BI-OBJECTIVE BIN PACKING PROBLEMS

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

BY

IŞIL ILICAK

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF INDUSTRIAL ENGINEERING

DECEMBER 2003

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 the 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 AZİZOĞLU
Co-Supervisor

Asst. Prof. Dr. Esra KARASAKAL
Supervisor

Examining Committee Members:

Prof. Dr. Meral AZİZOĞLU _____

Asst. Prof. Dr. Esra KARASAKAL _____

Assoc. Prof. Dr. Nur Evin ÖZDEMİREL _____

Assoc. Prof. Dr. Yıldırım SALDIRANER _____

Dr. Özge UNCU _____

ABSTRACT

BI-OBJECTIVE BIN PACKING PROBLEMS

ILICAK, Işıl

M. Sc., Department of Industrial Engineering

Supervisor: Asst. Prof. Dr. Esra KARASAKAL

Co-Supervisor: Prof. Dr. Meral AZİZOĞLU

December 2003, 81 pages

In this study, we consider two bi-objective bin packing problems that assign a number of weighted items to bins having identical capacities. Firstly, we aim to minimize total deviation over bin capacity and minimize number of bins. We show that these two objectives are conflicting. Secondly, we study the problem of minimizing maximum overdeviation and minimizing the number of bins. We show the similarities of these two problems to parallel machine scheduling problems and benefit from the results while developing our solution approaches. For both problems, we propose exact procedures that generate efficient solutions relative to two objectives. To increase the efficiency of the solutions, we propose some lower and upper bounding procedures. The results of our experiments show that total overdeviation problem is easier to

solve compared to maximum overdeviation problem and the bin capacity, the weight of items and the number of items are important factors that effect the solution time and quality. Our procedures can solve the problems with up to 100 items in reasonable solution times.

Keywords: Bin Packing, Multiobjective Optimization, Efficient Solutions

ÖZ

İKİ AMAÇLI KUTU PAKETLEME PROBLEMLERİ

ILICAK, Işıl

Yüksek Lisans, Endüstri Mühendisliği Bölümü

Tez Yöneticisi: Yrd. Doç. Dr. Esra KARASAKAL

Ortak Tez Yöneticisi: Prof. Dr. Meral AZİZOĞLU

Aralık 2003, 81 sayfa

Bu çalışmada, ağırlıkları belirli cisimleri eşit kapasitelerdeki kutulara yerleştiren iki amaçlı kutu paketleme problemlerini ele aldık. İlk olarak, kutu kapasitesinden toplam sapmayı ve kutu sayısını en azlamayı amaçladık. Problemin amaç fonksiyonlarının birbiriyle çeliştiğini gösterdik. İkinci olarak, maksimum sapmayı ve kutu sayısını en azlayan problem üzerinde çalıştık. Problemlerin, paralel makinalı çizelgeleme problemine benzerliklerini gösterdik ve çözüm yöntemleri geliştirirken bu benzerliklerden yararlandık. İki amaca göre etkin çözümler elde eden kesin yöntemler geliştirdik. Çözümlerin verimliliğini arttırmak için bazı alt ve üst sınır yöntemleri önerdik. Deneylerimizin sonucu, toplam sapma probleminin, maksimum sapma problemine göre daha kolay

olduğunu ve kutu kapasitesinin, cisimlerin ağırlıklarının ve cisim sayısının çözüm süresi ve kalitesi üzerinde önemli etkileri olduğunu göstermektedir. Yaklaşımlarımız, cisim sayısı 100'e kadar olan problemleri makul çözüm süreleri içinde çözmektedir.

Anahtar Kelimeler: Kutu Paketleme, Çok Amaçlı Optimizasyon, Etkin Çözümler

ACKNOWLEDGEMENTS

I would like to express my deepest gratitudes to my thesis supervisor Asst. Prof. Dr. Esra Karasakal and my co-supervisor Prof. Dr. Meral Azizoğlu, for their informative and constructive comments, guidance, support and encouragement throughout this study. It has been a great pleasure for me to work under their supervision.

I thank my dear friend and colleague Bülvin Karlıkaya for her support and encouragement.

I offer sincere thanks to my parents, Hatice and Yener Ilıcak, my sister İdil Ilıcak and my aunt Hale Üstünbal for their encouragement.

Finally, I wish to thank to my husband Sertan for his patience and encouragement.

TABLE OF CONTENTS

LIST OF TABLES

TABLE

LIST OF FIGURES

FIGURE

CHAPTER 1

INTRODUCTION

Bin packing problem has been widely researched in operations research literature. In the classical bin packing problem, a set of items of different sizes and weights has to be packed into bins of limited capacities. A packing is feasible if the total sizes of the items in all bins do not exceed the bin capacity. As the problem has a wide range of application areas related to space and time minimization its study is of theoretical interest.

Bin packing problem has been categorized in many different ways. One categorization is due to the dimension of packs. Most widely studied version is one dimensional bin packing problem that has its place in manufacturing and production applications. There are many exact procedures, heuristics, on-line and off-line algorithms developed for one dimensional bin packing problem. Also, dual versions of this problem, i.e., problems with the objective of maximizing the number of bins used or maximizing the number of items packed to the bins are investigated.

New and improved versions of bin packing problems have arisen with improving industry demands. Variable sized bin packing problems with variable

bin capacities, problems with item weights greater than the bin capacity, are some notable examples. Due to its close relation to practical life, bin packing problems are continuing to gain importance in operational research area.

Bin packing arises in a variety of packaging and manufacturing problems. A well known manufacturing example is cutting stocks so as to minimize waste, therefore to minimize cost. In waste minimization, we layout the parts so as to use as few fixed-size sheets as possible. Identifying which part goes to which sheet in which location is a variation of bin packing problem so called the cutting stock problem. Another version of bin packing problem arises, namely how best to fit the boxes into trucks to minimize the number of trucks needed to ship all.

Classical bin packing problem finds its application in scheduling problems with parallel machines. If we assume parallel machines environment, a natural question may be how to assign items to the machines without exceeding the specified makespan, i.e. specified maximum completion time. With these information, we can make the following analogy: The machines can be represented by the bins of limited capacity. The capacity of the machines refers to the makespan, whereas the processing times are analogous to item weights.

If required completion time of all machines are the same, then we use bins with equal capacity. Having identical machine completion times represent the case where the items have a common due date.

In this study, we allow deviations over bin capacity. Hence, even if the total weight of items in a bin exceeds the bin capacity, the solution is feasible.

Our problem has two conflicting objectives. First objective is to minimize the number of bins used and second objective is to minimize total or maximum deviation over capacity. Hence, according to the second objective we study two problems. To the best of our knowledge, there is no such work in the literature.

Our problems are analogous to the production environments as follows: There are a number of identical parallel machines (bins) and the processing times (weights) of the jobs (items) are known. Due date of all jobs are the same (equal bin capacity). The aim is to minimize the number of machines used and minimize total lateness of the last jobs (total overdeviation) or makespan (maximum overdeviation).

In our study, we propose some exact procedures to generate all efficient solutions of our biobjective problems. Using the results of scheduling theory, we generate constructive and improvement upper bounding heuristics and use some well known lower bounds.

We also aim to see the effects of capacity, weight and number of items on the solution speed and quality. We investigate the differences and similarities between our biobjective problems.

The rest of the thesis is organized as follows:

Chapter 2 reports the literature survey related with bin packing problems. The related work is classified according to the types of bin packing problems.

In Chapter 3, we define our problems and give their mathematical formulations. The basic assumptions, notation used throughout the study are also given in this chapter.

In Chapter 4, we present our solution procedures to generate all efficient solutions. We propose some methods to find upper and lower bounds and some useful theorems.

The computational experiment associated with our procedures are reported in Chapter 5. In this chapter, we discuss the effects of capacity, weight and number of items on the solution speed and quality. We also make comparative discussions of the total deviation and maximum deviation problems.

In Chapter 6, we give our conclusions and point out the directions for future research.

CHAPTER 2

LITERATURE REVIEW

Bin packing problem is a widely studied operations research problem. Many different packing problems are defined, depending on the size and shape of the items, as well as on the form and the capacity of the bins. Similar problems occur in minimizing material wastage while cutting sheets into smaller parts, and in the scheduling of identical processors so as to minimize maximum completion time. Bin packing problem is simple to state, but hard to solve. It is shown to be NP-hard (Garey and Johnson, 1979).

Bin packing problem has a variety of different types. We classify the problem into three main categories as classical bin packing problem, dual bin packing problem (Type I) and dual bin packing problem (Type II). Each of these cases are reviewed below. We also review some other variations like on-line bin packing problems.

Classical Bin Packing Problem: The basic bin packing problem is the classical bin-packing problem (BPP). Given the weight of the items, and the capacity of bins, the objective is to minimize the number of bins used, without exceeding the capacity.

Dual Bin Packing Problem (Type I): Given the weight of the items and the capacity of the bins, the objective is to maximize the number of bins used, where the total weight of items in any bin is allowed to be greater than or equal to the capacity of the bins. The minimum usage of the bins should be equal to the capacity of the bins. This problem is also called bin covering problem.

Dual Bin Packing Problem (Type II): In this version the number of bins is fixed, and the objective is to maximize number of items packed without exceeding the capacity of the bins.

The bin packing problems are also categorized into two: single sized problem and variable sized problem. In single sized problems, the bins have equal capacities where in variable sized version the bins have different capacities.

BPP can also be divided into two classes: off-line and on-line. In off-line version, all items are initially available, i.e. the system is static, while in the on-line version the items arrive dynamically. Off-line algorithms first build an order of the items while on-line algorithms consider the items in the given order and pack each item into a bin according to a particular selection strategy. In this study we focus on off-line algorithms and assume that the bins have identical capacities unless stated otherwise.

