A CAPACITY ALLOCATION PROBLEM IN FLEXIBLE MANUFATURING SYSTEMS

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

OF

THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

SELİN BİLGİN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE

IN

THE DEPARTMENT OF INDUSTRIAL ENGINEERING

APRIL 2004

Approval of the Graduate School of Natural and Applied Sciences.

Prof. Dr. Canan Özgen
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Çağlar Güven
Head of Department

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

Prof. Dr. Meral Azizoğlu
Supervisor

Examining Committee Members

Prof. Dr. Meral Azizoğlu _____

Asst. Prof. Dr. Ferda Can Çetinkaya _____

Prof. Dr. Ömer Kırca _____

Assoc. Prof. Dr. Nur Evin Özdemirel _____

Assoc. Prof. Dr. Canan Sepil _____

ABSTRACT

A CAPACITY ALLOCATION PROBLEM IN FLEXIBLE MANUFACTURING SYSTEMS

BİLGİN, Selin

M. Sc. Thesis, Department of Industrial Engineering
Supervisor: Prof. Dr. Meral Azizoğlu

April 2004, 67 pages

In this study, we consider a capacity allocation problem in flexible manufacturing systems. We assume time and tool magazine capacities on the Numerical Controlled (NC) machines and limited number of available tools. Our problem is to allocate the available capacity of the NC machines to the required demand of the operations, so as to maximize the total weight of operation assignments. We formulate the problem as a Mixed Integer Linear Program and show that it is NP-hard in the strong sense. We solve the moderate-sized problems optimally by the available Integer Programming software. We also develop Lagrangean relaxation based upper bounds and several heuristic procedures. Our computational results have revealed that the Lagrangean upper bounds are very close to optimal solutions and the heuristic procedures produce near optimal solutions in very small solution times even when the problem sizes are large.

Keywords: Flexible Manufacturing Systems, Capacity Allocation, Lagrangean Relaxation

ÖZ

ESNEK İMALAT SİSTEMLERİNDE KAPASİTE PAYLAŞTIRMA PROBLEMİ

BİLGİN, Selin

Yüksek Lisans Tezi, Endüstri Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Meral Azizoğlu

Nisan 2004, 67 sayfa

Bu çalışmada, Esnek İmalat Sistemlerinde kapasite yerleştirme problemini ele aldık. Sayısal Denetimli (SD) makinelerde zaman ve makine ucu haznesi kapasitelerinde kısıtlamalar olduğunu ve sınırlı sayıda makine ucu bulunduğunu varsaydık. Problemimiz SD makinelerinin kapasitelerine operasyonları gereken talebi karşılayacak şekilde paylaştırarak toplam ağırlığı maximize etmektir. Problemimizi Karmaşık Tamsayılı Doğrusal Model olarak formüle ettik ve NP-Zor olduğunu gösterdik. Orta-boyutlu problemlerimizi mevcut tam sayılı programlama yazılım programıyla optimal olarak çözdük. Ayrıca Lagrange gevşetim tekniğiyle üst sınırlar ve sezgisel yöntemler geliştirdik. Deney sonuçlarımız Lagrange üst sınırlarının optimal sonuçlara çok yakın olduğunu ve sezgisel yöntemlerimizin büyük problemlerde bile kısa zamanda optimale yakın sonuçlar ürettiğini göstermiştir.

Anahtar Kelimeler: Esnek İmalat Sistemleri, Kapasite Yerleştirme, Lagrange Gevşetim

To my mom, my dad and my love…

ACKNOWLEDGEMENTS

I would like to express my gratitude to Prof. Dr. Meral Azizoğlu for her wonderful supervision. I really appreciate her untiring support and kindness. Studying with her was a great experience for me.

I would also present my special thanks to two magnificent people, my parents Meral Bilgin and Hasan Şener Bilgin. I am so lucky that I have their unlimited belief and love.

And a special thanks is due my fiancée N. Özgür Özpeynirci for everything he added to my life. With his both technical and moral support, I could perform such a good work.

I also express my thanks to all my friends for their kind encouragements.

TABLE OF CONTENTS

LIST OF TABLES

TABLE

LIST OF FIGURES

FIGURE

CHAPTER 1

INTRODUCTION

Flexible **M**anufacturing **S**ystems (FMS) are defined as integrated systems of computer numerically controlled (CNC) machines connected with automated material handling. 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 of 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 manufacturing industry and 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.

The main purpose of flexible manufacturing systems 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 and/or optimize some performance criteria.

In flexible manufacturing systems, tool management is another vital issue since large number of tools are required for processing the operations and the NC 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, available machining time, etc. Selecting the operations to be processed by considering the tooling constraints at the same time makes the problem much more complicated.

In the literature, there are a large number of studies on flexible manufacturing systems; various methodologies and systematic approaches have been proposed to solve the problems addressed. The general problem is stated as follows: given a fixed number of part types whose operations are to be processed on the machines carrying tool magazines of limited capacity, determine the assignment of operations and allocation of tools. In the literature, integer programming has been the primary modeling approach for these problems. In majority of the previous studies, capacity allocation problem is analyzed for operations and tools separately. However, little effort has been made to handle simultaneous assignment of operations and tools. These two tactical level allocation 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 allocation to machines and machine capacity allocations 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 thesis, we consider the tactical level problem of assigning operations together with their required tools to machines in a flexible manufacturing system 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. We assume the weights are not only associated to the operation characteristics but also to the machines. The amount of inventory for each operation in terms of time units is known. This can be interpreted as the demand or maximum production quantity. 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 capacity of machines in terms of time units are other constraining factors. The primary decisions are the operation selection and their assignment to the machines. 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 inventory of operations and the required tools for processing those operations, so that the total weight is maximized.

We formulate the problem as a Mixed Integer Programming model and prove that it is NP-hard in the strong sense. We make use of upper and lower bounding procedures in order to obtain near optimal solutions. Lagrangean relaxation technique is used for finding upper bounds. Several heuristics are developed that give good lower bounds in a very short time.

This thesis consists of six chapters, which are organized as follows:

In Chapter 2, the problem is defined and its mathematical formulation is discussed. The parameters, the decision variables and the constraints are explained in detail. Main assumptions considered throughout the study and the complexity of the problem are presented in this chapter.

The literature on flexible manufacturing systems and the capacity allocation problem is reported in Chapter 3. The related work is classified according to the tooling considerations.

In Chapter 4, we define our solution approaches. We present Lagrangean relaxation technique as the upper bounding procedure and discuss the application of this technique to our problem. Also heuristics developed to find near-optimal solutions are presented.

In Chapter 5, we discuss our experimental design, the parameters used to generate problem instances and the performance measures used to evaluate quality of the solution approaches. Later on, the results of our experiments are discussed.

We conclude the study and give suggestions for future research in Chapter 6.

CHAPTER 2

PROBLEM DEFINITION

In this chapter, we first define our problem with its underlying assumptions and then give the mathematical model of the problem. We finally discuss the complexity status of the problem.

2.1 Problem Statement

In our problem environment, there are n operations to be processed by a set of m machines. Each operation can be processed on more than one machine, i.e. operation splitting is allowed. We let w_{ij} be the weight of each unit of operation i (i=1,…,n) processed on machine j (j=1,…,m). w_{ij} can be interpreted as the unit profit brought by operation i if processed by machine j. Alternatively, w_{ij} can represent the assignment cost when it receives a negative value. W_i is the amount of inventory of operation i on hand. C_j is the time capacity of machine j. W_i and C_j are measured in same units, say in the minutes.

All machines are flexible in the sense that they function according to the loaded tools. Machine j has a tool magazine capacity of s_j tool slots. There are t tool types in the system. Due to the technological restrictions and/or budget limitations, a maximum of r_k tools of type k (k=1,…,t) are available in the system. To process operation i, a set of tools $l(i)$ should be available on the tool magazine(s) of the associated machine(s).

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.

Throughout this study, we make the following additional assumptions:

- Each machine can process each operation. In case of assignment restrictions, w_{ij} can be set a negative value; to guarantee that operation i is not assigned to machine j.
- The tool magazines of the machines are initially empty.
- All parameters i.e. W_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 not subject to change, i.e. the system is static.
- Each tool requires one tool slot.

2.2 Mathematical Formulation

In this section, we first redefine our indices, parameters and decision variables. Then, we give the mathematical representation of the problem.

Indices:

i: operation index

j: machine index

k: tool index

Parameters:

n: number of operations

m: number of machines

t: number of tool types

W_i: inventory 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

Decision variables:

X_{ij}: the amount of operation i assigned to machine j

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

Constraints:

- The total amount of operation i assigned to all machines does not exceed its on hand inventory.

$$\sum_{j=1}^{m} X_{ij} \le W_i \qquad \forall i \tag{2.1}$$

- The total amount of operations assigned to machine j does not exceed its capacity.

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

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

$$\sum_{k=1}^{t} Z_{kj} \le s_j \qquad \forall j \tag{2.3}$$

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

$$\sum_{j=1}^{m} Z_{kj} \le r_k \qquad \forall k \tag{2.4}$$

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

$$\sum_{(i|k \in l(i))} X_{ij} \le \text{Min}\left\{ C_j, \sum_{(i|k \in l(i))} W_i \right\} Z_{kj} \qquad \forall k, j \tag{2.5}$$

- X_{ij}'s are nonnegative continuous variables and Z_{kj}'s are binary variables.

$$X_{ij} \ge 0 \qquad \forall i, j \tag{2.6}$$

(2.7)

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

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

$$\text{Maximize} \quad \sum_{i=1}^{n}\sum_{j=1}^{m} w_{ij} X_{ij} \tag{2.8}$$

2.3 Complexity

When all $w_{ij}=w$, the problem reduces to the maximum total capacity utilization problem. Moreover when tooling constraints, i.e. tool magazine capacity (2.3) and tool availability (2.4) are not binding, the problem reduces to the well-known maximum flow problem with the following analogy:

Let S and T denote the source and sink nodes, respectively. Let nodes $O_1, O_2, \ldots, O_n$ represent the set of operations and nodes $M_1, M_2, \ldots, M_m$ the set of machines. The arcs can be defined as follows:

- Arcs from source node S to operation nodes O_i. The capacity of each arc (S, O_i) is the inventory of operation i on hand (W_i).
- Arcs from operation nodes O_i to machine nodes M_j. The capacity of each arc (O_i, M_j) can be assumed as infinite. Also every arc from operation nodes to machine nodes has a weight w.
- Arcs from machine nodes M_j to sink node T. The capacity of each arc (M_j, T) is the capacity of machine j (C_j).

  This representation results in the network shown in Figure 2.1.

