.25
	7	6.77	8.20	93.44	96.35	5.55	6.79	96.22	98.50
150	3	2.00	3.09	32.31	33.38	2.11	3.25	33.50	34.63
	5	3.82	4.75	102.11	105.18	3.95	5.08	104.93	109.34
	7	6.31	7.94	145.57	147.91	5.87	6.78	148.52	150.84
200	3	2.53	3.38	74.67	79.96	1.98	4.59	75.79	78.95
	5	3.79	5.06	227.72	230.01	3.81	5.10	231.00	232.82
	7	5.40	7.44	322.73	330.41	5.21	6.94	325.43	331.37
300	3	1.88	2.55	158.05	170.96	1.86	2.47	158.03	163.20
	5	3.80	5.28	493.64	502.84	3.31	4.36	495.80	499.87
	7	4.68	5.79	700.25	708.73	4.67	6.07	701.31	705.85
500	3	1.36	1.74	324.33	334.81	1.66	2.76	320.54	321.74
	5	3.29	4.39	1032.57	1065.88	3.16	3.91	1024.30	1048.33
	7	5.00	6.26	1481.48	1485.83	4.91	5.68	1496.83	1563.78

6.4 Multicriteria Evaluation Techniques

We treat the bicriteria problem as a single objective problem by combining the objectives to form a single function. It might be possible to employ different multicriteria evaluation techniques.

To bring two objectives into common optimization sense, maximizing total weight of operation assignments may be treated as minimizing total penalty of unassigned operations. Note that the tooling cost and the unassignment cost concerns are often in conflict, as reducing the tooling costs would leave more unassigned operations, thereby increasing the total unassignment cost. If we let f_1 denote the total unassignment cost and f_2 denote the total tooling cost, we can define a function g which is non-decreasing in f_1 and f_2.

The total unassignment cost f_1 can be expressed as

$$f_1 \equiv \sum_i u_i (1 - \sum_j Xij)$$

where u_i is the penalty of not assigning operation i to any machine. This penalty may be proportional to the weight of the operation.

For the sake of clarity of presentation, we define a binary variable Y_i such that

$$Y_i = \begin{cases} 1 & \text{if job } i \text{ is not assigned to any machine} \\ 0 & \text{otherwise} \end{cases}$$

Accordingly,

$$Y_i = 1 - \sum_{j=1}^{m} X_{ij} \qquad \forall i \qquad (6.20)$$

and $f_1 \equiv \sum_i u_i Y_i$.

The total tooling cost f_2 can be represented as

$$f_2 \equiv \sum_k \sum_j c_k Z_{kj}$$

where c_k is the cost of using or purchasing tool k.

We may consider the trade-offs between f_1 and f_2 in the form of hierarchical, constrained or simultaneous optimization. In the next subsections, we discuss each form.

6.4.1 Hierarchical Optimization Problem

We define two types of priorities as primary and secondary. Once we set our primary objective to f_1, the problem becomes

$$\begin{aligned}&Min\ f_2\\&s.t.\quad f_1 = f_1^*\\&\qquad (6.2)-(6.6),(6.20)\end{aligned}$$

where f_1* is the optimal solution to the single criterion problem of minimizing f_1. Note that among the solutions that minimize the total unassignment cost, we are looking for the one having minimum total tooling cost. Hence, the tooling cost is a tie-breaking objective.

Alternately, we can define the following objective and remove constraint f_1=f_1*.

$$\begin{aligned}&Min\ f_1 + e_2 f_2\\&s.t.\quad (6.2)-(6.6),(6.20)\end{aligned}$$

where $0<e_2<<1$.

The second hierarchical problem is to primarily minimize the total tooling cost and secondarily minimize the total unassignment cost. The corresponding objective is

$$\begin{aligned}&Min\ f_2 + e_1 f_1\\&s.t.\quad (6.2)-(6.6),(6.20)\end{aligned}$$

where $0<e_1<<1$.

6.4.2 Constrained Optimization Problem

We define two constrained optimization problems as

(1) $Min\quad f_1$

$s.t.\quad f_2 \le k_2$

(6.2)–(6.6), (6.20)

(2) $Min\quad f_2$

$s.t.\quad f_1 \le k_1$

(6.2)–(6.6), (6.20)

The first problem puts a limit on the total tooling cost, hence may well represent a budget constraint. The second problem puts an upper limit on the total unassignment cost, which can be interpreted as a lower limit on the total profit, i.e., the total weight of the assigned operations.