Off-line algorithms are divided into two classes: approximate and exact algorithms. Majority of the studies in the literature on BPP consider approximate algorithms and measure their performances. There are a few exact algorithms in

the literature, as well. They use the following measures to evaluate the performances of the approximate algorithms:

Worst Case Ratio (r): The worst case performance ratio of algorithm A is defined as the largest real number $p(A)$ such that;

$$A(I) / OPT(I) \leq p(A) \qquad \text{for all instances I.}$$

where;

A(I) = value obtained by solving instance I

OPT(I) = optimal value of instance I

Asymptotic Worst Case Ratio(r^{∞}): For an approximate algorithm A, this measure is defined as the minimum real number $r^{\infty}(A)$ such that, for some positive integer k

$$A(I) / OPT(I) \leq r^{\infty}(A) \qquad \text{for all instances satisfying } OPT(I) \geq k$$

These worst case ratios are the usual measures for the quality of an approximation algorithm. These ratios are less than one for maximization problems, while they are greater than one for minimization problems. Hence, for maximization problems, the better the performance of the algorithm, the greater the ratio. The inverse is true for minimization problems. The ratio is smaller for better algorithms.

2.1 Classical Bin Packing Problem

In this section we first give approximation algorithms for the Classical BPP, then continue with the exact algorithms.

2.1.1 Approximation Algorithms

We start with the simplest approximation algorithms and continue with the more complicated ones. Johnson et. al. (1974), developed a number simple algorithms each of which is discussed below.

Next Fit Algorithm (NF): The simplest approximate approach to the bin-packing problem is the Next-Fit (NF) algorithm. The items are ordered arbitrarily. First item is assigned to first bin. Thereafter, each item is assigned to the current bin, if it fits; otherwise it is assigned to a new bin. As there are n assignments due to n items, the time complexity of the algorithm is O(n). A modification of the NF algorithm, called Next-Fit Decreasing (NFD), sorts the items according to nonincreasing order of their weights and proceeds like NF. Therefore the time complexity is defined by sorting n items in O(nlogn) steps. Worst case performance ratio of NF is shown as 2.

First Fit Algorithm (FF): A better algorithm First-Fit (FF), assumes that the items are indexed arbitrarily and assigns each item to the lowest indexed bin into which it fits; only when the current item cannot fit into any bin, a new bin is introduced. The time complexity of the algorithm is O(n). It has been proved by Johnson et. al. (1974) that asymptotic worst case ratio of FF algorithm is 17/10. FF algorithm has a modification like NF assuming that the items are sorted in

nonincreasing order of their weights. Modified algorithm, First-Fit Decreasing (FFD) has the same time complexity with NFD.

Best Fit Algorithm (BF): The algorithm, Best-Fit (BF), is obtained from FF by assigning the current item to the feasible bin (if any) having the smallest residual capacity where the residual capacity is the capacity of the bin minus the sum of the items already assigned. BF satisfies the same worst-case ratios as FF. As opposed to Best Fit (BF), another algorithm Worst Fit (WF) selects a partially filled bin having the largest residual capacity. BF and WF have Best-Fit Decreasing (BFD), Worst-Fit Decreasing (WFD) versions with time complexity of O(nlogn).

Minimum Bin Slack Heuristic (MBS): Minimum Bin Slack (MBS) heuristic is developed by Gupta and Ho (1999). At each step an attempt is made to find a set of items (packing) that uses the bin capacity at the greatest extent. The items are listed according to nonincreasing order of their weights. Each packing is determined in a search procedure. The search procedure tests all possible subsets of unassigned items that fit into the bin so as to minimize total residual capacity. If the algorithm finds a subset that fills the bin completely, the search is stopped. After a packing is determined, the items involved are placed in a bin and removed from the list of unassigned items preserving the sort order. The process ends when all items are packed. The authors show that MBS is superior, in terms of solution quality when compared with First-Fit Decreasing and Best-Fit Decreasing algorithms.

MBS Based Heuristics: Flezar and Hindi (2002) proposed four heuristic procedures. The first heuristic called MBS′, based on the MBS heuristic. Before the search procedure is invoked, an item is chosen and permanently fixed in the packing. The authors suggested that a good choice of the fixed item is the one with greatest size. They showed that MBS does not dominate MBS′ in terms of solution quality.

Second heuristic proposed by Flezar and Hindi (2002) is Relaxed MBS′. The modification is due to accepting some packing with positive slack, without seeking better ones. Suggestion of the authors is to run it several more times, in addition to running the procedure with zero allowable slack.

Third heuristic is Perturbation MBS′. Starting from an initial solution, their heuristic successively builds a new solution by perturbing the current one. The perturbation is done in the following way: an item of a bin with a relatively large slack is selected as fixed and a packing containing this item, considering all other items, is created by the search procedure. Items of the new packing are transferred to a new bin and when the bin becomes empty it is immediately removed.

Sampling MBS′ is another proposed heuristic. This algorithm invokes MBS′ several times by changing the ordering of items in the unassigned items list and adopts the best solution. The algorithm terminates when the lower bound on the number of bins is achieved or when the algorithm fails to improve the incumbent solution for a certain number of iterations. The order in the list is based probabilistically on the nonincreasing order of item weights. The authors

show that their MBS based heuristics give good results in reasonably short solution times.

Variable Neighborhood Search (VNS): Hansen and Mladenovic (1997) proposed an effective meta heuristic by variable neighborhood search method. The algorithm is based on moves where a move is defined as the transfer of an item from its current bin to another one or the swap of a pair of items among their respective bins. Only feasible moves are considered. The objective is to maximize the sum of the weights of the items; hence, the algorithm tries to fill the bins as much as possible.

2.1.2 Exact Algorithms

In this subsection we present the exact algorithms developed for solving Classical BPPs.

Branch and Bound Algorithms: Eilon and Christofides (1971) presented a simple depth-first enumerative algorithm based on the "best-fit decreasing" branching strategy. At any decision node, assuming that b bins have been initialized, their current residual capacities are sorted by increasing values. The item is assigned to the first bin into which it fits, if there is no available bin, a new bin is opened.

Hung and Brown (1978) presented a branch and bound algorithm for the generalization of the BPP to the case where the bins have different capacities. Their branching strategy is based on a characterization of equivalent assignments, thereby reducing the number of explored nodes. The computational

results of these algorithms indicate that they can solve only small-sized problem instances can be solved.

MTP Algorithm: Martello and Toth (1990) proposed an algorithm, MTP, based on a "first fit decreasing" branching strategy. The items are initially sorted according to nonincreasing order of their weights. The algorithm indexes the bins according to their initial order. At each decision node, the first unassigned item is put to the feasible initialized bin or to a new bin. A backtracking step involves the removal of the current item from its current bin, and its assignment to the next feasible bin.

BISON Procedure: Scholl, Klein and Jurgens (1997) present a fast hybrid procedure BISON for solving the classical bin packing problem. The procedure combines the tabu search and a branch and bound procedure. BISON computes some new lower bounds for the problem and uses well-known heuristics like FFD, WFD, and BFD to find an initial upper bound. If the upper bound is equal to the lower bound the procedure terminates, if not more powerful lower bounds are computed and compared with the upper bound, if they are not equal, procedure continues with tabu search.

The search procedure tries to find a feasible solution for the dual instance where the number of the bins is equal to the best-known lower bound. If such a solution does not exist, then the number of bins is increased by one and the procedure continues until a feasible solution is found or the number of bins reaches to the best-known upper bound.

After tabu search application if lower and upper bounds are still not equal then local lower bound method that uses a depth-first search branch and bound procedure is implemented.

The authors compare BISON with MTP and empirically show that their algorithm outperforms MTP.

2.2 Dual Bin Packing Problem (Type I)

In the following subsections we present approximation and exact algorithms for solving dual version of the bin packing problem. As in classical case, exact algorithms are not as many as approximating ones.

2.2.1 Approximation Algorithms

We again start with a simple heuristic and continue with more complicated ones.

Heuristic Simple (SI): Csirik et. al. (1999) developed two simple algorithms for bin covering problem. As a preliminary work, they adapted two heuristics of classical bin packing problem. First heuristic is Heuristic Simple (SI) that starts with sorting the items in their nonincreasing order of weights. They defined an index k_1, such that the sum of the weights up to k_1^{th} item is less than one and including $(k_1+1)^{th}$ item makes the summation greater than or equal to one. Items up to k_1 are packed in the first bin, and the slack of the bin is filled by adding the items from the end of the list, until the capacity permits. The

procedure terminates when all items are assigned. The authors proved that the worst case ratio of this heuristic is 2/3.

Improved Heuristic Simple (ISI): Second heuristic by Csirik et. al. (1999), Improved Simple Heuristic (ISI) is the improved version of Heuristic Simple. The heuristic divides the items into three groups being X, Y and Z where in each group the items are sorted according to the nonincreasing order of their weights. These sublists contain items with weights between (0, 1/3], (1/3, 1/2] and over 1/2. Algorithm is composed of two phases. Phase 1 proceeds as follows: the weight of the first item in sublist X and total weight of the first two items in sublist Y are compared and the greater weight is assigned to an empty bin. Empty parts in the bins are filled by the items from Z-sublist starting from the end. The procedure continues until X and Y sublist or Z sublist is empty. The authors aimed to assign larger items to empty bins so as to use more bins. In Phase 2, unassigned items in the sublists are assigned to new bins by two, three or according to the Next Fit heuristic. The authors proved that ISI has a worst case performance ratio of 3/4 which is better than the worst case ratio of Heuristic Simple (SI).