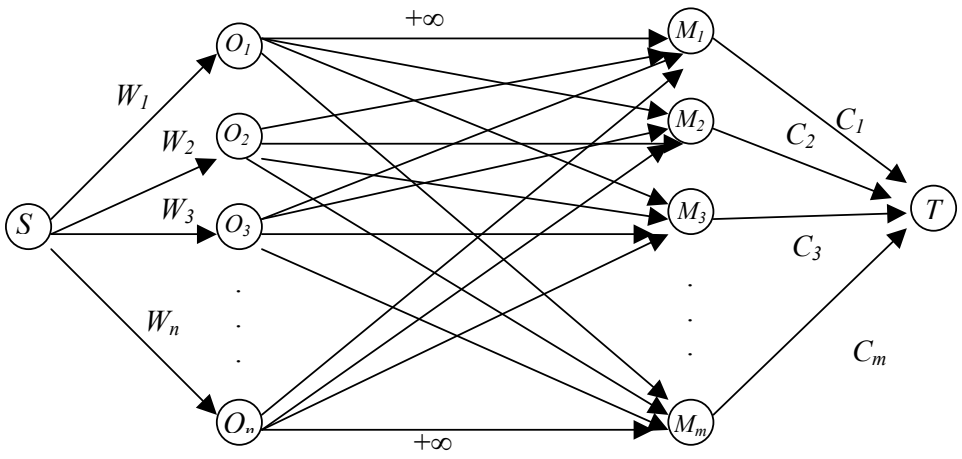


Figure 2.1. Network Representation of Maximum Flow Problem

The maximum flow problem is polynomially solvable (Papadimitriou and Steiglitz, 1982) so is our problem with identical weights and no tooling constraints. For arbitrary w_{ij} values, the problem remains polynomially solvable in the absence of tooling constraints as it can be represented as a Linear Program. However when tooling constraints are introduced, the complexity changes drastically. When all w_{ij}s are unity, $r_k=r$ for each tool k, and $s_j=s$ for each machine j, the problem reduces to the maximization of the total capacity usages. This special case of our problem is referred to as Maximum Flow Problem with degree constraints by Toktay and Uzsoy (1998). They show that this problem is NP-hard in the strong sense, so is our problem with arbitrary w_{ij}, r_k and s_j values.

CHAPTER 3

LITERATURE SURVEY

In this chapter, we summarize the previous studies on operation assignment and tool allocation problem. In the related studies there are a number of operations to be processed on the CNC machines in a work center. The capacities of the machines should be allocated to these operations according to some criterion (criteria). We classified the literature on capacity allocation problem according to whether tooling constraints are considered or not.

The capacity restrictions are due to tool magazines, machine hours and the number of tools. We may refer to our problem as capacity allocation with tooling constraints. When each operation requires a single tool and the tool requirements of the operations are distinct, the problem is referred to as capacity allocation problem with side constraints (Toktay and Uzsoy, 1998).

3.1 Capacity Allocation Problem without 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 among the different operations to be processed there. They set an upper limit on the number of setups allowed for each machine over a shift, that is a machine cannot process more than a given number of different steps during a shift. Also they put a restriction on the number of different machines that can process the same step 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 work-in-progress (WIP) processed at the work center during the shift and 2) minimizing the total deviation from predetermined production goals. They show that the problems are equivalent and conclude that they are NP-hard in the strong sense. They develop two heuristic procedures and report that their heuristics perform very well under different problem settings.

Akçalı *et al.* (2003) consider a work center with parallel machines so as to maximize the total throughput. They assume 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 interpreted 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 local search approach. They implement their heuristics on several problem sets and observe their satisfactory performance.

Çatay *et. al* (2002) study the capacity allocation problem with machine duplications in semiconductor manufacturing. They model the problem of assigning individual operations to 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 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.

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 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 results are better compared to that of Shanker and Tzen (1985). Shanker and Tzen (1985) consider a bicriteria problem of balancing the workloads among the machines and meeting the

job due dates for a random flexible manufacturing system with random job arrivals and stochastic operation times.

3.2 Capacity Allocation Problem with Tooling Constraints

D'Alfonso and Ventura (1995) study the tool assignment problem where the machines have limited tool slots in their tool magazine and tools require multiple slots. The objective is to minimize the number of daily production travels between the machines. They examine two algorithms for this problem: one is a Lagrangean relaxation approach with 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 cases.

Liang and Dutta (1993) propose an integrated approach for 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 and 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 reasonable time.

Ventura *et. al* (1988) present the tool loading problem in a flexible manufacturing system, as an assignment model. The objective is to minimize the timespan 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 modeled two problems: 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.

Another study on machine loading and tool allocation problem in flexible manufacturing systems is the one held by Sarin and Chen (1987). They aim to assign all operations of the parts and the associated tools to 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 corresponding to 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 the Lagrangean relaxation to their problem.

Chen *et. al* (1995) define the part selection problem in flexible manufacturing systems as the selection of the most cost effective set of parts to be processed simultaneously. Their objective is to maximize the total profit generated from FMS to produce only a 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 tooling and fixture constraints. Their computational experiment with the heuristics reveal that satisfactory solutions can be found in reasonably short time.

Ram *et. al* (1990) use network representation with simple side constraints for modeling the machine loading and tool allocation problem in a flexible manufacturing system, where the objective is to minimize the total cost 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 to the machines. This assignment is constrained by each machine's tool magazine capacity as well as by the production capacities expressed for the overall system and for each individual machine type. Their objective is to

balance the machine workloads. They apply 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 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.

Our study differs from the ones in the literature in the sense that we consider operation and tool assignments to the machines simultaneously with the objective of maximizing the total weight. The most closely related study to ours is Toktay and Uzsoy's (1998) study. Toktay and Uzsoy (1998) considered operation assignment with constraints on the number of machines that an operation can be assigned and the number of operations that a machine can process. However they ignore tooling constraints.

CHAPTER 4

SOLUTION APPROACH

As discussed, our problem is strongly NP-hard, as it reduces to a maximum flow problem with side constraints which is shown to be strongly NP-hard. In this study, our aim is to approach to this hard problem through bounding mechanisms. We propose some lower and upper bounding procedures and test their efficiencies by their relative closeness.

4.1 Upper Bounding Procedure (Lagrangean Relaxation)

In this section, first we discuss Lagrangean relaxation method and subgradient optimization, i.e. the method we use for solving Lagrangean problem in this study. Later, we describe the application of Lagrangean relaxation procedure to our problem.

4.1.1 Lagrangean Relaxation Method and Subgradient Optimization

Fisher (1981 and 1985) gives a review of Lagrangean relaxation technique and discusses some application areas. In the literature there are several successful applications of the technique, such as facility location, scheduling and generalized assignment problems.

Suppose we have the following integer programming problem:

(P) $Z = \text{Max } cx$

$Ax \leq b$

$Dx \leq e$

$x \geq 0$ and integer

Assume that when the constraint set $Ax \leq b$ is removed, the problem will be 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 $u=(u_1,u_2,\ldots,u_n)$ and solve the remaining 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 vector u. The value of u is an important factor on 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 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 makes the problem either a polynomially solvable one or gives a decomposable structure to the problem. In the latter case, the decomposed problems could be solved independently and relatively easier. Later we will discuss that once the constraint set (2.5) that links the operation assignments to 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 Lagrangean dual, and the following three methods are the most commonly used 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. 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:

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

where

Z^*: The objective value of the best known feasible solution to (P). 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 $Z_D(u^k)$ has failed to decrease in a specified number of iterations.

There are several stopping rules for the subgradient method (Beasley, 1995), when

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

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

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

4.1.2 Lagrangean Relaxation of Our Problem

Consider the relaxation of the constraint set (2.5) and the following Lagrangean Relaxation model (LR).

$$\text{(LR)} \quad \text{Maximize} \quad \sum_{i=1}^{n}\sum_{j=1}^{m} w_{ij}X_{ij} + \sum_{k=1}^{t}\sum_{j=1}^{m} u_{kj}(\text{Min}\{C_k, \sum_{(i|k\in l(i))} W_i\}Z_{kj} - \sum_{(i|k\in l(i))} X_{ij}) \qquad (4.2)$$

subject to (2.1), (2.2), (2.3), (2.4), (2.6) and (2.7)

where $u_{kj} \geq 0 \ \forall k,j$.

Note that (2.1) and (2.2) are only related with X_{ij}s and (2.3) and (2.4) are only related with Z_{kj}s. The objective function of (LR) can be rewritten as in (4.3).

$$\text{Maximize} \quad \sum_{i=1}^{n}\sum_{j=1}^{m} (w_{ij} - \sum_{(i|k\in l(i))} u_{kj})X_{ij} + \sum_{k=1}^{t}\sum_{j=1}^{m} (\text{Min}\{C_k, \sum_{(i|k\in l(i))} W_i\}u_{kj})Z_{kj} \qquad (4.3)$$

Note that the objective function can be separated in two parts: one as a function of X_{ij}'s and one as a function of Z_{kj}'s. These nice separations lead to two independent subproblems SLR1 and SLR2 each of which is stated below.

(SLR1)

$$\text{Maximize} \quad \sum_{i=1}^{n}\sum_{j=1}^{m} (w_{ij} - \sum_{(i|k\in l(i))} u_{kj})X_{ij} \qquad (4.4)$$

subject to

$$\sum_{j=1}^{m} X_{ij} \leq W_i \qquad \forall i \qquad (4.5)$$

$$\sum_{i=1}^{n} X_{ij} \leq C_j \qquad \forall j \qquad (4.6)$$

$$X_{ij} \geq 0 \qquad \forall i,j \qquad (4.7)$$

(SLR2)

$$\text{Maximize} \quad \sum_{k=1}^{t}\sum_{j=1}^{m}(\text{Min}\{C_k, \sum_{(i|k\in I(i))} W_i\}u_{kj})Z_{kj} \tag{4.8}$$

subject to

$$\sum_{k=1}^{t} Z_{kj} \le s_j \qquad \forall j \tag{4.9}$$

$$\sum_{j=1}^{m} Z_{kj} \le r_k \qquad \forall k \tag{4.10}$$

$$Z_{kj} \in \{0,1\} \quad \text{and integer} \qquad \forall k, j \tag{4.11}$$

SLR1 is a transportation problem that can be solved in a polynomial time.

SLR2 is a pure integer problem. However, it has very nice property that its optimal solution and the optimal solution to its LP relaxation are identical. In other words, its constraint matrix A is totally unimodular. For the sake of completeness we first define total unimodularity property, give some theorems and then show that our problem has this property.

Definition (Wolsey, 1998) An integer matrix A is called *totally unimodular* (TUM) if every square, nonsingular submatrix of A has determinant equal to 0, 1 or –1.

Theorem 4.1 (Nemhauser and Wolsey, 1988) The following conditions are sufficient to detect the unimodularity of a matrix A:

1. Each element of A is 0, 1, –1.
2. No more than 2 nonzero elements exist in each column.
3. Rows can be partitioned into 2 subsets S_1 and S_2 such that
    a) if a column contains 2 nonzero elements of the same sign, one element is in each of the subsets.
    b) if a column contains 2 nonzero elements of the opposite sign, both elements are in the same set.