For both problems we can set tie-breakers and define the following problem

(1′) $Min \quad f_1 + e_2 f_2$

$s.t. \quad f_2 \le k_2$

(6.2)–(6.6), (6.20)

where e_2 is a sufficiently small number.

In such a case, the solution having smallest f_2 value, among the ones that minimize f_1 while satisfying $f_2 \le k_2$, is selected. Note that optimal f_1 values are the same in (1) and (1′) and f_2 value of (1′) is never greater than that of (1).

Using the similar reasoning we can define the following problem

(2′) $Min \quad f_2 + e_1 f_1$

$s.t. \quad f_1 \le k_1$

(6.2)–(6.6), (6.20)

and make the similar conclusions as in (1′).

Finding the values of e_1 and e_2 is another issue that need to be considered.

6.4.3 Simultaneous Optimization Problem

Recall that we defined a function g which is non-decreasing in f_1 and f_2. If g is known and linear, the following single objective problem can be defined

$g(f_1, f_2) = wf_1 + (1-w)f_2$

where the objective is a convex combination of f_1 and f_2 for $0 \le w \le 1$. For a prespecified w, the problem can be solved by any mixed integer programming software.

If g is nonlinear, then the problem becomes a nonlinear integer model which is harder to solve one.

If g is linear but w is unknown, or f is nonlinear (known or unknown) an optimal solution is one of the efficient solutions with respect to f_1 and f_2. A solution s is called efficient with respect to f_1 and f_2 if there does not exist a solution s′ such that $f_1(s')\leq f_1(s)$ and $f_2(s')\leq f_2(s)$. To generate all efficient solutions, we can benefit from the procedures derived for hierarchical and constrained optimization problems.

Haimes *et al.* (1971) show that an optimal solution to a constrained optimization problem may give an efficient point. The *e*-constrained approaches in multi-criteria optimization vary the right hand side of the constraint systematically and find all efficient solutions (see Steuer, 1986). We may follow such an approach to generate all efficient solutions with respect to f_1 and f_2. To start such a procedure, we need upper bounds on the f_1 and f_2 values of all efficient solutions. Note that such values can be found by the single criterion optimization of f_2 and f_1. An optimal f_1 value that solves Min f_2 problem is an upper bound on the f_1 values of all efficient solutions. Similarly, the optimal f_2 value in Min f_1 problem is an upper bound on the f_2 values of all efficient solutions. We hereafter let f_i^{UB} denote an upper bound on the f_i values of all efficient solutions. Clearly a lower bound on the f_i values of all efficient solutions is the f_i value that solves Min f_i problem. We hereafter let f_i^{LB} denote a lower bound on the f_i values of all efficient solutions.

We now discuss the stepwise description of our approach to generate all efficient solutions. Each iteration of the algorithm finds an efficient solution.

Step 0. Find lower and upper bounds on the f_1 value of all efficient solutions by solving Min f_1 and Min f_2 problems, respectively.

Let f_1^{LB} and f_1^{UB} be the optimal solutions.

Let B= f_1^{UB} and k=0.

Step 1. Solve

$$\begin{aligned} &Min\ f_2 + e_1 f_1 \\ &s.t.\quad \sum_i u_i Y_i \leq B \\ &\qquad (6.2)-(6.6),(6.20) \end{aligned}$$

k=k+1.

Step 2. Let (f_1*, f_2*) be the solution.

If $f_1= f_1^{LB}$ stop, all k efficient solutions are generated.

Let B= f_1*-1

Go to Step 1.

Alternately, we could use the problem

$$\begin{aligned}
&Min\ f_1 + e\, f_2 \\
&s.t. \quad \sum_k c_k R_k \le P \\
&\qquad (6.2) - (6.6), (6.20)
\end{aligned}$$

in Step 1 to find f_2^{LB} and f_2^{UB} in Step 0 by solving Min f_2 and Min f_1 problems, respectively.

Generating all efficient solutions is an important problem once a decision maker does not know his/her utility function explicitly. Having seen the set of all solutions, he/she may define the favorable ones.