PTAAS Scheme: Csirik, Johnson and Kenyon (2001) developed better approximation algorithms for the bin covering problem. The authors tried to implement the Polynomial Time Asymptotic Approximation Scheme (PTAAS) to bin covering problem. They developed a parameterized algorithm A_ε such that for any fixed $\varepsilon > 0$, it runs in polynomial time and satisfies the conditions of asymptotic worst case performance ratio of $(1-\varepsilon)$. In classical bin packing, the

PTAAS can be used as very tiny items can be ignored. However in the dual version, tiny items play an important role in filling a bin. This is the reason why many authors did not prefer to use PTAAS scheme for bin covering. The authors interested in "robust" heuristics that do not only have worst case behavior close or equivalent to that of Next-Fit, but also have an average-case behavior that is probably much better. They considered discrete distributions. Their algorithm starts with identifying the set of large items. As a second step large groups are divided into $1/\varepsilon^2$ groups according to the rank so that all have the same cardinality. Algorithm considers some relaxations by rounding the large items to the value of smallest item, solving the relaxed and rounded problems and constructing a solution to the original problem. The authors also developed similar algorithms for on-line version of the problem.

First Fit Increasing Heuristic: Bruno and Downey (1985), analyzed First-Fit Increasing heuristic under the assumption that the item sizes are chosen uniformly. As the name implies, First-Fit Increasing (FFI) heuristic sorts items according to the increasing order of their weights. The authors showed that given a desired confidence of $1-\varepsilon$, the performance of the FFI policy can be made arbitrarily close to the optimal policy with any desired degree of confidence for large sample sizes. The authors derived a lower bound on the expected result of the FFI. Their proofs are based on the distribution of one-sided Kolmogorov-Smirnov statistic.

Pairing Heuristic: Csirik et al (1991), studied probabilistic analysis of the algorithms for dual bin packing problems. Their assumption is based on the fact

that item weights are generated from a uniform distribution. The authors adapted a pairing heuristic (PA) such that the largest unassigned item is paired with the smallest unassigned item so as to cover a bin. If no such item exists, all remaining items are added to the most recently opened bin. Beside PA, the authors analyzed the expected behavior of the Next-Fit, and Next-Fit Decreasing heuristics. Their main interesting conclusion is that the expected performance of the NF heuristic is better than the expected behavior of the NFD heuristic where the reverse is true for the classical bin packing problem. However, the authors could not give an intuitive explanation for this result.

2.2.2 Exact Algorithms

Labbe, Laporte and Martello (1995), provided an exact algorithm for the DBP. They assumed that the items are ordered according to nonincreasing order of their weights. They developed some reduction criteria, one of which states that if the sum of the weights of a combination of items is equal to the bin capacity, then those items are eliminated. Another reduction criterion is to assign items to the same bin whenever the sum of the weights of the items is greater than or equal to the capacity. Recall that this is similar to the Heuristic Simple (SI), developed by Csirik et al (1999).

The authors also derived some upper bounds for the problem. Their proposed algorithm first fills the empty spaces in the bins by adding a small weighted item.

The branching strategy of the algorithm is based to go on more promising nodes, which is similar to the inverse of "best-fit decreasing" strategy developed by Eilon and Christofides (1971). Results of the authors' experimentations indicate that the upper bounds are very sharp and the ratio to the lower bound is always close to one. The main result of the computational experiments is that the combined effect of the reduction criteria and the upper bounds made a hard problem relatively easy to solve one, in most cases.

2.3 Dual Bin Packing Problem (Type II)

In this section, approximating algorithms for Type II Dual Bin Packing problem are presented. This type of problem is not as widely studied as Type I, and there are only a few approximating algorithms.

2.3.1 Approximation Algorithms

First Fit Increasing Heuristic: Coffman, Leung, and Ting (1978) developed First- Fit Increasing (FFI) algorithm. The algorithm sorts the items according to the increasing order of weights, hence aims to find a maximum subset of smaller pieces. The authors proved that worst case performance ratio of FFI is 3/4.

Probabilistic Analysis of First Fit Increasing Heuristic: Foster and Vohra (1989), studied probabilistic analysis of First- Fit Increasing heuristic for the second version of the dual bin packing problem. The authors show that when the items are independent and identically distributed, then the relative error of the

algorithm approaches to zero as the number of items approaches to infinity. The authors also considered an on-line version of the problem and proposed a simple heuristic.

2.4 Variable Sized and On-line Bin Packing Problems

Variable sized bin packing problem (VSBP) and online version of the bin packing problem are studied in the literature as well. In this section we give a brief description of methods developed for these problems.

Friesen and Langston (1986), gave three heuristics for the variable sized bin packing problem, called Next-Fit using largest bins only (NFL), First-Fit decreasing using largest bins, and repacking to smallest possible bins (FFDLR) and First-Fit decreasing using largest bins but shifting when necessary (FFDLS). As the names of the heuristics imply, these algorithms are the same as Next-Fit and First-Fit Decreasing heuristics. Only difference is the selection criteria of the bins.

Zhang et. al. (1997), modified some algorithms of classical bin packing to the variable sized on-line bin packing problem by allowing to open one bin at a time. However the authors proved that even FFD has worst case performance ratio of 2 and this ratio cannot be improved by generalizing the case that allows to open more than one bin at a time.

Zhang et. al. (2001), developed the dual version of the variable sized bin packing problem so called bin batching problem where the objective is to pack

the items into maximum number of nonidentical bins. The authors developed an upper bound and derived worst case performance ratios for their problem.

Xing et. al. (2002), defined a new bin backing problem with over sized items. In this case, item weights are allowed to be greater than the bin capacity. The objective is to pack all items into bins so as to minimize the number of bins used. The authors aimed to minimize the difference between the sum of the capacities of the bins and the sum of the item weights assigned to the bins. The authors developed a two-stage procedure, first packing the over-sized items into largest bins and then packing the remaining ones. They proved that the two-stage procedure did not perform as satisfactory as expected, so the authors developed a new on-line algorithm with an asymptotic worst case ratio not greater than 7/4.

Azar and Regev (2001), defined another problem called on-line bin stretching. In bin-stretching problem, they fix the number of bins and try to pack the items while stretching the size of the bins as small as possible. The authors presented two on-line algorithms that guarantee a stretching factor of 5/3 for any number of bins. They then combined two algorithms and designed an algorithm with stretching factor of 1.625 for any number of bins. The authors proved that the best lower bound for any algorithm is 4/3 for any number of bins greater than or equal to 2. They indicated that the bin-stretching problem is equivalent to the classical scheduling problem with known makespan.

CHAPTER 3

MINIMIZING TOTAL AND MAXIMUM OVERDEVIATION

In this chapter, we introduce two bi-objective bin packing problems where the objectives are to minimize the total and the maximum deviation over capacity together with the minimization of the number of bins, and present mathematical models for these problems.

3.1 Problem Statement

Given n items, with weights w_i and bins with capacity c, the aim is to assign all items into minimum number of bins so that total weight of items in each bin does not exceed c. This problem is known as Classical Bin Packing problem, BPP (Johnson et al, 1974).

In this study we relax the capacity constraint of the classical BPP, and allow deviations over capacity. In other words, we allow to assign items to bins even if the bin usage exceeds c. The deviation over capacity is referred to as overdeviation and defined as:

Overdeviation: Total weight of items in the bin minus the capacity of the bin. Hence,

Overdeviation = Max $\{0, \sum_i w_i\, x_{ij} - c\}$

where

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

Total overdeviation: Sum of overdeviation of all of the bins.

Total overdeviation = $\sum_j$ Max $\{0, \sum_i w_i\ x_{ij} - c\}$

Maximum overdeviation: Maximum overdeviation among all the bins.

Maximum overdeviation = Max_j { Max $\{0, \sum_i w_i\ x_{ij} - c\}$ }

In the thesis, we first consider the problem of minimizing total overdeviation and minimizing the number of bins. Next, we deal with the problem of minimizing the maximum overdeviation together with minimizing the number of bins.

For the sake of clarity, we refer the problem of minimizing total overdeviation as Problem I, and the problem of minimizing maximum overdeviation as Problem II. The objectives of these problems are summarized below.

Objectives of Problem I:

- Minimize the number of bins
- Minimize the total overdeviation

Objectives of Problem II:

- Minimize the number of bins
- Minimize the maximum overdeviation

As we decrease the number of bins, the total or the maximum deviation over capacity will increase. Overdeviations can only be tolerated if they cause reduction in the total number of bins. Therefore, the objectives of both problems are conflicting. When objectives conflict, there may not be a single solution which optimizes both objectives at the maximum extent. Hence, our aim is to find the set of efficient solutions.

Before proceeding further, for the sake of clarity, we give some definitions in the following subsection.

3.2 Some Definitions

In this section we define dominance and efficiency.

Dominance: Let, $f_i(x)$ be the value of objective i, at solution x. For any minimization problem, solution $x \in X$ dominates solution $x' \in X$ if

$f_i(x) \leq f_i(x')$ $\quad \forall\, i$ and

$f_i(x) < f_i(x')$ $\quad \exists\, i$

Efficiency: Solution $x' \in X$ is efficient (nondominated) if there exists no solution dominating it.

Let us, give an example. Assume that we have two objectives, $f_1(x)$ and $f_2(x)$, to minimize. Figure 3.1 shows the possible objective function values of our objectives.