Theorem 4.2 (Nemhauser and Wolsey, 1988) A matrix A is totally unimodular if any one of the matrices A^T, -A, (A,A), (A,I) is totally unimodular.

Theorem 4.3 (Nemhauser and Wolsey, 1988) If A is totally unimodular, then all the vertices of the convex polytope defined by the constraints $Ax=b$, $x\geq 0$, are integral for any integer vector b.

Theorem 4.4 Constraint matrix A of SLR2 has total unimodularity property.

Proof The proof can be done in two ways.

1st way. Note that constraint matrix A can be written as follows:

$$A = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ . & . & . & . & . & . & . & . & . \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{matrix} \begin{matrix} \\ \\ \\ \} Set1 \\ \\ \\ \} Set2 \\ \\ \\ \} Set3(I) \\ \\ \end{matrix}$$

The proof will be done by induction.

Note that determinant $A_k=0$ or 1 for k=1 as all entries are 0 or 1. Now we will assume that all (k-1)*(k-1) submatrices are unimodular and show that all k*k submatrices are unimodular as well.

Two cases arise:

Case 1: There is a row from Constraint Set 3. Two subcases should be considered:

Case1.1 There is a row with single 1.

This implies $\det(A_k)=\pm\det(A_{k-1})$

Case 1.2 There is a row with all entries zero.

This implies $\det(A_k)=0$

Case 2: There is no row from Constraint Set 3. Three subcases should be considered:

Case 2.1 There is a column with all entries zeros.

This implies $\det(A_k)=0$

Case 2.2 There is a column with a single 1.

This implies $\det(A_k)=\pm\det(A_{k-1})$

Case 2.3 All columns have two 1's, one at the origin and the other at the destination. This implies the rows are linearly dependent and therefore

$\det(A_k)=0$

Note that in all cases, $\det(A_k)= \pm 1$ or 0. Therefore, we can conclude that constraint matrix A of SLR2 is totally unimodular.

2nd way. We can decompose matrix A into (A′,I) as follows:

$$A = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ . & . & . & . & . & . & . & . & . \end{bmatrix} \begin{matrix} \left.\vphantom{\begin{matrix}a\\a\\a\\a\\a\\a\end{matrix}}\right\} A' \\ \left.\vphantom{\begin{matrix}a\\a\\a\end{matrix}}\right\} I \end{matrix}$$

Rows of A′ can be partitioned into two subsets S_1 and S_2.

$$A' = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & . & . \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & . & . \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & . & . \\ . & . & . & . & . & . & . & . & . & . & . \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & . & . \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & . & . \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & . & . \\ . & . & . & . & . & . & . & . & . & . & . \end{bmatrix} \begin{matrix} \left.\vphantom{\begin{matrix}a\\a\\a\end{matrix}}\right\} S_1 \\ \left.\vphantom{\begin{matrix}a\\a\\a\end{matrix}}\right\} S_2 \end{matrix}$$

It can easily be seen that A′ satisfies the sufficient conditions, so it is totally unimodular. Therefore, according to Theorem 4.1, A'^T and (A'^T,I) are totally unimodular. So is $(A'^T,I)^T$ which gives A. We can conclude that A is totally unimodular. □

Combining the results of Theorems 4.3 and 4.4, we can conclude that an LP relaxation of SP2 produces a solution in which all Z_{kj}s are either 0 or 1. Hence SLR2 can be solved by LP technique in polynomial time.

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_{kj}^{0}=0, Z*=0,λ_0=2.

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

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 step size t_p

$$t_p = \frac{\lambda_p(Z_D(u^p) - Z^*)}{\sum_{k=1}^{t}\sum_{j=1}^{m}(\text{Min}\{C_j, \sum_{(i|k\in I(i))} W_i\}Z_{kj} - \sum_{(i|k\in I(i))} X_{ij})^2}$$

Step 5. Update Lagrangean multipliers

$$u_{kj}^{p+1}=\text{Max}\ \{0, u_{kj}^{p}-t_p(\ \text{Min}\ \{C_j, \sum_{(i|k\in I(i))} W_i\ \}Z_{kj}- \sum_{(i|k\in I(i))} X_{ij}\ )\}$$

Return to Step 2.

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

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

We now discuss our Lagrangean heuristic to be used in Step 2 of the above procedure. We develop a myopic heuristic procedure that only considers the operations assigned by the Lagrangean solution. The heuristic selects the operation and machine pair having the maximum weight among the ones that can be assigned without violating tooling constraints. Below is the stepwise description of our Lagrangean heuristic.

Step 1. Let N_L be the set of operations assigned according to the Lagrangean upper bound solution.

Step 2. Find the operation-machine pair with maximum weight from set N_L. Call the pair as operation i' and the machine j'. Check whether

- the tool magazine of machine j' has enough number of tool slots for required tools of operation i',

- the tools required to process operation i' are available on hand.

Step 3. If both of the above conditions are satisfied, assign the tools required to process operation i' to machine j'. $Z_{kj'}=1$ $k \in l(i')$. Update $W_{i'}$, $C_{j'}$, $s_{j'}$, r_k and set N_L.

Step 4. Return to Step 2 until all operation-machine assignments are considered.

4.1.3 Strengthening the Lagrangean Solution

In this subsection, we discuss some approaches we incorporated into the Lagrangean relaxation formulation to increase its strength. In doing so, we add some constraints to SLR1 that will reduce the value of the upper bound without violating its validity. The added two constraints are given below:

1. $\sum_{i=1}^{n}\sum_{j=1}^{m} w_{ij}X_{ij} \le UB$

This constraint is added in order to set an upper bound on the optimal solution value of the Lagrangean problem. Note that, in a maximization problem, obtaining smaller upper bounds is always desirable. One such upper bound is available through LP relaxation. Note that the LP relaxation solution is a valid upper bound and can be obtained in polynomial time. When we use LP relaxation solution, i.e. Z_{LP}, as the UB, we guarantee that the resulting Lagrangean relaxation solution dominates the optimal LP relaxation solution.

2. $\sum_{i=1}^{n}\sum_{j=1}^{m} w_{ij}X_{ij} \ge LB$

Note that we assume $w_{ij}>0$, however $w_{ij} - \sum_{(i|k\in l(i))}^{n} u_{kj}$ can be negative, and the operations with negative coefficients never appear in the Lagrangean solution. When we add the above constraint, some negatively weighted operations appear in the Lagrangean solution and the upper bound becomes tighter. Initially we use the best solution of our heuristic procedures (that will be discussed in the next section) as LB. We update LB whenever our Lagrangean heuristic provides a better solution.

4.2 Lower Bounding Procedures

Several heuristic procedures are developed in order to obtain near optimal solutions to our problem in polynomial time. These heuristics are explained below:

4.2.1 Lower Bound 1 (Greedy Heuristic)

Lower Bound 1 operates myopically each time selecting an operation and machine pair whose contribution to the objective function is the maximum.

We sort the operation-machine assignments in non-increasing order of the w_{ij} values. For the next operation-machine pair in the list, we check the feasibility of the assignment with respect to the tool magazine capacity of the machine and the availability of the required tools for processing the operation. If the feasibility conditions are satisfied, we assign the operation to the machine and update W_i, C_j, s_j and r_k values. If not, we consider the next operation-machine pair in the list. Below is the stepwise description of the greedy heuristic:

Step 0. Sort the operation-machine pairs in non-increasing order of the w_{ij} values.

Step 1. If the list is empty, stop.

Let (i', j') be the next operation-machine pair in the list.

Let $l(i')$ be the additional tools required when operation i' is put on machine j'.

Check whether $s_{j'}>|l(i')|$ and $r_k>0$ for $k\in l(i')$.

Step 2. If any one of the conditions stated in Step 1 is not satisfied, return to Step 1.

Assign operation i' to machine j', i.e. let $X_{i'j'}=\text{Min}\{W_{i'},C_{j'}\}$.

Set $W_{i'}=W_{i'}-X_{i'j'}$

$C_{j'}=C_{j'}-X_{i'j'}$

$s_{j'}=s_{j'}-|l(i')|$

$r_k=r_k-1$ for $k\in l(i')$

Go to Step 1.

4.2.2 Lower Bound 2 (One-at-a-time Assignment)

Lower Bound 2 first ignores the tooling constraints and makes the optimal assignment and then try to resolve the infeasibilities brought due to the tooling constraints. In doing so, it calculates the relative importance of the tools on machines where the importance is defined as the amount of processing to be lost when the tool is removed from the machine's tool magazine. The tools that result with the maximum total improvement are kept. The heuristic then fixes the tools on the machines, makes reallocation and afterwards reassigns the not-yet-allocated operations to the machines, according to the rules of Lower Bound 1 (greedy heuristic).

Below is the stepwise description of the one-at-a-time assignment heuristic:

Step 1. Optimal operation assignment: First, assign the operations to machines by relaxing the tooling constraints through the following LP model

(OPT1) Maximize $\sum_{i=1}^{n}\sum_{j=1}^{m} w_{ij}X_{ij}$

subject to $\sum_{j=1}^{m} X_{ij} \le W_i \qquad \forall i$

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

$X_{ij} \ge 0 \qquad \forall i, j$

Step 2. Calculate tool contributions (tw_{kj}) on each machine, i.e. the sum of the weights of the operations that are assigned to machine j and require tool k.

$$tw_{kj} = \sum_{(i|k\in I(i))} w_{ij}X_{ij} \qquad \forall k,j$$

Note that tw_{kj} can be interpreted as an amount of reduction in objective function when tool k is removed from machine j.

Assign the tools to machines to maximize total tool contribution by the following model:

(OPT2) Maximize $\sum_{k=1}^{t}\sum_{j=1}^{m} tw_{kj} Z_{kj}$

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

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

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

Step 3. Optimal operation assignment with the tools assigned in Step 3 is found as follows: Set w_{ij} to a negative value if any tool from *l(i)* is not assigned to machine *j*. As we are maximizing, X_{ij} will be zero if w_{ij} is negative. Solve (OPT1), i.e. the optimal operation assignment model with updated weights.

Step 4. Greedy assignment: Fixing the already assigned operations and tools, the remaining ones are allocated according to the greedy heuristic defined in Section 4.2.1. We consider the tools that are already on the tool magazines of the machines, while deciding on the operation assignments.

We now show that the constraint set of OPT2 is totally unimodular, therefore it can be solved in polynomial time. This result implies that Lower Bound 2 can be found in polynomial time, as well.

Theorem 4.5 The constraint matrix of OPT2 is totally unimodular.

Proof Constraint matrix of OPT2 is same as the constraint matrix of SLR2, which is proved to be totally unimodular in Theorem 4.4. Therefore, we can conclude that OPT2's constraint matrix is also totally unimodular. □

Hence, we can use LP relaxation of the model and still obtain all binary Z_{kj} values.