On the other hand, for a utility function $g(f_1,f_2)$ that is non-decreasing in f_1 and f_2, we have an optimal solution in the efficient set. Hence in place of finding an optimal solution, it may be easier to generate all efficient solutions and select the one that minimizes g.

CHAPTER 7

CONCLUSIONS AND DIRECTIONS FOR FUTURE RESEARCH

In this study, we address the tactical level capacity allocation problem in Flexible Manufacturing Systems. Our problem is to assign the operations and their associated tools to Computer Numerically Controlled (CNC) machines. We consider two different objective functions. First, we assume that there are limited number of tools of each type and try to maximize the total weight over all assignments. Next, we remove the upper bound on the number of tools that can be used and assume that tools of each type can be purchased/used at a cost. Our aim is to maximize the total weight minus the total tooling cost. We model the problems as integer linear programs and prove that they are NP-hard in the strong sense.

For the total weight problem, we develop several upper bounds that are overestimates of the optimal objective function values. The optimal solutions of the continuous relaxations of all the integer variables and partial relaxations of integer variables are used in upper bounds. We also strengthen these upper bounds by adding some valid inequalities. We also propose an upper bound that is based on the upper bounds on the number of the operations that can be assigned to the machines, that we call cardinality based upper bound. Our computational experiments show that the linear relaxation and cardinality based upper bounds are very easy to compute. Although they are not very close to the optimal solutions, they can be used to evaluate the partial solutions in implicit enumeration techniques. Other upper bounds give near optimal results, however at an expense of higher computational time.

We use branch and bound algorithm to find exact solutions to the total weight problem that uses the linear relaxation based upper bounds. We develop some optimality properties and use along with other reduction mechanisms to reduce the problem size. The results of our computational experiments reveal that we can solve large size problem instances with up to 500 operations and 2 machines to optimality in reasonable times.

We also develop several lower bounds that give good feasible solutions in reasonable times. These lower bounds are due to the priority based heuristics, tabu search, best improving search and beam search algorithms. The priority based heuristic is not a good estimate of the optimal solution; however it may be used to obtain initial feasible solutions in optimization and improving type algorithms. Our experiments have shown that the tabu search performs quite satisfactory in obtaining near optimal feasible solutions. However, we observe that beam search algorithm gives superior results in very short time. We find optimal solutions for 78% of the problem instances by the beam search algorithm and the deviations from the optimal solution for most of the other problem instances are less than 1%.

For the bicriteria problem that trade-offs between total weight and total cost, we suggest a solution approach that is based on Lagrangean relaxation idea. By the proposed Lagrangean relaxation approach we obtain both lower and upper bounds on the objective function values. We employ subgradient optimization technique to solve the Lagrangean problems. We test the performance of the algorithm by the deviations of the lower and upper bounds from the optimal solution for small size problems with 3 machines. We also compute the deviations between the lower and upper bounds for the problem instances with up to 500 operations and 7 machines. The results of our computational experiments show that the deviations are very small for all problem instances over all combinations. We observe around 5% deviation between the lower and upper bounds even for the largest size problems.

We now discuss two noteworthy extensions of our study: tool selection and capacity expansion problems.

i. The Tool Selection Problem: We assume that the tools required for an operation are known and fixed. However, in practice, several alternative tools may be used for processing an operation. This option may be treated easier when each operation requires a single tool. The costs of the alternative tools differ in the sense that they may have different purchasing costs or the quality of the operation may be affected by the tool used. Future research may extend our procedures derived for the total weight problem so as to incorporate tool selection decisions.

ii. The Capacity Expansion Problem: We assume that the time and tool magazine capacities of the machines are fixed. However, in some manufacturing environment, these capacities can be expanded at some cost using overtime, extra shifts or subcontracting. In such environments, clearly the benefit-cost trade-off appears. Increasing the capacities leads to an increase in the total weight (or profit) as it gives room for more operation assignment, however we have to pay a cost for the associated capacity expansion. Future research may consider the extension of our procedures derived for the bicriteria problem to the objective of maximizing total weight minus total cost of increasing the capacity.

REFERENCES

Akçalı E., Üngör A., Uzsoy, R., (2005), "Short-term capacity allocation problem with tool and setup constraints", *Naval Research Logistics*, 52, 754-764.