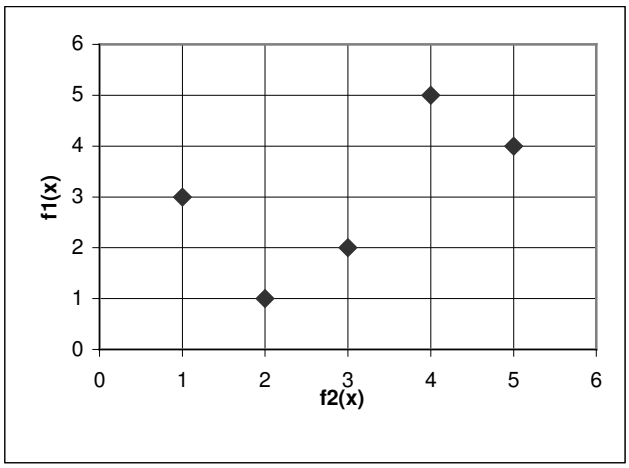

Figure 3.1 An Example Problem

Suppose we have five feasible solutions to our problem. As can be observed from the figure,

Solutions d and e are dominated by a, b and c.

Solution c is dominated by solution b.

Solutions a and b are not dominated by any solution hence they are efficient.

3.3 Mathematical Formulation

We present the mathematical formulations of the two problems in this section. Below, we present indices and additional notation that will be used throughout the thesis:

Parameters:

n the number of items

m the number of bins

Indices:

i item index, where $i = 1, 2, 3....n$

j bin index, where $j = 1, 2, 3....m$

Decision Variables:

f maximum overdeviation

d_j overdeviation from capacity of bin j

$$y_j = \begin{cases} 1 & \text{if bin } j \text{ is selected} \\ 0 & \text{otherwise} \end{cases}$$

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

Mathematical formulations of our problems, Problem I and Problem II are as follows:

Constraints:

Constraint set 3.1 ensures that every item is assigned to exactly one bin.

$$\sum_j x_{ij} = 1, \qquad \forall \text{ i} \qquad (3.1)$$

Constraint set 3.2 guarantees that if an item is assigned to a bin, then the bin should be opened.

$$x_{ij} \leq y_j, \qquad \forall \text{ i and j} \qquad (3.2)$$

Constraint set 3.3 gives the overdeviation due to bin j.

$$\sum_i w_i x_{ij} - d_j \leq c y_j, \qquad \forall \text{ j} \qquad (3.3)$$

Variables are defined as binary or continuous with the following constraints.

$$y_j \leq 1 \qquad \forall \text{ j}$$

$$y_j \geq 0 \qquad \forall \text{ j}$$

$$d_j \geq 0 \qquad \forall \text{ j}$$

$$x_{ij} = 0, 1 \qquad \forall \text{ i and j}$$

Note that y_j takes binary values even though it is defined as a continuous variable since x_{ij} is a binary variable.

Objective Functions:

The model has two objectives; minimization of the total number of bins used, and the minimization of total overdeviation.

$$\text{Minimize} \quad \sum_j y_j \tag{3.4}$$

$$\text{Minimize} \quad \sum_j d_j \tag{3.5}$$

The above model is a mixed integer linear program (MIP). The large number of binary variables makes its solution impractical, even infeasible with avaliable MIP softwares.

The above formulation can be modified for Problem II as follows:

Constraints:

In addition to our constraint sets given above, we have one additional constraint. Constraint set 3.6 defines the maximum overdeviation f.

$$f \geq d_j, \qquad \forall \text{ j} \tag{3.6}$$

Objective Functions:

The model for Problem II has two objectives: minimization of the total number of bins used, and the minimization of the maximum overdeviation.

Minimize $\sum_j y_j$ (3.7)

Minimize f (3.8)

Note that, the first objective of the models, is the objective of classical bin packing problem. In a classical bin packing problem capacity cannot be exceeded whereas in our models we can exceed capacity. To generate the efficient solutions, we minimize the total overdeviation from bin capacity ($\sum_j d_j$) [for Problem I] and the maximum overdeviation (f) [for Problem II] for all possible values of the number of bins. The maximum number of bins that can be used is equal to the number of items, and the minimum number of bins that can be used is equal to one.

3.4 Ranges for Efficient Solutions

In this section, we define the ranges for efficient solutions. Let (x,y) represent a solution where the number of bins is equal to x and the total/maximum overdeviation is equal to y.

Efficient solutions lie between the following two solutions.

- Solution with one bin, hence with the maximum total overdeviation.

  Note that, an upper bound on the total/maximum overdeviation of all efficient solutions is ($\sum_i w_i$ – c), and (1, $\sum_i w_i$ – c) is an efficient solution.

- Solution with zero total overdeviation

Remark 3.1

Finding the efficient solution with zero total overdeviation is equivalent to solving classical bin packing problem.

It is mentioned that classical bin packing problem does not allow any overdeviation, therefore its optimum solution, z*, gives an efficient solution (z*,0).

To summarize;

Range for the number of bins = [1, z*]

Range for the total/maximum overdeviation = $[0, \sum_i w_i - c\ ]$

Hence; efficient solutions lie between (z*,0) and $(1, \sum_i w_i - c\ )$

In the following chapter, procedures developed to generate all efficient solutions for Problem I and Problem II are presented.

CHAPTER 4

SOLUTION PROCEDURES

In this chapter, we present the procedures for solving our problems together with upper and lower bounds on the objective function values.

4.1 Generation of All Efficient Solutions

Remark 3.1, states that finding an efficient solution with zero total/maximum overdeviation is equivalent to solving a classical BPP. To generate the set of efficient solutions, we develop a two stage procedure. In the first step, we solve the classical BPP and minimize the number of bins for zero total/maximum overdeviation [to generate the solution (z*,0)]. In the second step, we minimize the total/maximum overdeviation for the given number of bins.

In Step I, we describe how the classical BPP is solved, and the optimal number of bins, z*, is found for zero total/maximum overdeviation.

STEP I (MIP I):

In this step the following MIP is solved.

Constraints:

$$\sum_j y_j \leq Y_{UB} \tag{4.1}$$

$$\sum_j y_j \geq Y_{LB} \tag{4.2}$$

$$\sum_j x_{ij} = 1, \qquad \forall \text{ i} \tag{4.3}$$

$$\sum_i w_i x_{ij} \leq c y_j, \qquad \forall \text{ j} \tag{4.4}$$

$$x_{ij} \leq y_j, \qquad \forall \text{ i and j} \tag{4.5}$$

$$y_j \leq 1, \qquad \forall \text{ j} \tag{4.6}$$

$$x_{ij} = 0, 1, \qquad \forall \text{ i and j}$$

$$y_j \geq 0, \qquad \forall \text{ j}$$

Constraint sets (4.1) and (4.2) are the upper and lower bounds on the optimal number of bins (these bounds are defined in sections 4.1.1 and 4.1.2). We introduce these constraints to speed-up the solution process of the model. The remaining constraints are the ones in our original mathematical model. The only difference is that we set decision variable d_j to zero and modify constraint set 3.3 (see constraint set 4.4).

Objective Function:

The objective function is to minimize the number of bins.

$$\text{Minimize} \qquad z = \sum_j y_j \tag{4.7}$$

Let z* be the optimal objective function value of MIP I. In Step II, we generate the remaining efficient solutions by solving a new MIP (MIP II) iteratively.

MIP II, generates an efficient solution with z*-1 number of bins, which is an adjacent efficient solution to the optimal solution of MIP I (solution with zero total overdeviation). The model tries to minimize the total/maximum overdeviation while keeping the number of bins at z*-1. In order to generate all efficient solutions, we iteratively solve MIP II by updating the number of bins (decreasing the number of bins by one each time).

STEP II (MIP II):

In this step, the following MIP minimizes the total overdeviation. Let;

$$k^* = \begin{cases} z^* & \text{if we are solving STEP II for the first time} \\ k^*-1 & \text{otherwise} \end{cases}$$

Constraints:

$$\sum_j y_j = k^* - 1 \qquad (4.8)$$

$$\sum_j d_j \leq D_{UB} \qquad (4.9)$$

$$\sum_j d_j \geq D_{LB} \qquad (4.10)$$

$$\sum_j x_{ij} = 1, \qquad \forall i \qquad (4.11)$$

$$\sum_i w_i x_{ij} - d_j \leq c y_j, \qquad \forall j \qquad (4.12)$$

$$x_{ij} \leq y_j, \qquad \forall i \text{ and } j \qquad (4.13)$$

$$y_j \leq 1, \qquad \forall j \qquad (4.14)$$

$$x_{ij} = 0, 1, \qquad \forall i \text{ and } j$$

$$y_j \geq 0, \qquad \forall j$$

Note that, while solving MIP II for the first time, the optimal objective function value of MIP I is used in constraint set 4.8 for k*. In the following iterations, we decrease the value of k* by one. We again introduce upper and lower bounds on the objective function by constraints (4.9) and (4.10) to increase solution speed. The rest of the constraints are the ones in our original model.

Objective Function:

Our objective is to minimize total overdeviation.

$$\text{Minimize} \qquad t = \sum_j d_j$$

For minimizing maximum overdeviation Step II is modified as follows:

Constraints:

$$f \leq F_{UB} \tag{4.15}$$

$$f \geq F_{LB} \tag{4.16}$$

$$f \geq d_j, \qquad \text{for all j} \tag{4.17}$$

Again we have upper and lower bounds on the objective function (see constraint sets 4.15 & 4.16), and constraint set 4.17 is added for finding the maximum overdeviation. In addition to these, we also have constraint sets 4.8, 4.11, 4.12, 4.13, 4.14.

Objective Function:

Our objective is to minimize maximum overdeviation.