4.2.3 Lower Bound 3 (Matching-Based Heuristic)

In place of selecting one operation, machine pair at-a-time, Lower Bound 3 selects *m* 'operation-machine' pairs at-a-time. In doing so, it applies the matching algorithm and makes one-to-one assignment of operations to machines in each

iteration. 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 that results in the maximum total weight. The network representation of our matching algorithm is given in Figure 4.1

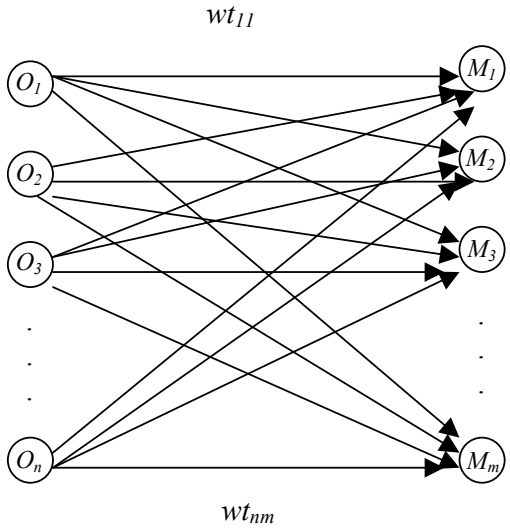


Figure 4.1 Network Representation of the Maximum Weighted Matching Problem

In the above network, the operations are represented by nodes $O_1, O_2, \ldots, O_n$ and the machines are represented by nodes $M_1, M_2, \ldots, M_m$. 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 objective function. The maximum allowable assignment is $\text{Min}\{W_i, C_j\}$ and the total weight of this assignment is $w_{ij}*\text{Min}\{W_i, C_j\}$. We try to send flow from operation nodes to machine nodes simultaneously in each iteration.

The steps of 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, wt_{ij} as $w_{ij}*\text{Min}\{W_i, C_j\}$. Set w_{ij} a negative value, if operation i cannot be assigned to machine j due to either the lack of any required tool or the violation of

the tool magazine capacity. In such a case, X_{ij}* will take on value 0 at optimality. Stop when all w_{ij}s are negative.

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

$$\text{(MA)} \quad \text{Maximize} \quad \sum_{i=1}^{n}\sum_{j=1}^{m} wt_{ij}\, y_{ij} \tag{4.12}$$

$$\text{subject to} \quad \sum_{i=1}^{n} y_{ij} \le 1 \qquad \forall j \tag{4.13}$$

$$\sum_{j=1}^{m} y_{ij} \le 1 \qquad \forall i \tag{4.14}$$

$$0 \le y_{ij} \le 1 \text{ and integer} \qquad \forall i, j \tag{4.15}$$

where

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

Objective function (4.12) maximizes the total weight of assignments. Constraint set 4.13 ensures that at most one operation is assigned to a machine. By constraint set 4.14, an operation is assigned to at most one machine. y_{ij}s are set to 0 or 1 by the constraint set 4.15.

Note that MA is an assignment problem, whose constraint set is shown to be totally unimodular. This implies that the integrality constraints are redundant.

Step 3. Assign all the required tools by the operations to the corresponding machines. Check the feasibility of the matching. If any of the tools are used more than its available number, find all operations using that tool. Remove the least weighted operation among those. Continue until a feasible assignment is reached. Stop when no further feasible assignment can be done.

Step 4. Update $W_i=W_i-X_{ij}$

$C_j=C_j-X_{ij}$

$s_j=s_j-|l(i)|$

$r_k=r_k-1 \quad \text{for } k \in l(i)$

Go to Step 1.

4.2.4 Lower Bound 4 (m-at-a-time Assignment)

In this heuristic approach, we combine the ideas underlying lower bounding procedures 2 and 3. As in the first 4 steps of one-at-a-time heuristic, we first assign operations to machines without considering tooling constraints. Then, according to this assignment, we calculate the tool contributions and assign the tools to the machines such that the total tool contribution is maximized. After updating the weights, the maximum weighted matching algorithm is used to assign the remaining operations and tools to the machines. The steps of this heuristic are as follows:

Step 1. Optimal operation assignment: First, assign the operations to machines without considering the tooling constraints. Solve the following LP model

$$\text{(OPT1)} \quad \text{Maximize} \quad \sum_{i=1}^{n}\sum_{j=1}^{m} w_{ij}X_{ij}$$

$$\text{subject to} \quad \sum_{j=1}^{m} X_{ij} \le W_i \qquad \forall i$$

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

$$X_{ij} \ge 0 \qquad \forall i, j$$

Step 2. Calculate tool contributions (tw_{kj}) on each machine.

$$tw_{kj} = \sum_{(i|k\in I(i))} w_{ij}X_{ij} \qquad \forall k,j$$

Assign the tools to the machines in order to maximize the total tool contribution by the following model:

$$\text{(OPT2)} \quad \text{Maximize} \quad \sum_{k=1}^{t}\sum_{j=1}^{m} tw_{kj}Z_{kj}$$

$$\text{subject to} \quad \sum_{k=1}^{t} Z_{kj} \le s_j \qquad \forall j$$

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

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

Step 3. Set w_{ij} to a negative value if any required tool of operation i is not assigned to machine j. Stop, when all w_{ij}s are negative. Solve the optimal operation assignment model (OPT1) in Step 1 with the new weights.

Step 4. Construct a bipartite graph containing the operation and the machine nodes. Calculate the weight of each arc from operation i to machine j, by letting $wt_{ij}=w_{ij}*\text{Min}\{W_i, C_j\}$.

Step 5. Solve the following linear program:

(MA) Maximize $\sum_{i=1}^{n}\sum_{j=1}^{m} wt_{ij}\, y_{ij}$

subject to $\sum_{i=1}^{n} y_{ij} \le 1 \qquad \forall j$

$\sum_{j=1}^{m} y_{ij} \le 1 \qquad \forall i$

$0 \le y_{ij} \le 1 \qquad \forall i, j$

Step 6. Assign all the required tools to the corresponding machines. Check the feasibility of the matching. If any of the tools is used more than its available number, find all operations using that tool. Remove the least weighted operation among those. Continue until a feasible assignment is obtained. Stop when no more feasible assignment can be done.

Step 7. Update $W_i=W_i-X_{ij}$

$C_j=C_j-X_{ij}$

$s_j=s_j-|l(i)|$

$r_k=r_k-1$ for $k\in l(i)$

Go to Step 4.

4.2.5 Lower Bound 5 (Lagrangean Heuristic)

Lagrangean relaxation can be used to obtain lower bounds. Lagrangean problem rarely gives feasible solution to the original problem. Once the resulting solution is infeasible, the infeasibilities are resolved by a heuristic, which is often called Lagrangean heuristic. In our problem, the infeasibilities can be due to

assigning operations without their entire tool sets. Our Lagrangean heuristic is a greedy procedure whose steps are given below:

Step 1. Let N_L be the set of operations assigned according to the Lagrangean solution.

Step 2. Select the maximum weighted operation-machine assignment from set N_L. Let the selected operation be i' and the selected machine be j'. Check whether

a. the tool magazine of machine j' has enough tool slots to accommodate the required tools of operation i',
b. the tools required to process operation i' are available on hand.

Step 3. If both of the above conditions are satisfied, assign the tools required to process operation i' to machine j'.

Let $Z_{kj}=1$ $k \in l(i')$.

Update $W_{i'}$, $C_{j'}$, $s_{j'}$, r_k and set N_L.

Go to Step 2 until all operation-machine assignments are considered, i.e. stop when set N_L is empty.

We run the above heuristic at each iteration of Lagrangean relaxation procedure. The best solution over all iterations defines the solution of the Lagrangean heuristic.

4.2.6 A Numerical Example

We illustrate our lower bounding procedures through the following example with n=10 operations, m=3 machines, and t=10 tool types. We assume the weights w_{ij}, are arbitrary, and set $|l(i)|>1$, r_k=3 and s_j=20. The weights, the inventories of the operations and machine capacities are given in the following table.

i \ j	1	2	3	W_i
1	133	104	104	60
2	71	123	135	252
3	42	107	66	302
4	83	61	101	79
5	91	135	66	551
6	63	145	59	440
7	63	107	34	445
8	108	143	48	533
9	83	108	73	329
10	125	56	104	151
C_j	711	140	551	

The matrix that shows the tools required by each operation is given below

i \ k	1	2	3	4	5	6	7	8	9	10
1	1	0	0	0	0	0	1	0	0	0
2	0	0	0	0	1	0	0	1	0	0
3	1	0	0	1	0	0	0	0	0	0
4	1	1	0	0	0	1	0	1	0	0
5	0	0	0	0	1	0	0	0	0	1
6	1	1	1	0	1	0	0	0	0	0
7	1	0	0	0	0	0	0	1	0	0
8	0	0	1	1	0	1	0	0	0	0
9	1	0	0	1	1	0	1	0	1	0
10	0	0	0	1	1	1	1	0	0	1

Now we apply the heuristic procedures to our example problem.

Lower Bound 1 (Greedy Heuristic)

Step 0. The sorted list of operation-machine pairs in non-increasing order of w_{ij}s is as given below:

(*i,j*): (6,2), (8,2), (2,3), (5,2), (1,1), (10,1), (2,2), (8,1), (9,2), (3,2), (7,2), (1,2), (1,3), (10,3), (4,3), (5,1), (4,1), (9,1), (9,3), (2,1), (3,3), (5,3), (6,1), (7,1), (4,2), (6,3), (10,2), (8,3), (3,1), (7,3)

Step 1. $\text{Max}_{ij}\{w_{ij}\}=w_{62}$ hence $i'=6$, $j'=2$, $l(i')=\{1,2,3,5\}$

$s_{j'}=20>4=|l(i')|$ and $r_k>0$ for $k=1,2,3,5$

Step 2. Since the above conditions are satisfied, assign i' to j'.

Let $X_{62}=\text{Min}\{W_6, C_2\}=\text{Min}\{440,140\}=140$

Update $W_6=440-140=300$

$C_2=140-140=0$

$s_2=20-4=16$

$r_k=2$ for $k=1,2,3,5$

Ignore machine 2 since its capacity is full.

Step1. $\text{Max}_{ij}\{w_{ij}\}=w_{23}$ hence $i'=2$, $j'=3$, $l(i')=\{5, 8\}$

$s_{j'}=20>|l(i')|=2$ and $r_k>0$ for $k=5, 8$

Step 2. Assign i' to j', and let

$X_{23}=\text{Min}\{W_2, C_3\}=\text{Min}\{252,551\}=252$

Update $W_2=252-252=0$

$C_3=551-252=299$

$s_2=20-2=18$

$r_5=1$, $r_8=2$

Ignore operation 2 as $X_{23}=W_2$.

Step 1. $\text{Max}_{ij}\{w_{ij}\}=w_{11}$ hence $i'=1$, $j'=1$, $l(i')=\{1, 7\}$

$s_{j'}=20>|l(i')|=2$ and $r_k>0$ for $k=1,7$

Step 2. Assign i' to j', and let

X_{11}=Min{W_1, C_1} =Min{60,711}=60

Update W_1=60-60=0

C_1=711-60=651

s_1=20-2=18

r_1=1, r_7=2

Ignore operation 1, as $X_{11}=W_1$.