Beasley J.E., (1995), *Lagrangean Relaxation*. In: C. R. Reeves. Modern heuristic techniques for combinatorial problems, McGraw-Hill: New York, 243-303.

Berrada M., Stecke K.E., (1986), "A branch and bound approach for machine load balancing in flexible manufacturing systems", *Management Science*, 32, 1316-1335.

Bilgin S., Azizoğlu M., (2006), "Capacity and tool allocation problem in flexible manufacturing systems", *Journal of the Operational Research Society*, 57, 670-681.

Chekuri C., Khanna S., (2000), "A PTAS for the multiple knapsack problem", *Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 213-222.

Chen F.F., Ker J. –I., Kleawpatinon K., (1995), "An effective part-selection model for production planning of flexible manufacturing systems", *International Journal of Production Research*, 33, 2671-2683.

Çatay B., Erengüç Ş.S., Vakharia A.J., (2003), "Tool capacity planning in semiconductor manufacturing", *Computers & Operations Research*, 30, 1349-1366.

Çatay B., Erengüç Ş.S., Vakharia A.J., (2005), "Capacity allocation with machine duplication in semiconductor manufacturing", *Naval Research Logistics*, 52, 659-667.

D'Alfonso T.H., Ventura J.A., (1995), "Assignment of tools to machines in a flexible manufacturing system", *European Journal of Operational Research*, 81, 115-133.

Dawande M., Kalagnanam J., Keskinocak P., Ravi R., Salman F.S., (2000), “Approximation algorithms for the multiple knapsack problem with assignment restrictions”, *Journal of Combinatorial Optimization*, 4, 171-186.

Della Croce F., Ghirardi M., Tadei R., (2004), “Recovering beam search: Enhancing the beam search approach for combinatorial optimization problems”, *Journal of Heuristics*, 10, 89-104.

Denizel M., Erengüç S.S., (1997), “Exact solution procedures for certain planning problems in flexible manufacturing systems”, *Computers & Operations Research*, 24, 1043-1055.

Fisher M.L., (1981), “The Lagrangean Relaxation method for solving integer programming problems”, *Management Science,* 27, 1-18.

Fisher M.L., (1985), “An applications oriented guide to Lagrangean Relaxation”, *Interfaces*, 15, 10-21.

Ghirardi M., Potts C.N., (2005), “Makespan minimization for scheduling unrelated parallel machines: A recovering beam search approach”, *European Journal of Operational Research*, 165, 457-467.

Glover F., (1986), “Future paths for integer programming and linkage to artificial intelligence”, *Computers and Operations Research*, 13, 533-549.

Glover F., Laguna M., (1997), *Tabu Search*, Kluwer Academic Publishers: London.

Grieco A., Semeraro Q., Tolio T., (2001), “A review of different approaches to the FMS loading problem”, *The International Journal of Flexible Manufacturing Systems*, 13, 361-384.

Gray A.E., Seidmann A., Stecke K.E., (1993), “A synthesis of decision models for tool management in automated manufacturing”, *Management Science*, 39, 549-565.

Haimes Y.Y., Wismer D.A., Lasdon L.S., (1971), “On bicriteria formulation of the integrated systems identification and system optimization”, *IEEE Transactions on Systems, Man. And Cybernetics*, 1, 296-297.

Held, M.H., Wolfe, P., Crowder, H.D., (1974), "Validation of Subgradient Optimization", *Mathematical Programming*, 6, pp. 62-88.

Hso V.N., De Matta R., (1997), "An efficient heuristic approach to recognize the infeasibility of a loading problem", *The International Journal of Flexible Manufacturing Systems*, 9, 31-50.

Kellerer H., Pferschy U., Pisinger D., (2004), *Knapsack Problems*, Springer: Berlin.

Kim Y.-D., Lee G.-C., Lim S.-K., Choi S.-K., (2003), "Tool requirements planning in a flexible manufacturing system: minimizing tool costs subject to a makespan constraint," *International Journal of Production Research*, 41, 3339-3357.

Liang M., Dutta S. P., (1993), "An integrated approach to the part selection and machine loading problem in a class of flexible manufacturing systems", *European Journal of Operational Research*, 67, 387-404.