Minimize f

In the following subsections, we discuss the development of upper and lower bounds used in MIP I and MIP II. For the ease of reference, we give MIP I, MIP II for Problem I and MIP II for Problem II in compact forms in Figures 4.1, 4.2 and 4.3 respectively.

Minimize $z = \sum_j y_j$

$$\sum_j y_j \le Y_{UB} \qquad (4.1)$$

$$\sum_j y_j \ge Y_{LB} \qquad (4.2)$$

$$\sum_j x_{ij} = 1, \quad \text{for all i} \qquad (4.3)$$

$$\sum_i w_i x_{ij} \le c y_j, \quad \text{for all j} \qquad (4.4)$$

$$x_{ij} \le y_j, \quad \text{for all i and j} \qquad (4.5)$$

$$y_j \le 1, \quad \text{for all j} \qquad (4.6)$$

$x_{ij} = 0, 1,$ for all i and j

$y_j \ge 0,$ for all j

Figure 4.1 MIP I

Minimize $t = \sum_j d_j$

$$\sum_j y_j = k^* - 1 \qquad (4.8)$$

$$\sum_j d_j \leq D_{UB} \qquad (4.9)$$

$$\sum_j d_j \geq D_{LB} \qquad (4.10)$$

$$\sum_j x_{ij} = 1, \quad \text{for all i} \qquad (4.11)$$

$$\sum_i w_i x_{ij} - d_j \leq c y_j, \quad \text{for all j} \qquad (4.12)$$

$$x_{ij} \leq y_j, \quad \text{for all i and j} \qquad (4.13)$$

$$y_j \leq 1, \quad \text{for all j} \qquad (4.14)$$

$x_{ij} = 0, 1,$ for all i and j

$y_j \geq 0,$ for all j

Figure 4.2 MIP II for Problem I

Minimize $b = f$

$\sum_j y_j = k^* - 1$		(4.8)
$\sum_j x_{ij} = 1,$	for all i	(4.11)
$\sum_i w_i x_{ij} - d_j \leq cy_j,$	for all j	(4.12)
$x_{ij} \leq y_j,$	for all i and j	(4.13)
$y_j \leq 1,$	for all j	(4.14)
$x_{ij} = 0, 1,$	for all i and j	
$y_j \geq 0,$	for all j	
$f \leq F_{UB}$		(4.15)
$f \geq F_{LB}$		(4.16)
$f \geq d_j,$	for all j	(4.17)

Figure 4.3 MIP II for Problem II

4.1.1 Development of Lower Bounds

In this subsection, the methods used to develop lower bounds on the objective function values are introduced. We generate these bounds for both steps of the problems.

Below, we introduce a method for finding the lower bound for the first step of the procedure. The first step is common for both of our problems, and therefore the lower bound given in Theorem 4.1 is valid for both problems.

Theorem 4.1

$\lceil \sum_i w_i$ / capacity $\rceil$ is a lower bound on the minimum number of bins with zero total/maximum overdeviation.

Proof

An optimal solution to the problem with item splitting is $\lceil \sum_i w_i$ / capacity$\rceil$ (Marthello and Toth, 1990). As an optimal solution to any relaxation of the problem leads to lower bound, $\lceil \sum_i w_i$ / capacity $\rceil$ is a valid lower bound on the number of bins, Y.

Hence, we set

$$Y_{LB} = \lceil \sum_i w_i \text{ / capacity } \rceil \qquad (4.18)$$

in our formulation.

Theorem 4.2 introduces a lower bound on the objective function of the second step of our procedure for Problem I (where the objective is to minimize total overdeviation).

Theorem 4.2

Max $\{0, \lceil \sum_i w_i$ – (number of bins × c)$\rceil\}$ is a lower bound on the optimal total overdeviation.

Proof

(Number of bins × c) is an upper bound on the utilization of all bins. So,

$\lceil \sum_i w_i$ – (number of bins × c)$\rceil$ becomes a lower bound on the load excess of (number of bins × c), therefore it is a valid lower bound on the total overdeviation.

Hence;

$$D_{LB} = \lceil \text{Max } \{0,[\sum_i w_i - (\text{number of bins} \times \text{capacity})]\} \rceil \qquad (4.19)$$

in our formulation.

Theorem 4.3, presents the lower bound on the objective function of second step of our procedure for Problem II.

Theorem 4.3

Max $\{0, \lceil [\sum_i w_i$ – (number of bins × c)] / n$\rceil\}$ is a lower bound on the optimal maximum overdeviation.

Proof

(Number of bins × c) / n is an upper bound on the utilization of all bins. So, note that $\lceil (\sum_i w_i)$ / n – (number of bins × c) / n$\rceil$ becomes a lower bound on the assignment over (number of bins × c) / n, therefore it is a valid lower bound on the maximum overdeviation.

Hence;

$$F_{LB} = \lceil \text{ Max } \{0, \lceil [\sum_i w_i - (\text{number of bins} \times c)] / n \rceil \} \quad (4.20)$$

in our formulation.

These lower bounds are the rounded up results of the LP relaxations of the problems. One example for this, is the lower bound of Classical BPP. The continuous relaxation of BPP problem can be immediately solved by values x_{ii} = 1, x_{ij} = 0 (j ≠ i) and $y_i = w_i / c$ (Marthello and Toth, 1990).

4.1.2 Development of Upper Bounds

For the first step (MIP I), we use Best Fit Decreasing Heuristic as an upper bound for classical BPP, since BFD is easy to implement and it has a good worst case performance ratio. For the sake of completeness, we give the details of the heuristic. We start the procedure by sorting the items according to the nonincreasing order of their weights. Then, starting from the first item of the list we assign the items to a feasible bin with smallest remaining capacity. The stepwise description of the heuristic is given below.

Best-Fit Decreasing Heuristic (Garey & Johnson, 1979) :

1. Sort the items in their nonincreasing order of weights
2. Put the first item to a feasible bin with smallest remaining capacity
3. If there is no feasible bin available, open a new bin.
4. Continue until all items are assigned.

Worst case performance ratio of BFD is proven to be 1.222 by Garey, Coffman and Johnson (1984).

First step of the problems are the same, we use above heuristic for both problems. Below, a constructive heuristic and an improvement of that heuristic are given for the second step of Problem I.

We modify the Best Fit Decreasing Heuristic for the case where the number of bins is given. The only difference is, when all the bins are full, we assign the item to any bin, instead of opening a new bin. We present our heuristic below:

A Constructive Heuristic:

1. Sort the items in their nonincreasing order of weights.
2. Assign the first item of the list to the feasible bin with smallest remaining capacity. If there is no slack in any one of the bins, assign the item to an arbitrarily selected bin.
3. Continue until all the items are assigned.

We improve the constructive heuristic by the following improvement heuristic.

An Improvement Heuristic:

If $(D_{UB}-D_{LB})/D_{UB} \geq R$; where R is equal to 0.4 then; starting from the solution found from the constructive heuristic we apply our improvement heuristic. We obtain 0.4 as a suitable value for R by experimentation (i.e. starting

from the value 0.1, several R values are implemented and we found out that, up to the value 0.4, there is no need to apply improvement heuristic).

We eliminate some of the bins with their assigned items to decrease the problem size and consider only the ones with no overdeviation. Those bins are then sorted in nonincreasing order of their total assigned weights. Hence, the bins with total assigned weight equal to their capacity are eliminated first. The improvement heuristic is given below.

1. Take the bins with zero overdeviation to a list
2. Sort the bins of the list in nonincreasing order according to their assigned weights
3. Eliminate the first B bins together with their assigned items
4. Solve the MIP II in Step II (section 4.1) for the reduced problem

An alternative for third step of the improvement heuristic may be to eliminate the bins one by one unless slacks are zero, instead of eliminating B bins. However it would blow up the computational time. So, we decided to eliminate B bins.

Schematic view of Improvement Heuristic can be seen in Figures 4.4, 4.5 and 4.6.

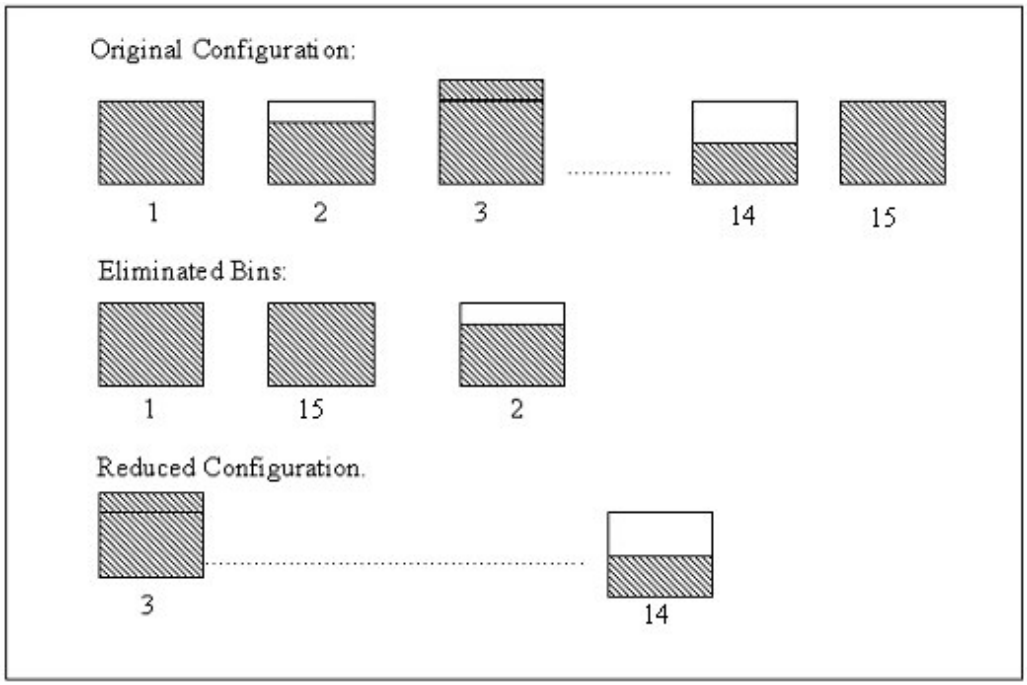

Figure 4.4 Figure of elimination procedure

In the illustrative example shown in Figure 4.4, B is taken to be 3. So, we have to eliminate 3 bins from the original configuration. Bin number 3 is not taken into the list as its capacity is exceeded. When we sort the bins in nonincreasing order according to their assigned weights, bins with number 1, 15 and 2 are taken into the list. The bin with number 14 is not eliminated since its assigned weight is less than the other three bins. Note that the total overdeviations of the eliminated bins are zero.