Step 1. $\text{Max}_{ij}\{w_{ij}\}=w_{10,1}$ hence i'=10, j'=1, $l(i')$={4, 5, 6, 10}

$s_{j'}$=18>|$l(i')$|=4 and r_k>0 for k=4, 5, 6, 10

Step 2. Assign i' to j', and let

$X_{10,1}$=Min{W_{10}, C_1} =Min{151,651}=151

Update W_{10}=151-151=0

C_1=651-151=500

s_1=18-4=14

r_4=2, r_5=0, r_6=2, r_{10}=2

Ignore operation 10 as $X_{10,1}=W_{10}$.

Step 1. $\text{Max}_{ij}\{w_{ij}\}=w_{81}$ hence i'=8, j'=1, $l(i')$={3}

$s_{j'}$=14>|$l(i')$|=1 and r_k>0 for k=3

Step 2. Assign i' to j', and let

X_{81}=Min{W_8, C_1} =Min{533,500}=500

Update W_8=533-500=33

C_1=500-500=0

s_1=14-1=13

r_3=1

Ignore machine 1 since its capacity is full.

Step 1. $\text{Max}_{ij}\{w_{ij}\}=w_{43}$ hence i'=4, j'=3, $l(i')$={1, 2, 6}

$s_{j'}$=18>|$l(i')$|=3 and r_k>0 for k=1,2,6

Step 2. Assign i' to j', and let

X_{43}=Min{W_4, C_3} =Min{79, 299}=79

Update W_4=79-79=0

C_3=299-79=220

s_3=18-3=15

r_1=0, r_2=1, r_6=1

Ignore operation 4 as $X_{43}=W_4$.

Step 1. $\text{Max}_{ij}\{w_{ij}\}=w_{93}$ hence $i'=9, j'=3$, $l(i')=\{4, 7, 9\}$

$s_{j'}=15>|l(i')|=3$ and $r_k>0$ for k=4,7,9

Step 2. Assign i' to j', and let

$X_{93}=\text{Min}\{W_9, C_3\}=\text{Min}\{329, 220\}=220$

Update W_9=329-220=109

C_3=220-220=0

s_3=15-3=12

r_4=1, r_7=1, r_9=2

Ignore machine 3 since its capacity is full.

All machines' capacities are full; therefore we terminate the procedure here. The objective function value is 159214. The resulting assignment is as follows:

i \ j	1	2	3
1	60		
2			252
3			
4			79
5			
6		140	
7			
8	500		
9			220
10	151		

Lower Bound 2 (One-at-a-time Assignment)

Step 1. The operations are assigned to the machines ignoring the tooling constraints through OPT1. The resultant assignment found by LP is as follows:

i \ *j*	1	2	3
1	60		
2			252
3			
4			79
5			
6		140	
7			
8	533		
9			187
10	118		33

Step 2. Note that tools 1, 3, 4, 5, 6, 7 and 10 are loaded on machine 1; tools 1, 2, 3, and 5 are loaded on machine 2; tools 1, 2, 4, 5, 6, 7, 8, 9 and 10 on machine 3 according to the assignments in Step 1. We report the contributions of the loaded tools in the following table:

k \ *j*	1	2	3
1	7980*	20300	21630
2		20300	7979
3	57564	20300	
4	72314		17083
5	14750	20300	51103
6	72314		11411
7	22730		17083
8			41999
9			13651
10	14750		3432

$$* tw_{kj} = \sum_{(i|k \in l(i))} w_{ij} X_{ij}$$

Solving OPT2 by LP, we assign the tools to machines so that the total tool contribution is maximized, and hence we obtain the following assignment:

k \ *j*	1	2	3
1	1	1	1
2		1	1
3	1	1	
4	1		1
5	1	1	1
6	1		1
7	1		1
8			1
9			1
10	1		1

We set w_{ij} to a negative value (say –1) if any of the tools in *l(i)* is not assigned to machine *j*. Therefore, w_{12}, w_{21}, w_{22}, w_{32}, w_{41}, w_{42}, w_{52}, w_{61}, w_{63}, w_{71}, w_{72}, w_{82}, w_{83}, w_{91}, w_{92}, $w_{10,2}$ are all set to –1. Solving OPT1 with the new weights gives the following assignment scheme:

i \ *j*	1	2	3
1	60		
2			252
3			
4			79
5			
6		140	
7			
8	533		
9			187

10	118	33

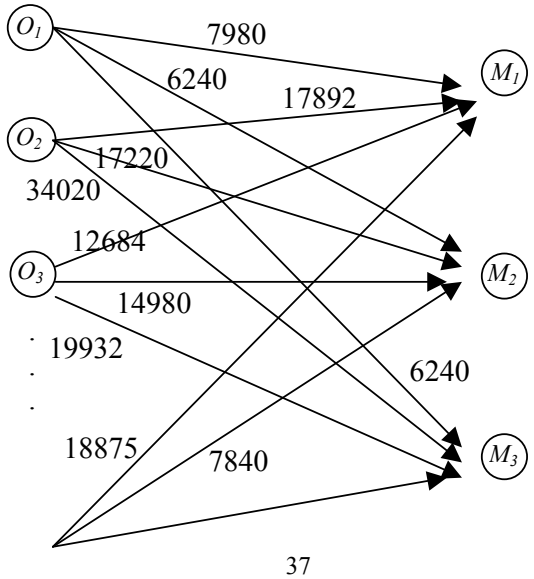
As the capacities of all machines are full, no further assignments can be done. The objective function value is 159676.

Lower Bound 3 (Matching Based Heuristic)

Step1. Construct a bipartite graph containing the operation and the machine nodes. Arcs between nodes have the following weights, and the graph below represents the network.

i \ *j*	1	2	3
1	7980*	6240	6240
2	17892	17220	34020
3	12684	14980	19932
4	6557	4819	7979
5	50141	18900	36366
6	27720	20300	25960
7	28035	14980	15130
8	57564	20020	25584
9	27307	15120	24017
10	18875	7840	15704

*$wt_{ij} = w_{ij}$*Min$\{W_i, C_j\}$



15704

Step 2. Solving the linear program for the maximum weight matching gives X_{81}=533, X_{62}=140, X_{53}=551.

Step 3. Assign all the necessary tools to the corresponding machines and see that the tooling constraints are satisfied.

Step 4. Update

W_8=0, W_6=300, W_5=0

C_1=178, C_2=0, C_3=0

s_1=17, s_2=16, s_3=18

r_3=r_5=1, r_1=r_2=r_4=r_6=r_{10}=2, r_7=r_8=r_9=3

Steps 1&2. After fixing the assigned operations and tools and solving the matching problem again, obtain $X_{10,1}$=151.

Step 3. Assigning the tools required by operation 10 to machine 1 does not violate the tooling constraints.

Step 4. Update

W_{10}=0

C_1=27

s_1=14

r_5=0, r_{10}=1, r_7=2

Steps 1&2. After fixing the assigned operations and tools and solving the matching problem again, obtain X_{11}=27.

Step 3. Assigning the tools required by operation 7 to machine 1 does not violate the tooling constraints.

Step 4. Machine 1's capacity is fulfilled with this assignment. Terminate the procedure. The objective function value is 136696.

Lower Bound 4 (m-at-a-time Assignment)

Step 1,2&3. First three steps of this heuristic is the same with Lower Bound 2; we can use the operation and tool assignments of LB 2 found in its Step 2.

i \ j	1	2	3
1	60		
2			252
3			
4			79
5			
6		140	
7			
8	533		
9			187
10	118		33

Step 4. As the capacities of the machines are full, no further assignments can be done. The objective function value is 159676.

CHAPTER 5

COMPUTATIONAL RESULTS

In this chapter, we discuss the results of experiments designed to evaluate the performance of our upper and lower bounding procedures. We first introduce the design of our experiments, i.e. the generation of our data. Next, we define the performance measures. In the last section, we report and discuss the results of the computational tests.

5.1 Design of the Experiment

To test the performance of our upper and lower bounds, we generate several random problem instances. The parameters used to generate these instances are listed below:

1. *Problem Size*: Number of operations (n), number of machines (m) and number of tool types (t) are the parameters that define the problem size. The number of operations is set to 50, 75, 100 and 150 in our experiments. The number of machines is set to 3, 5 and 7. The number of tool types is set to 10 for 3 machines and 20 for 5 and 7 machines cases. Table 5.1 shows the number of operations, machines and tool types used in our experiments.

Table 5.1 m, t and n values of our experiments

m	t	n			
3	10	50	75	100	150
5	20	50	75	100	150
7	20	50	75	100	150

2. *Weights:* Chen *et. al* (1995) generated the profit per operation uniformly between integers \$25 and \$150. We used the same interval for the weights of operations. We consider two weight classes: in the first class, the weight of job i is w_i, i.e. independent from the machine it is assigned. In the second class the weights of the operations depend on the machines, i.e. w_{ij} is the weight of operation i on machine j.

3. *The number of tools and tool slots:* Number of tools of each type available on hand (r_k) is set to 3 for each tool type. The number of slots on the tool magazine of machines (s_j) is taken as 20 for all machines. We also use r_k =5 and s_j =15 cases to analyze the effects of r_k and s_j.

4. *The number of tools required by the operations l(i):* We consider two cases of $|l(i)|$. In the first case each operation is assumed to require only one tool, i.e. $|l(i)|=1$. In the second case $|l(i)|$ is a discrete uniform random variable between 2 and 5 when the number of tool types is set to 10; and when the number of tool types is 20, $|l(i)|$ is discrete uniform random variable between 5 and 10. The tools in $l(i)$ are selected randomly.

5. *The inventories of the operations and the capacities of the machines:* The inventories of the operations (W_i) and capacities of the machines (C_j) are generated in terms of minutes by a similar method proposed in Toktay and Uzsoy (1998). Capacities are continuous uniform random variables between 0 and 720. The inventory of each operation i, W_i, is generated as 720*U(0,1)*0.8. To see the effect of the inventory and capacity, we perform test runs with W_i ~720*U(0,1)*0.2 and Cj~U(0,360), as well.

In our main runs, we have 3 different values for n and two different values for m; also we consider two cases for weights and two cases for $|l(i)|$. This results in 48 (4*3*2*2) combinations and 10 problem instances are generated for each combination, which results in a total of 480 problem instances.

The following parameters are used for the Lagrangean relaxation experiments:

1. The initial values for the Lagrangean multipliers are set to 0.
2. The initial value for the scalar λ_k is taken as 1. In the majority of the previous studies λ_k is initialized to 2. However our preliminary runs showed that 1 as

an initial value gives better results than 2. We halve λ_k whenever Lagrangean solution cannot be improved in 30 iterations.

3. As a stopping criterion, we set the number of iterations to 1500. Preliminary runs showed that the objective function value of Lagrangean problem starts to change in around 200^{th} iteration or never improves. Therefore, we terminate the execution of the program if the result of Lagrangean problem does not change in the first 500 iterations.