Marquez F.P., Arenales M.N., (2007), "The constrained compartmentalized knapsack problem", *Computers & Operations Research*, 34, 2109-2129.

Martello S., Toth P., (1990), *Knapsack problems algorithms and computer implementations*, John Wiley & Sons: NewYork.

Morton T.E., Pentico D.W., (1993), *Heuristic scheduling systems*, Wiley: New York.

Nemhauser G.L., Wolsey L.A., (1988), *Integer and combinatorial optimization,* John Wiley & Sons: New York.

Pisinger D., (1999), "An exact algorithm for large multiple knapsack problems", *European Journal of Operational Research*, 114, 528-541.

Ram B., Sarin S., Chen C.S., (1990), "A model and a solution approach for the machine loading and tool allocation problem in a flexible manufacturing system", *International Journal of Production Research*, 28, 637-645.

Sarin S.C., Chen C.S., (1987), “The machine loading and tool allocation problem in a flexible manufacturing system”, *International Journal of Production Research*, 25, 1081-1094.

Shachnai H., Tamir T., (2001a), “On two class-constrained versions of the multiple knapsack problem”, *Algorithmica*, 29, 442-467.

Shachnai H., Tamir T., (2001b), “Polynomial time approximation schemes for class-constrained packing problems”, *Journal of Scheduling*, 4, 313-338.

Shanker K., Srinivasulu A., (1989), “Some solution methodologies for loading problems in a flexible manufacturing system”, *International Journal of Production Research*, 27, 1019-1034.

Shanker K., Tzen Y. –J. J., (1985), “A loading and dispatching problem in a random flexible manufacturing system”, *International Journal of Production Research*, 23, 575-595.

Sodhi M.S., Askin R.G., Sen S., (1994), “Multiperiod tool and production assignment in flexible manufacturing systems”, *International Journal of Production Research*, 32, 1281-1294.

Stecke K.E., (1983), “Formulation and solution of nonlinear integer production planning problems for flexible manufacturing systems”, *Management Science,* 29, 273-288.

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

Swarnkar R., Tiwari M.K., (2004), “Modeling machine loading problem of FMSs and its solution methodology using a hybrid tabu search and simulated annealing-based heuristic approach”, *Robotics and Computer-Integrated Manufacturing*, 20, 199-209.

Toktay L.B., Uzsoy R., (1998), “A capacity allocation problem with integer side constraints”, *European Journal of Operational Research*, 109, 170-182.

Valente J.M.S., Alves R.A.F.S., (2006), “Beam search algorithms for the single machine total weighted tardiness scheduling problem with sequence-dependent setups”, *Computers & Operations Research*, doi: 10.1016/j.cor.2006.11.004.

Ventura J.A., Chen F.F., Leonard M.S., (1988), “Loading tools to machines in flexible manufacturing systems”, *Computers and Industrial Engineering*, 15, 223-230.

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Özpeynirci, Selin
Nationality: Turkish (TC)
Date and Place of Birth: 30 October 1980, İzmir
Marital Status: Married

EDUCATION

Degree	**Institution**	**Year of Graduation**
MS	METU Industrial Engineering	2004
BS	METU Industrial Engineering	2002
High School	Karşıyaka High School, İzmir	1998

WORK EXPERIENCE

Year	**Place**	**Enrollment**
2002-2007	METU Industrial Engineering	Research Assistant

FOREIGN LANGUAGE

Advanced English, Beginner French

PUBLICATIONS

1. Bilgin S., Azizoğlu M., (2007), "Operation assignment and capacity allocation problem in automated manufacturing systems", *Computers and Industrial Engineering*, forthcoming.

2. Bilgin S., Azizoğlu M., (2007) "A branch and bound algorithm for operation assignment problem in flexible manufacturing systems", *Proceedings of 27th National Conference on Operations Research/Industrial Engineering*, 692-697.

3. Bilgin S., Azizoğlu M., (2006), "Capacity and tool allocation problem in flexible manufacturing systems", *Journal of the Operational Research Society*, 57, 670-681.

4. Bilgin S., Azizoğlu M., (2005) "Capacity and tool allocation problem in flexible manufacturing systems", *Proceedings of 35th International Conference on Computers and Industrial Engineering*, 309-314.