Suppose, the MIP II solution of the reduced configuration of Figure 4.4 is as given in Figure 4.5.

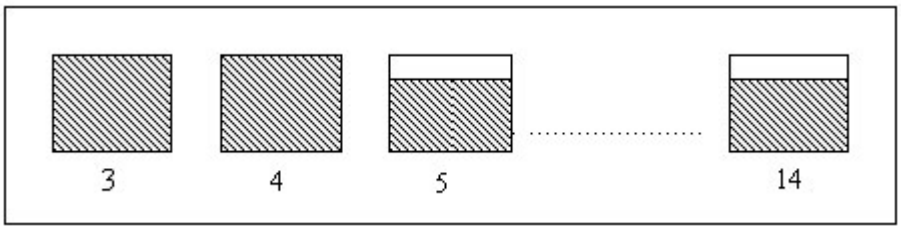

Figure 4.5 Solution of reduced configuration

If there is no overdeviation as in the reduced configuration of Figure 4.5, then there is no need to solve the original problem, and the combined configurations, (i.e. eliminated bins and solution of reduced configuration) will give the solution (see Figure 4.6).

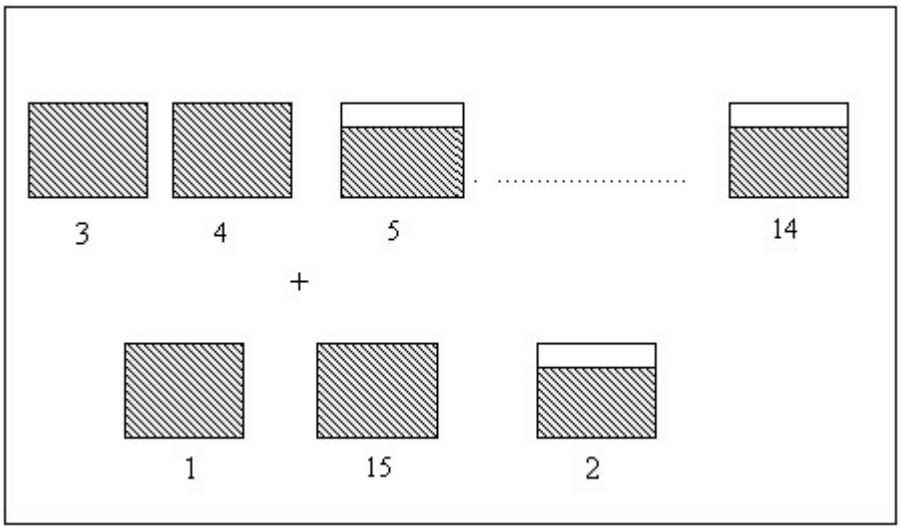

Figure 4.6 Final Solution by Adding Two Configurations

How to find B?

An important concern is how to decide on the number of bins to be eliminated. We develop a method for finding a suitable value for B by

experimentation. In the experiments, we observe that if we choose too small or too big a B value, we cannot obtain a satisfactory improvement.

We set,

$$B = \lceil s^* \times 10(w_a / \text{capacity}) \rceil$$

where

$$s^* = \begin{cases} 1 & \text{if } n = 50 \\ 1.5 & \text{if } n = 75 \\ 2 & \text{if } n = 100 \end{cases}$$

$w_a = \sum w_i / n$, i.e. average weight of items,

For example, assume that we have 50 items with total weight of 2872. Then the average weight of items is 57.44 (2872 / 50). Let the capacity be equal to 100. Hence, $B = \lceil 10(57.44 / 100) \rceil = 6$.

The formulation takes into account the capacity, the weight of items and the number of items. So, as the number of items and the weight of these items increase, the number of the bins increases, and we have to eliminate more bins. However, as the capacity of the bins increases the reverse is true.

Finally, we present a heuristic for minimizing maximum overdeviation. As this problem is similar to the problem of minimizing makespan with parallel machines without preemptions, we used a well known heuristic for upper bound calculation.

Heuristic for Minimizing Maximum Overdeviation (Graham, 1969):

1. Sort the items in their nonincreasing order of weights.

2. Assign first item of the list to the feasible bin with smallest remaining capacity. If there is no slack in any one of the bins, assign the item to bin with smallest overdeviation.

3. Continue until all items are assigned.

The main difference of this heuristic from our constructive heuristic for Problem I is, when all the bins are full we assign the item to the bin with smallest overdeviation instead of assigning it to an arbitrary bin, as our main concern is to balance the overdeviations of the bins so as to minimize maximum overdeviation.

Worst case performance of this heuristic is proven to be 4/3-1/3n where n is the number of bins (Graham, 1969). Due to this satisfactory worst case performance, we did not implement an improvement heuristic.

4.2 A Preliminary Remark

In this section we present an important property of Problem I, by Theorem 4.4.

Theorem 4.4

When all bins are fully loaded, the efficient solutions can be obtained through the following procedure:

Let TD(n) be the minimum total overdeviation where the number of bins is equal to n. Then a unit reduction in the number of bins increases the total overdeviation by c units. That is:

$$TD(n-1) = TD(n) + c$$

Proof

Total overdeviation with n bins is given by

$$TD(n) = \sum_i w_i - n \times c$$

Total overdeviation with n-1 bins is given by

$$TD(n-1) = \sum_i w_i - (n-1) \times c$$

The difference between two deviations is as follows:

$$TD(n-1) - TD(n) = \sum_i w_i - (n-1) \times c - \sum_i w_i + n \times c = c$$

Hence;

$$TD(n-1) = TD(n) + c$$

□

4.3 An Approach for Problem I

In this section, using the above results, we develop an approach for Problem I.

STEP I: Solve MIP I

STEP II: Using the optimal objective function value z* of MIP I as a parameter in MIP II, solve MIP II.

Continue to solve MIP II by decreasing the number of bins by one until there is no slack in any of the bins.

Find the remaining efficient solutions by using the result of Theorem 4.4.

4.4 An Example for Problem I

An example problem is presented in this section to illustrate the procedure for Problem I.

Let;

Number of items = 100

Capacity = 100

Suppose the weights of items are generated between [1,50] from uniform distribution, and they add up to;

$$\sum_i w_i = 2745$$

We implement our procedure, for the above data.

STEP I (MIP I)

To solve Step I, first we calculate upper and lower bounds on the objective function.

Calculating Bounds for Step I:

Lower bound for Step I is calculated by the result of Theorem 4.1.

$$Y_{LB} = \lceil \sum_i w_i \text{ / capacity} \rceil$$

$$Y_{LB} = \lceil 2752 / 100 \rceil$$

$$Y_{LB} = 28$$

Upper bound for Step I is calculated by using Best Fit Decreasing Heuristic. First we sort the items in their nonincreasing order of weights. Then, starting from the first item of the list, we assign the item to a feasible bin with smallest remaining capacity. If there is no feasible bin available, we open a new bin. Assigning all items accordingly, we find the number of bins as;

$$Y_{UB} = 33$$

The optimal solution of Step I gives:

$$z^* = 29$$

After finding z*, we continue our procedure by MIP II.

STEP II (MIP II)

Iteration 1: Minimize total overdeviation with z*-1 = 28 bins

Again, we calculate the upper and lower bounds.

Calculating Bounds for Step II:

Lower bound for Step II is calculated using Theorem 4.2

$$D_{LB} = \lceil \text{ Max } \{0, [\sum_i w_i - (\text{number of bins} \times \text{capacity})]\} \rceil$$

The number of bins is equal to z*-1

$$D_{LB} = \lceil \text{ Max } \{ 0,[2752 - (28 \times 100)] \} \rceil$$

$$D_{LB} = 0$$

To calculate the upper bound, first we use the constructive heuristic. By sorting the items in their nonincreasing order of weights, we assign the first item of the list to the bin with smallest remaining capacity, if there is no slack in any of the bins, then we assign the item arbitrarily to any bin. The upper bound is found as,

$$D_{UB} = 20$$

Then we decide whether we need an improvement or not, as follows:

$(D_{UB} - D_{LB}) / D_{UB} = 1 > 0.4$

Hence, we improve the upper bound by improvement heuristic as,

$B = \lceil s^* \times 10(w_a / \text{capacity}) \rceil$

$s^* = 2$

$w_a = 27.52$

capacity = 100

$B = \lceil 2 \times 10(27.52 / 100) \rceil = 6$

We eliminate 6 bins.