Lagrangean relaxation and heuristic algorithms are coded in Visual C++ 6.0 version. Original problems, LP relaxation of the problems and the linear programs in Lagrangean relaxation and the heuristic algorithms are solved using CPLEX 8.1 version. All computational experiments are conducted on an Intel Pentium II 550 MHz under the Windows NT operating system.

5.2 Performance Measures

In evaluating the performance of our upper bounding procedures, we use the following performance measures:

1. The solution time: Average and maximum values of solution times in Central Processing Unit (CPU) seconds
2. The percent deviation from the optimum: Average and maximum values of (UB-OPT)/OPT for the cases with known optimal solution
3. Frequency of optimality: The number of times upper bound gives the optimal solution for the cases with known optimum solution

In evaluating the performance of our lower bounding procedures, i.e. heuristics, we use the following performance measures:

1. The CPU time: Average and maximum values of solution times in CPU seconds
2. The percent deviation from the optimum: Average and maximum values of (OPT-LB)/OPT for the cases with known optimal solution
3. The percent deviation from the upper bound: Average and maximum values of (UB-LB)/UB for the cases with unknown optimal solution. For those cases, we use UB as an estimator of the optimal solution.
4. Frequency of optimality: The number of times the best lower bound gives the optimal solution for the cases with known optimal solution.

For the individual performances of the heuristics, we report frequency of the best solution to indicate the number of times each heuristic gives the best solution. We use the best of all heuristics as a lower bound.

5.3 Discussion of the Results

In this section, we discuss the effects of changes in the problem parameters on the performances of our procedures.

Table 5.2 shows the CPU times and deviations of the upper bound from the optimal solution for n=50, 75; m=3, 5, 7; $|l(i)|$=1, $|l(i)|$>1 and machine dependent and independent weights.

Table 5.2 The computational results for upper bound for n=50 and 75

a) $|I(i)|=1$

i. machine dependent weights (w_{ij})					ii. machine independent weights (w_i)				
m=3	CPU Time (sec)		(UB-OPT)/OPT*100		m=3	CPU Time (sec)		(UB-OPT)/OPT*100	
n	Avg	Max	Avg	Max	n	Avg	Max	Avg	Max
50	121.897	135.845	0	0	50	123.648	129.516	0	0
75	125.453	129.796	0	0	75	132.02	154.772	0	0
m=5	CPU Time (sec)		(UB-OPT)/OPT*100		m=5	CPU Time (sec)		(UB-OPT)/OPT*100	
n	Avg	Max	Avg	Max	n	Avg	Max	Avg	Max
50	126.943	130.527	0	0	50	130.474	151.597	0	0
75	145.413	158.878	0	0	75	146.357	148.693	0	0
m=7	CPU Time (sec)		(UB-OPT)/OPT*100		m=7	CPU Time (sec)		(UB-OPT)/OPT*100	
n	Avg	Max	Avg	Max	n	Avg	Max	Avg	Max
50	130.686	138.198	0	0	50	130.887	134.873	0	0
75	131.211	134.893	0.005	0.053	75	141.038	150.506	0	0

b) $|l(i)|>1$

i. machine dependent weights (w_{ij})					ii. machine independent weights (w_i)				
m=3	CPU Time (sec)		(UB-OPT)/OPT*100		m=3	CPU Time (sec)		(UB-OPT)/OPT*100	
n	Avg	Max	Avg	Max	n	Avg	Max	Avg	Max
50	125.879	132.23	0	0	50	121.036	123.337	0	0
75	129.122	146.56	0	0	75	127.557	131.028	0	0
m=5	CPU Time (sec)		(UB-OPT)/OPT*100		m=5	CPU Time (sec)		(UB-OPT)/OPT*100	
n	Avg	Max	Avg	Max	n	Avg	Max	Avg	Max
50	403.552	423.949	0.974	2.244	50	142.824	153.04	0.526	1.359
75	157.648	181.831	0.673	2.189	75	155.578	158.738	0.447	1.107
m=7	CPU Time (sec)		(UB-OPT)/OPT*100		m=7	CPU Time (sec)		(UB-OPT)/OPT*100	
n	Avg	Max	Avg	Max	n	Avg	Max	Avg	Max
50	155.882	430.328	5.166	13.213	50	158.496	189.364	5.074	7.689
75	152.101	433.773	1.683	3.937	75	158.085	178.056	1.582	2.359

5.3.1 Effects of Number of Operations and Machines

As can be seen from Table 5.2, the CPU time is not significantly affected by the changes in n. As mentioned before, we terminate the program if no improvement is observed in the Lagrangean solution in 500 iterations. Otherwise, 1500 iterations are performed. The CPU time deviations are mainly due to the differences in iteration limits used in Lagrangean solutions. Also the deviations from the optimal solutions are not sensitive to n, i.e. increasing the number of operations does not increase the complexity of the problem. When n is large, the number of alternatives for the selected operations becomes high.

When $n \leq 75$ and $m \leq 5$, CPLEX can solve the original problem in small CPU times, however the solution times increase considerably with an increase in the problem size. For example CPLEX can solve the problem instances with n=50, m=3, $|l(i)|>1$ and arbitrary weights, in less than 1 second, on average. When m is set to 7 while keeping the other parameters fixed, the average CPU time increases to 1001.968 seconds. When n becomes 100, the Integer Programming solvers failed to solve the majority of the problem instances. As the upper bounds' performances are quite satisfactory, they can be good estimates of the optimal solution. Therefore for the cases the optimal solutions are not known, we report LB performances relative to UBs, in place of optimal solutions.

Tables 5.3 and 5.4 show the computational results for the lower bounds using the same parameters. Table 5.3 reports the maximum and average CPU times and the maximum and average deviation of the lower bounds from the optimal solution as a percentage of optimal solution when n=50 and 75. And Table 5.4 gives the CPU times and deviation from the upper bound when n=100 and 150.

As can be seen from the tables, the CPU times are similar when n=50 and 75. However, when n becomes 100 and 150, the CPU times increase. As the heuristics are based on many iterative solutions, increasing n increases the number of alternatives and thereby increasing its solution time. Similar to the results of the upper bound, gaps between lower bound and optimum solution are not sensitive to n. However, the deviation between lower and upper bounds increase as n increases from 100 to 150.

Table 5.3 The computational results for lower bound when n=50 and 75

a) $|l(i)|=1$

machine dependent weights (w_{ij})					machine independent weights (w_i)				
m=3	CPU Time (sec)		(OPT-LB)/OPT*100		m=3	CPU Time (sec)		(OPT-LB)/OPT*100	
n	Avg	Max	Avg	Max	n	Avg	Max	Avg	Max
50	0.915	2.463	0	0	50	1.073	2.483	0	0
75	0.788	1.712	0	0	75	0.908	2.353	0	0
m=5	CPU Time (sec)		(OPT-LB)/OPT*100		m=5	CPU Time (sec)		(OPT-LB)/OPT*100	
n	Avg	Max	Avg	Max	n	Avg	Max	Avg	Max
50	0.688	1.422	0	0	50	0.792	1.752	0	0
75	0.732	2.393	0	0	75	0.753	1.502	0.007	0.066
m=7	CPU Time (sec)		(OPT-LB)/OPT*100		m=7	CPU Time (sec)		(OPT-LB)/OPT*100	
n	Avg	Max	Avg	Max	n	Avg	Max	Avg	Max
50	0.802	1.972	0	0	50	0.718	1.872	0.076	0.442
75	0.671	1.522	0	0	75	0.773	1.592	0.08	0.522

b) $|l(i)|>1$

machine dependent weights (w_{ij})					machine independent weights (w_i)				
m=3	CPU Time (sec)		(OPT-LB)/OPT*100		m=3	CPU Time (sec)		(OPT-LB)/OPT*100	
n	Avg	Max	Avg	Max	n	Avg	Max	Avg	Max
50	0.847	1.702	0	0	50	0.941	2.223	0	0
75	0.793	1.532	0	0	75	0.747	1.742	0	0
m=5	CPU Time (sec)		(OPT-LB)/OPT*100		m=5	CPU Time (sec)		(OPT-LB)/OPT*100	
n	Avg	Max	Avg	Max	n	Avg	Max	Avg	Max
50	2.238	17.445	2.02	8.514	50	2.742	24.755	2.947	9.705
75	1.754	17.024	3.178	7.435	75	3.084	22.492	3.947	9.476
m=7	CPU Time (sec)		(OPT-LB)/OPT*100		m=7	CPU Time (sec)		(OPT-LB)/OPT*100	
n	Avg	Max	Avg	Max	n	Avg	Max	Avg	Max
50	7.823	45.745	11.782	22.628	50	8.013	44.984	9.452	17.819
75	7.954	50.142	6.004	17.909	75	12.14	67.937	4.721	6.216

Table 5.4 The computational results for lower bound when n=100 and 150

a) $|l(i)|=1$

machine dependent weights (w_{ij})					machine independent weights (w_i)				
m=3	CPU Time (sec)		(UB-LB)/UB*100		m=3	CPU Time (sec)		(UB-LB)/OPT*100	
n	Avg	*Max*	Avg	Max	n	Avg	Max	Avg	Max
100	0.674	1.732	0	0	100	0.706	1.422	0	0
150	0.573	1.542	0	0	150	0.619	1.351	0	0
m=5	CPU Time (sec)		(UB-LB)/UB*100		m=5	CPU Time (sec)		(UB-LB)/OPT*100	
n	Avg	Max	Avg	Max	n	Avg	Max	Avg	Max
100	0.849	2.363	0	0	100	0.664	1.712	0	0
150	0.59	1.141	0	0	150	0.963	4.005	0	0
m=7	CPU Time (sec)		(UB-LB)/UB*100		m=7	CPU Time (sec)		(UB-LB)/OPT*100	
n	Avg	Max	Avg	Max	n	Avg	Max	Avg	Max
100	0.652	2.523	0	0	100	0.716	1.592	0.044	0.19
150	0.628	1.412	0	0	150	0.651	1.522	0.008	0.079

b) $|l(i)|>1$

machine dependent weights (w_{ij})					machine independent weights (w_i)				
m=3	CPU Time (sec)		(UB-LB)/UB*100		m=3	CPU Time (sec)		(UB-LB)/OPT*100	
n	Avg	Max	Avg	Max	n	Avg	Max	Avg	Max
100	0.641	1.171	0	0	100	0.627	1.371	0	0
150	0.581	1.301	0	0	150	0.709	3.505	0	0
m=5	CPU Time (sec)		(UB-LB)/UB*100		m=5	CPU Time (sec)		(UB-LB)/OPT*100	
n	Avg	Max	Avg	Max	n	Avg	Max	Avg	Max
100	2.763	32.656	3.634	15.683	100	1.349	8.261	2.148	7.275
150	15.302	86.434	17.667	27.648	150	2.721	30.624	3.418	8.814
m=7	CPU Time (sec)		(UB-LB)/UB*100		m=7	CPU Time (sec)		(UB-LB)/OPT*100	
n	Avg	Max	Avg	Max	n	Avg	Max	Avg	Max
100	13.607	69.279	12.788	22.171	100	12.85	82.458	13.325	21.352
150	36.32	192.253	25.928	37.626	150	15.508	81.026	10.205	16.176

It is easily seen from Tables 5.2, 5.3 and 5.4 that the solution times increase as m increases. Since the number of binary variables is an increasing function of m, an increase in the solution times with an increase in m is an expected result.

When the number of machines is 3, both upper and lower bounding procedures can easily find the optimal solutions. The solution times are very short particularly for the heuristics. As m increases, the CPU times and the deviations from the optimum solutions increase for both bounds. This difference is clearer for the lower bounds. For the upper bounds, the average deviation from the optimum solution is less than 5%, in most cases.

As we discussed, we generate two sets for the weights of the operations. We aim to observe the performances of our upper and lower bounds when the weights depend on the machine (w_{ij}), i.e. arbitrary and when $w_{ij}=w_i$ for all j, i.e. the weights are machine independent. Tables 5.2, 5.3 and 5.4 show the results considering two different cases for weights.

We can conclude from the tables that the performances of upper bound for arbitrary w_{ij} and machine independent weights w_i are very close, but the maximum deviation is higher in arbitrary w_{ij} case. The average deviation also increases but not as much. Note that the average deviations are satisfactory for both cases. Also upper bound returns the optimal value in majority of the problem instances. When the weights are arbitrary, the CPU times are greater. When the weights are identical for the machines, all machines yield the same cost, thereby reducing the number of alternatives, which in turn reduces the CPU times.

Note that the performance of the lower bounds do not change significantly with the changes in weights. The average deviations from the optimal solution are less than 10% in most cases. The CPU times seem to be greater for machine independent weights w_i.

5.3.2 Effect of the Number of Tools Required for Processing an Operation

We also investigate the effect of the number of tools required for processing an operation, for $|l(i)|=1$ and $|l(i)|>1$ cases by analyzing Tables 5.2, 5.3 and 5.4.

It can be easily seen that the upper bounding procedure performs excellent and finds the optimal value in almost all problem instances (179 out of 180) when

$|l(i)|$=1. The deviations from the optimum solution increase for $|l(i)|$>1 case, but the average gaps are still at most around 5%. The gap between upper and lower bounds increase when $|l(i)|$>1. The CPU times of two cases are almost identical.

The tables also show that the lower bound performs significantly better when $|l(i)|$=1. In most of the problems the optimal solution is found very quickly. When multiple tools are required for processing the operations, the CPU time and deviation from optimum increases, however the average deviation is still satisfactory. For n=100 and 150, the CPU times and difference between upper and lower bounds increase as $|l(i)|$ increases. When $|l(i)|$=1 the number of tool slots in the tool magazine of the machines is not a hard constraint as in $|l(i)|$>1 case, hence the problem becomes easier.

5.3.3 Effect of Number of Tool Types

To analyze the effect of t on the solution time and quality, we generate two cases: t=10 and t=20. Table 5.5 shows the upper bound performances for these cases where n=75, m=5, arbitrary weights; $|l(i)| > 1$.

Table 5.5 Computational results for different values of t for upper bound

	CPU Time (sec)		(UB-OPT)/OPT*100		
t	Avg	Max	Avg	Max	# opt
10	149.855	375.309	0.268	1.183	1
20	157.648	181.831	0.673	2.189	1

The CPU times again are not affected from t values. The deviation from the optimum value slightly increases when the number of tool types increases. However the deviations are still 2%, at most. In both cases, the upper bound returns the optimal solution in one out of 10 problem instances.

Table 5.6 shows the results of the lower bounds for the same problem combinations.

Table 5.6 Computational results for different values of t for lower bounds

t	CPU Time (sec)		(OPT-LB)/OPT*100		# opt
	Avg	Max	Avg	Max	
10	0.886	6.008	0.482	1.885	4
20	1.754	17.024	3.178	7.435	0

The results show that the procedure performs better when t=10. As the number of tool types increases from 10 to 20, the average and maximum CPU times double. The average and maximum percentage gaps between the lower bound and optimum increase as t increases. Also lower bounds return the optimum solution for 4 instances when t=10, however when t=20, the optimal solution is never hit by the heuristics. The number of binary variables in the problem increases as a function of the number of tool types; therefore the difficulty of the problem increases with an increase in t.

5.3.4 Effect of the Number of Tools Available in the System

To investigate the effect of number of tools available in the system, i.e. r_k, we tried two different values when n=75; m=5; t=20; for arbitrary weights and $|l(i)| > 1$ case. Table 5.7 shows the performance of the upper bound for different values of r_k.

Table 5.7 The computational results for upper bound for different values of r_k

r_k	CPU Time (sec)		(UB-OPT)/OPT*100		# opt
	Avg	Max	Avg	Max	
3	157.648	181.831	0.673	2.189	1
5	136.978	149.174	0	0	10

Note that, increasing the number of available tools makes the problem easier since the number of tools is no longer acts as a constraint. When r_k is 3, the algorithm gives very small deviations, around 2% at most. But when r_k is set to 5, the upper

bound finds the optimal solution in all problem instances. The CPU times are slightly smaller for r_k=5 case.

We also investigate the performance of the lower bounds for different values of r_k and report the results in Table 5.8.

Table 5.8 The computational results for different values of r_k for lower bounds

r_k	CPU Time (sec)		(OPT-LB)/OPT*100		# opt
	Avg	Max	Avg	Max	
3	1.754	17.024	3.178	7.435	0
5	0.492	1.181	0	0	10

The results of the lower bounding procedures show significant changes as r_k changes. With r_k=5, heuristics give excellent results, the optimum solution is found in all problem instances within 1 second. Also r_k=3 case gives very good results, with average CPU time of 1.75 sec and the average deviation from optimum around 3%. Since the problem becomes easier when r_k=5, the results are consistent with our expectations.

5.3.5 Effect of Tool Magazine Capacities

To observe the effect of tool magazine capacities on the performances of the upper and lower bounds, we perform some experiments for s_j=15 and 20. The results for n=75; m=5; arbitrary weights; and $|l(i)| > 1$ cases are given in Table 5.9.

Table 5.9 The computational results for different cases of s_j for upper bound

s_j	CPU Time (sec)		(UB-OPT)/OPT*100		# opt
	Avg	Max	Avg	Max	
15	157.539	165.147	10.569	18.603	0
20	157.648	181.831	0.673	2.189	1

As can be seen from the table, the problem performs better when s_j=20 since the number of tool slots is not constraining the problem. The CPU times do not change with s_j. However the upper bound and optimum solutions become closer when the machines have more tool slots.

Table 5.10 gives the performance measures of the lower bounds for the same problem combination.

Table 5.10 The computational results for different cases of s_j for lower bound

	CPU Time (sec)		(OPT-LB)/OPT*100		
s_j	Avg	Max	Avg	Max	# opt
15	9.004	59.245	2.257	8.781	4
20	1.754	17.024	3.178	7.435	0

As expected lower bound behaves different than the upper bound. The average and maximum CPU times are greater when s_j=15. The lower bound deviations are nearly the same for s_j=15 and 20. Note that, when s_j=15, the lower bound returns the optimal solution in 4 out of 10 problem instances.

5.3.6 Effect of Inventory Amounts

To check the performance for different inventory amounts, we compare two cases: W_i=0.2*U(0.720)*720 and W_i=0.8*U(0,720)*720, i.e. low and high inventory levels respectively. Table 5.11 gives the results for two cases when n=75, m=5, t=20, arbitrary weights; $|l(i)| > 1$.

Table 5.11 The computational results for upper bound for different values of inventories

	CPU Time (sec)		(UB-OPT)/OPT*100		
W_i	Avg	Max	Avg	Max	# opt
0.2*U(0.720)*720	138.512	146.49	19.273	30.095	0
0.8*U(0,720)*720	157.648	181.831	0.673	2.189	1

We can conclude from the table that the upper bounding procedure performs better when W_i~0.8*U(0,720)*720. The CPU times do not differ significantly.

The performance measures of the lower bounds are given in Table 5.12.

Table 5.12 The computational results for different values of inventories for lower bound

W_i	CPU Time (sec)		(OPT-LB)/OPT*100		# opt
	Avg	Max	Avg	Max	
0.2*U(0.720)*720	9.235	57.873	5.856	19.447	0
0.8*U(0,720)*720	1.754	17.024	3.178	7.435	0

For the heuristics, we observe the similar results. In terms of both the deviations and CPU times, the heuristic procedures perform better when W_i~0.8*U(0,720)*720. Also CPU times are shorter in this case. Low level of inventory is more constraining, that makes the problem more difficult to solve.

5.3.7 Effect of Machine Capacities

To see the effect of different capacities on the performance of upper and lower bounding procedures, we compare two cases: C_j~U(0,360) and C_j~U(0,720) for n=75, m=5, t=20, arbitrary weights; $|l(i)| > 1$. Tables 5.13 and 5.14 show the results for upper and lower bounds, respectively.

Table 5.13 The computational results for upper bound for different values of machine capacities

C_j	CPU Time (sec)		(UB-OPT)/OPT*100		# opt
	Avg	Max	Avg	Max	
U(0,360)	137.303	141.273	3.948	11.920	1
U(0,720)	157.648	181.831	0.673	2.189	1

Table 5.14 The computational results for different values of machine capacities for lower bound

C_j	CPU Time (sec)		(OPT-LB)/OPT*100		# opt
	Avg	Max	Avg	Max	
U(0,360)	6.504	29.412	6.945	19.503	0
U(0,720)	1.754	17.024	3.178	7.435	0

The above tables show that both upper and lower bounding procedures give better results when C_j~U(0,720). This result is in line with our expectations as high machine capacity case is less constraining.

5.3.8 Individual Performances of the Lower Bounding Procedures

Finally we investigate the individual performances of the lower bounding procedures. The performances are reported on Table 5.15 for arbitrary w_{ij}, where the average CPU times and average percent deviations from the optimum solution ((OPT-LB)/OPT*100) for n=50, 75 and the average deviations between lower and upper bounds ((UB-LB)/UB*100) for n=100,150 are given. We run LB5, i.e. Lagrangean heuristic, while finding an upper bound with Lagrangean relaxation procedure, hence no additional time is spent; and therefore the CPU times of LB5 are not given in table 5.15.

As can be seen from the table, all heuristics perform very well for m=3. LB2 and LB4 find the optimal solution for all problem instances, and LB1 for majority of the instances. LB3 and LB5 also find near optimal results. When m increases to 5, LB3 seems to dominate other heuristics in terms of the average deviation. For m=7 case, the performances of LB2, LB3 and LB4 are very close and better than those of LB1 and LB5. The CPU time of LB3 is longer than other heuristics however it can find better solutions than others in many cases.