Then, result of the reduced problem is 7, hence

$D_{UB} = 7$

By solving Step II

t* (minimum total overdeviation) = 0

There are still unused capacities in some bins

Iteration 2: Minimize total overdeviation with 27 bins

$D_{LB} = 52$

$D_{UB} = 54$

$(D_{UB} - D_{LB}) / D_{UB} = 0.04$

We do not implement improvement heuristic, hence $D_{UB} = 54$

By solving the model;

$D^* = 54$

There are unused capacities in some bins

Iteration 3: Minimize total overdeviation with 26 bins

$D_{LB} = 152$

$D_{UB} = 152$

$D_{UB} = D_{LB}$ hence $D^* = 152$

There is no slack in any of the bins. Using the result of Theorem 4.4, we continue to generate efficient solutions by increasing total overdeviation by c units and decreasing the number of bins by 1. We stop when the number of bins reaches to its lower limit of one.

Number of bins = 25	$D^* = 152 + 100 = 252$
Number of bins = 24	$D^* = 252 + 100 = 352$
Number of bins = 23	$D^* = 352 + 100 = 452$
.	.
.	.
.	.
Number of bins = 1	$D^* = 2652$

All efficient solutions are shown in Figure 4.7. Solutions obtained by MIP I and MIP II are depicted in Figure 4.8. The linear behaviour of solutions after the bins are fully utilized can be easily seen in Figure 4.7.

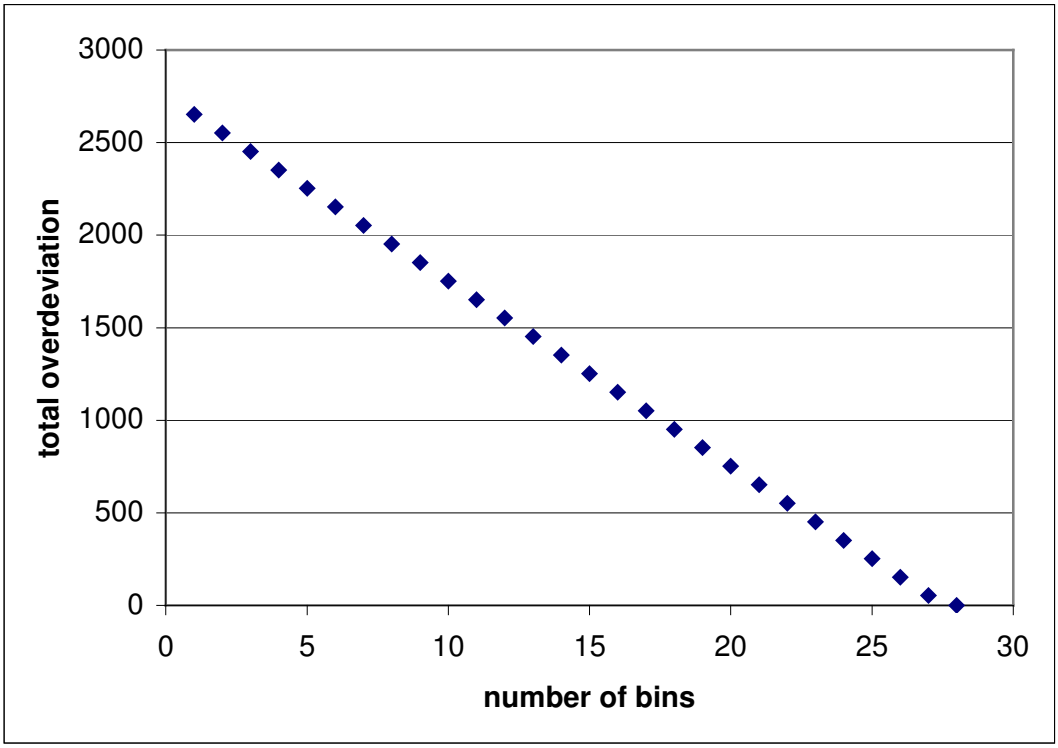

Figure 4.7 All Efficient Solutions

When the number of bins reaches 26, the solutions generated afterwards show a linear trend (see Figure 4.7). From Theorem 4.4, it is known that after all bins are fully loaded, by removing one bin, the total overdeviation increases by c units. Solutions in the cycle are the ones generated by solving MIP I and MIP II.

In Figure 4.8, the efficient solutions generated before all the bins are fully loaded, can be seen more clearly. These solutions are generated by solving MIP I, and MIP II. As can be seen, there is no such linear trend.

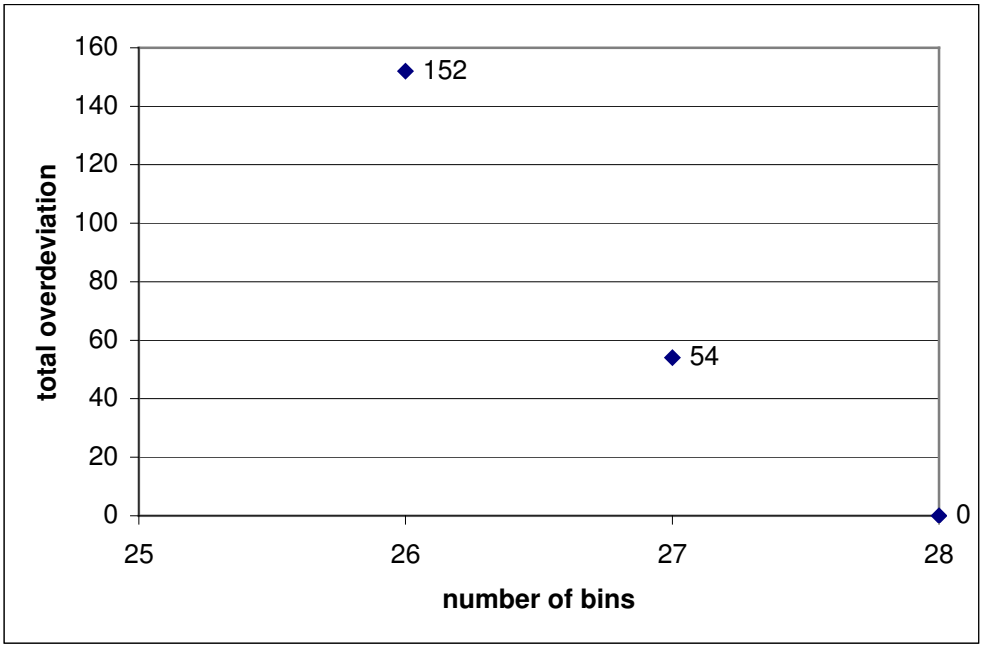

Figure 4.8 Efficient Solutions Found by Solving MIP I & MIP II

4.5 An Approach for Problem II

In this section, the approach developed for Problem II is presented.

Main difference between Problem I and Problem II is that, there is no such linear relationship between the number of the bins and the maximum overdeviation after some solution as stated in Theorem 4.4. So we have to solve Step II (MIP II) for all possible number of bins.

STEP I: Solve MIP I

STEP II: Using the optimal objective function value z* of MIP I as a parameter in MIP II, solve MIP II

Continue to solve MIP II by decreasing number of bins by one for all possible number of bins

4.6 An Example for Problem II

Let;

Number of items = 100

Capacity = 200

Suppose the weights of items are generated between [1,50] from uniform distribution, and they add up to;

$$\sum_i w_i = 2378$$

We start to implement our procedure for the above data.

STEP I (MIP I):

To solve Step I, first we calculate upper and lower bounds on the objective function.

Calculating Bounds for Step I:

Lower bound for Step I is calculated by using the result of Theorem 4.1.

$$Y_{LB} = \lceil \sum_i w_i / \text{capacity} \rceil$$

$$Y_{LB} = \lceil 2378 / 200 \rceil$$

$$Y_{LB} = 12$$

Upper bound for Step I is calculated by using Best Fit Decreasing Heuristic and found as;

$$Y_{UB} = 13$$

The optimal solution of Step I gives:

$$z^* = 12$$

After finding z*, we continue our procedure by MIP II.

STEP II (MIP II):

Iteration 1: Minimize maximum overdeviation with z*-1 (11) bins

Again, we calculate the upper and lower bounds.

Calculating Bounds for Step II:

Lower bound for Step II is calculated from Theorem 4.3

$$F_{LB} = \lceil \text{Max } \{0, [\sum_i w_i - (\text{number of bins} \times \text{capacity})] / n\} \rceil$$

The number of bins is equal to z*-1

$$F_{LB} = \lceil \text{Max} \{ 0, [2378 - (11 \times 200)] / 11 \} \rceil$$

$$F_{LB} = 17$$

For upper bound, we use the heuristic by Graham (1969). We sort the items in their nonincreasing order of weights, we assign the first item of the list to the bin with smallest remaining capacity, if there is no slack in any of the bins, then we assign the item to the bin with smallest overdeviation. Then, the upper bound is:

$$F_{UB} = 17$$

$F_{UB} = F_{LB}$ hence $F^* = 17$

Iteration 2: Minimize maximum overdeviation with 10 bins

$$F_{LB} = 38$$

$$F_{UB} = 39$$

By solving the model;

$$F^* = 39$$

Iteration 3: Minimize maximum overdeviation with 9 bins

$$F_{LB} = 65$$

$$F_{UB} = 65$$

$F_{UB} = F_{LB}$ hence $F^* = 65$

Although there is no slack in any of the bins we should continue to generate efficient solutions by solving MIP II. We stop when the number of bins reaches its lower limit of one.

By solving MIP II, following results are obtained:

Number of bins = 8	F* = 99
Number of bins = 7	F* = 140
Number of bins = 6	F* = 197
Number of bins = 5	F* = 276
Number of bins = 4	F* = 395
Number of bins = 3	F* = 593
Number of bins = 2	F* = 969
Number of bins = 1	F* = 2178

All efficient solutions are shown in Figure 4.9.