Table 5.15 also shows that LB1, LB2 and LB4 find the optimal solutions very quickly, when $|l(i)|$=1 for most of the problems, for all values of n and m. LB3 and LB5 yield very small deviations, however relatively larger than the others. LB3

performs better as the number of operations and machines increase. The solution times are not much affected from the increases in n.

Table 5.15 Performance measures of lower bounds

a) $|l(i)|=1$

	n	**LB 1**		**LB 2**		**LB 3**		**LB 4**		**LB 5**
		Avg CPU	Avg Dev	Avg CPU	Avg Dev	Avg CPU	Avg Dev	Avg CPU	Avg Dev	Avg Dev
m=3	50	0.02	0.065	1.302	0	0.879	4.719	1.46	0	0.747
	75	0.017	0.05	1.073	0	0.839	2.737	1.223	0	0.303
	100	0.019	0	0.939	0	0.682	2.234	1.057	0	0.351
	150	0.054	0.002	0.896	0	0.398	2.642	0.946	0	0.511
m=5	50	0.024	0.023	0.903	0	0.746	2.462	1.081	0	0.323
	75	0.057	0	1.041	0	0.618	2.729	1.212	0	0.345
	100	0.029	0	1.348	0	0.89	2.062	1.127	0	0.308
	150	0.054	0.01	0.856	0	0.566	1.295	0.885	0	0.58
m=7	50	0.042	0.064	1.256	0	0.798	3.038	1.111	0	0.264
	75	0.058	0.065	0.977	0.001	0.713	2.73	0.938	0.001	0.661
	100	0.031	0.01	0.941	0	0.551	1.864	1.085	0	0.522
	150	0.081	0.01	0.852	0	0.647	1.733	0.934	0	0.402

b) $|l(i)|>1$

	n	LB 1		LB 2		LB 3		LB 4		LB 5
		Avg CPU	Avg Dev	Avg CPU	Avg Dev	Avg CPU	Avg Dev	Avg CPU	Avg Dev	Avg Dev
m=3	50	0.02	0.049	1.092	0	0.82	3.037	1.455	0	2.549
	75	0.014	0	1.146	0	0.9	1.903	1.113	0	2.123
	100	0.017	0.011	0.944	0	0.705	2.15	0.899	0	2.044
	150	0.044	0	0.84	0	0.586	1.949	0.856	0	1.233
m=5	50	0.025	10.781	0.837	8.322	6.808	3.661	1.742	10.754	21.374
	75	0.038	14.684	1.022	7.313	4.589	4.685	1.368	5.259	25.426
	100	0.052	17.309	1.586	5.16	7.889	5.677	1.525	5.178	25.28
	150	0.056	35.039	0.855	22.927	59.409	19.912	0.888	25.555	38.949
m=7	50	0.034	30.791	1.25	19.469	28.645	20.127	1.365	16.696	27.993
	75	0.041	29.957	0.886	9.413	29.921	10.591	0.97	8.889	26.035
	100	0.035	43.412	0.87	16.967	52.365	18.482	1.159	18.182	32.243
	150	0.07	57.217	0.88	33.968	136.261	27.153	0.994	35.02	40.773

Table 5.16 gives the number of problems lower bounds give the best result among all five for arbitrary w_{ij} case. The numbers in parentheses show the number of times the lower bounds find the optimum solution.

Table 5.16 The number of problems LBs find the best and optimum solutions

a) $|l(i)|=1$

	n	LB1	LB2	LB3	LB4	LB5
m=3	50	8(8)*	10(10)	0	10(10)	0
	75	8(8)	10(10)	0	10(10)	0
	100	10(10)	10(10)	2(2)	10(10)	0
	150	9(9)	10(10)	2(2)	10(10)	0
m=5	50	8(8)	10(10)	0	10(10)	0
	75	10(10)	10(10)	0	10(10)	0
	100	10(10)	10(10)	1(1)	10(10)	0
	150	9(9)	10(10)	1(1)	10(10)	0
m=7	50	5(5)	10(10)	0	10(10)	0
	75	6(6)	9(9)	0	9(9)	0
	100	9(9)	10(10)	0	10(10)	0
	150	8(8)	10(10)	0	10(10)	0

* The numbers in the parentheses give the number of optimal solutions

b) $|l(i)|>1$

	n	LB1	LB2	LB3	LB4	LB5
m=3	50	8(8)	10(10)	1(1)	10(10)	0
	75	10(10)	10(10)	1(1)	10(10)	0
	100	9(9)	10(10)	3(3)	10(10)	0
	150	10(10)	10(10)	1(1)	10(10)	0
m=5	50	3(1)	2(1)	4	5(1)	0
	75	0	2	6	4	0
	100	2	5	3	2	0
	150	1	4	6	2	0
m=7	50	1	2	3	4	0
	75	1	4	2	3	0
	100	0	6	3	1	0
	150	0	5	5	1	0

As can be observed from the tables, LB1, LB2 and LB4 perform better for small m and find many optimal solutions especially when $|l(i)|=1$. LB3 gives better results as m increases and when $|l(i)|>1$. LB5 does not give the best solution in any of the problems.

Hence we can conclude that in order to get a satisfactory approximate solution to our NP-hard problem, all procedures, except LB5, should be used.

CHAPTER 6

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 machines so as to maximize the total weight. We model the problem as a mixed integer linear program and prove that this problem is NP-hard in the strong sense.

We propose upper and lower bounding procedures for solvingthis NP-hard problem. Lagrangean relaxation approach with subgradient optimization technique is used to obtain strong upper bounds on the optimal objective function value. Several heuristic procedures are developed that give near-optimal solutions in small computational times.

The results of our computational experience have revealed that our upper and lower bounding procedures give satisfactorily good solutions in reasonable solution times. Lagrangean relaxation and heuristic procedures return the optimal solution for many problems and for some others they produce solutions that are quite close to optimal ones.

We observe from our experiment that the number of machines, m, is the most dominant factor that affects the difficulty of the problem. As m increases, the complexity of the problem increases significantly. Also the change in number of tools required to process an operation, i.e. $|l(i)|$, adds to the difficulty of the problem. When few machines are in the system or when only one tool is required by each operation, both the upper and lower bounding algorithms find the optimal solutions very quickly. Decreasing the number of tools available, the machine capacities, the inventories of operations, the number of tool slots in the tool magazines of the machines, the number of tool types, all increase the complexity of the problem.

However, the average deviations from the optimal solution are quite small even for most difficult problem combination.

Lower bounding procedures LB1, LB2 and LB3 produce near optimal solutions for easy problems. Also LB4 gives very good results for small problems but it dominates others when more difficult problems are considered. LB5 gives near optimal solutions for small problems, but for all problem combinations, it is dominated by other procedures. Hence, we can conclude that to arrive a good approximate solution all lower bounds except LB5 should be considered together.

To the best of our knowledge, there is no other study in the literature on capacity allocation problem in flexible manufacturing systems considering operation assignment and tool allocation decisions simultaneously. Our study can be extended to a number of research areas, some of which are mentioned below:

- In addition to the allocation problem, the sequencing of the operations on the machines can be considered. The solution approaches may consider simultaneous or sequential solutions of allocation and sequencing problems.
- We assume machine dependent weights, however machine independent capacity usages for the operations. Future research may consider arbitrary capacity usages as well.
- We assume that the operations can be split between the machines. Assuming a single machine assignment for each operation can be another consideration. In such a case implicit enumeration techniques, like branch and bound procedures, can be of great help.
- We assume that no changeovers can be done after the tool magazines are loaded. An interesting extension may be to allow tool changeovers and subsequently to detect the sequence of changeovers.
- We find quite satisfactory upper bounds. The incorporation of those bounds to an enumeration scheme can be an interesting research extension. In doing so, the quality of the bounds and the effort spent to find them should be well established.
- We approach the problem by relaxing the constraint that links operation and tool assignments. The relaxations of other constraints, in particular the ones that are relaxed with tooling, can open a new research avenue.

REFERENCES

Akçalı, E., Üngör, A., Uzsoy, R., (2003), "Tool- and Setup-Constrained Short-Term Capacity Allocation Problem", *Proceedings of International Symposium on Computer and Information Sciences*, Antalya, Turkey, pp. 163-170.

Beasley, J. E., (1995), "Lagrangean Relaxation", In: Reeves, C. R., (1995), Modern Heuristic Techniques for Combinatorial Problems, McGraw-Hill, pp. 243-303.

Berrada, M., Stecke, K. E., (1986), "A Branch and Bound Approach for Machine Load Balancing in Flexible Manufacturing Systems", *Management Science*, Vol. 32, pp. 1316-1335.

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*, Vol. 33, pp. 2671-2683.

Çatay, B., Erengüç, Ş. S., Vakharia, A.J., (2002), "Capacity Allocation with Machine Duplication in Semiconductor Manufacturing", submitted to *Naval Research Logistics*

D'Alfonso, T. H., Ventura, J. A., (1995), "Assignment of Tools to Machines in a Flexible Manufacturing System", *European Journal of Operational Research*, Vol. 81, pp. 115-133.

Fisher, M. L., (1981), "The Lagrangean Relaxation Method for Solving Integer Programming Problems", *Management Science*, Vol. 27, pp. 1-18.

Fisher, M. L., (1985), "An Applications Oriented Guide to Lagrangean Relaxation", *Interfaces*, Vol. 15, pp. 10-21.

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

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*, Vol. 67, pp. 387-404.

Nemhauser, G. L., Wolsey, L. A., (1988), Integer and Combinatorial Optimization, John Wiley & Sons Inc.

Papadimitriou, C. H., Steiglitz, K., (1982), Combinatorial Optimization Algorithms and Complexity, Prentice-Hall Inc.

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*, Vol. 28, pp. 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*, Vol. 25, pp. 1081-1094.

Shanker, K., Tzen, Y. –J. J., (1985), "A Loading and Dispatching Problem in a Random Flexible Manufacturing System", *International Journal of Production Research*, Vol. 23, pp. 575-595.

Shanker, K., Srinivasulu, A., (1989), "Some Solution Methodologies for Loading Problems in a Flexible Manufacturing System", *International Journal of Production Research*, Vol. 27, pp. 1019-1034.

Sodhi, M. S., Askin, R. G., Sen, S., (1994), "Multiperiod Tool and Production Assignment in Flexible Manufacturing Systems", *International Journal of Production Research*, Vol. 32, pp. 1281-1294.

Toktay, L. B., Uzsoy, R., (1998), "A Capacity Allocation Problem with Integer Side Constraints", *European Journal of Operational Research*, Vol. 109, pp. 170-182.

Ventura, J. A., Chen, F. F., Leonard, M.S., (1988), "Loading Tools to Machines in Flexible Manufacturing Systems", *Computers and Industrial Engineering*, Vol. 15, pp. 223-230.

Wolsey, L. A., (1998), Integer Programming, John Wiley & Sons Inc.