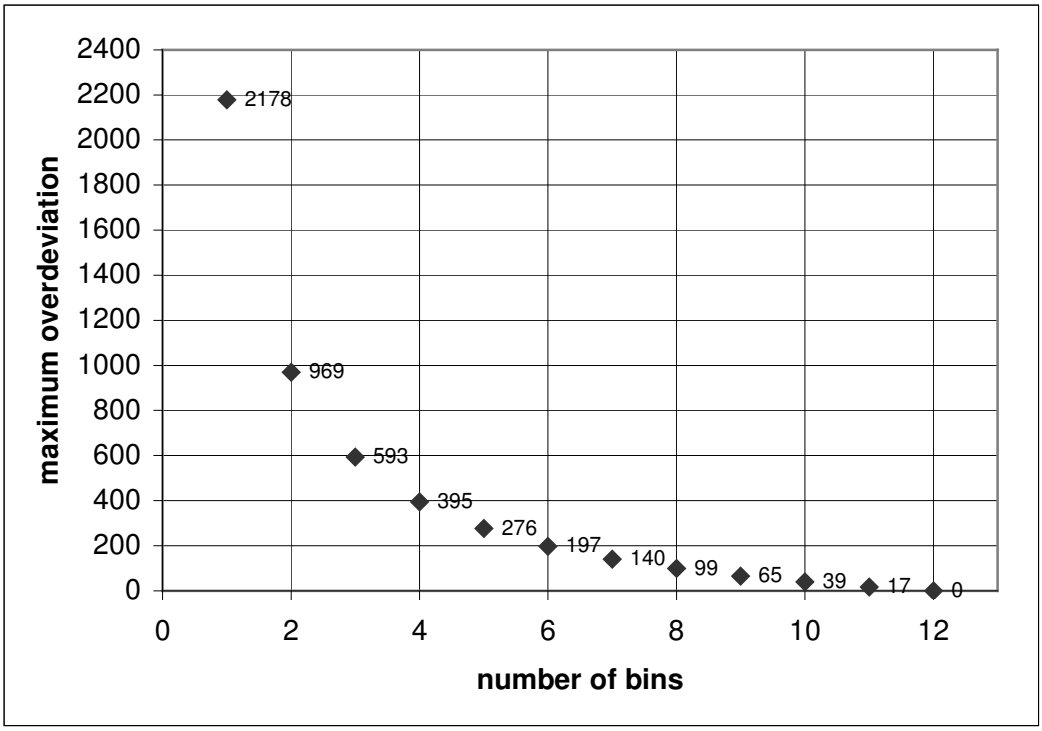

Figure 4.9 All Efficient Solutions

As can be seen from the above figure, there is no linear relationship between the maximum overdeviation and the number of bins. Hence, in Step II we have to solve MIP II for all possible number of bins, to generate the set of efficient solutions.

CHAPTER 5

COMPUTATIONAL EXPERIMENTS

In this chapter, we present the experiment environment and results of the experiments.

5.1 Design of Experiments

In our computational experiments, the class of a problem is determined by the number of items, item weights and bin capacities. 18 different problem classes were constructed and we perform our experiments with problems having 50, 75 and 100 items. Item weights are generated from uniform distribution in the ranges [1,50] and [1,100]. We use bin capacities of 100, 150 and 200. For each problem class, we generate 10 problem instances. For each problem instance, we find all efficient solutions.

The Central Processing Unit (CPU) time limit is set to 3600 seconds, that is the execution is terminated if the optimal solution is not returned in 3600 seconds.

We report average and maximum values of CPU times, lower bound and upper bound performances and the number of unsolved instances within specified termination limit.

The computational experiments are performed on Pentium II processor with 128 MB RAM. GAMS 20.2 with CPLEX solver is used to solve MIP problems. All heuristics are coded with VISUAL BASIC 6.0.

5.2 Computational Results

In this section we present the results of our computational experiments for problems I and II.

5.2.1 Problem I – Minimization of Total Overdeviation & Number of Bins

In this subsection, we present the results of our computational experiments for Problem I.

Solution Times:

The average and maximum CPU times are given in Table 5.1. Max(1) and Max(2) represent the maximum CPU time observed and Avg(1) and Avg(2) represent the average of CPU times. CPU times of unsolved problems are included in Max(1) and Avg(1), while they are excluded in Max(2) and Avg(2). The gray shaded entries mean that none of the problem instances can be solved within specified termination limit of 3600 seconds.

Table 5.1 Solution Times for Problem I (seconds)

n = 50					
c	**w_i**	**Max(1)**	**Avg(1)**	**Max(2)**	**Avg(2)**
100	[1, 50]	800	63	800	63
150	[1, 50]	4	3	4	3
200	[1, 50]	3	3	3	3
100	[1, 100]	3600	1100	242	32
150	[1, 100]	3600	800	150	30
200	[1, 100]	3600	500	500	60
n = 75					
100	[1, 50]	3600	422	1400	87
150	[1, 50]	8	4.5	8	4.5
200	[1, 50]	10	3.5	10	3.5
150	[1, 100]	3600	1218	3370	198
200	[1, 100]	3600	654	560	33
n = 100					
100	[1, 50]	3600	448	386	60
150	[1, 50]	16	7	16	7
200	[1, 50]	12	5	12	5
200	[1, 100]	3600	750	645	44

When there are 50 items and weights are generated between [1, 50], it is seen that as capacity increases, the average CPU time decreases. Max(1) and Max(2) values are the same for this weight distribution, implying that all instances are solved. Moreover, there is a clear decrease in the Avg(1) values for the problem classes where weights are generated between [1, 100], as capacity increases. However the Avg(2) values do not show such a relationship. This is

due to the exclusion of the CPU times of unsolved instances in Avg(2) values. Comparing the weight ranges, we see that, as the average weight of items increases, CPU time increases.

The difficulty of our problems arises from the binary variables x_{ij}. It is clear that, as capacity increases, we need fewer bins to assign all items. This decreases the upper value of index j, hence the number of binary variables, x_{ij}. The decrease in the number of required bins reduces the problem size, hence makes the problem simpler, i.e. CPU times decrease with increasing bin capacity. The same reason also holds for the average weight of items. As the average weight of items decrease, more items fill one bin; therefore, fewer bins are used.

When n = 75 and 100 items, we observe the same relations between bin capacity and CPU times, and item weights and CPU times. As the capacity increases CPU times decrease, and as the average weights of items increases CPU times increase.

When the weights are generated between [1, 50], the CPU time increases with an increase in the number of items. The same relation also holds when n = 75 and 100, and the weights are generated between [1, 100]. The reason is the same, that is an increase in the number of items affects the value of index i which increases the size of the problem.

The Lower Bound and Upper Bound Performances:

In Table 5.2, the average upper bound values (AVG. UB), the average lower bound values (AVG. LB) and the average optimal values (OPT) are

reported for the number of bins. Note that in the table absolute values are presented, i.e. we report average values of the bounds rather than any percentages. We prefer to use absolute values since the objective function values are too small.

Table 5.2 The Lower Bound, Upper Bound and Optimal Solution Values

AVG. UB, OPT		**n = 50**		**n=75**		**n=100**	
c	**w_i**	**UB**	**OPT**	**UB**	**OPT**	**UB**	**OPT**
100	[1, 50]	29	28	41.8	40.6	50.6	50.4
150	[1, 50]	31	31	51	50.3	39.2	38.6
200	[1, 50]	50	50	54	53.8	49.6	49
100	[1, 100]	73.7	71.7				
150	[1, 100]	58.4	54	124.7	104.4		
200	[1, 100]	50.9	48.4	72.6	68.7	88.5	80.8
AVG. LB, OPT		**LB**	**OPT**	**LB**	**OPT**	**LB**	**OPT**
100	[1, 50]	28	28	39.8	40.6	49	50.4
150	[1, 50]	31	31	50.3	50.3	38.4	38.6
200	[1, 50]	50	50	53.8	53.8	48.9	49
100	[1, 100]	65.1	71.7				
150	[1, 100]	50.4	54	95.2	104.4		
200	[1, 100]	47.1	48.4	67.3	68.7	77.7	80.8

When the number of items is 50 and weights are generated between [1, 50], we see that both the upper bound and the lower bound values are very close to the optimal solution. We also observe that when capacities are 150 and 200, the average upper bound and lower bound values are equal which means that for

all instances both bounds give the optimal solutions. When the numbers of items are 75 and 100 and weights are generated between [1, 50], the bounds are very close to the optimal solutions. When weights are generated between [1, 50], we do not need many bins, therefore the value of index *j* is small. As the number of items increases, we need extra bins. Hence, the problem becomes harder to solve.

When the weights are generated between [1, 100] it is seen that the values of the bounds are still close to the optimal solution values. We observe that the results are in line with our expectations.

When n = 75 and 100 items and weights are generated between [1, 100] the performance detoriates. However, relationships observed for n = 50 also hold here too. As the capacity increases or the number of items decreases or the average weight of items decreases, the bounds get closer to the optimal solutions.

The results reveal that the problem becomes harder as capacity decreases, and the number of items or the average weight of items increases, but the performances of our bounds are good for all problem classes.

Number of Unsolved Instances:

Table 5.3 reports the number of unsolved instances of Problem I, the total number of efficient solutions including the ones obtained either by solving MIP I and MIP II or directly from Theorem 4.4, the percentage of unsolved problems and the percentage of solutions found by Theorem 4.4.

Table 5.3 Number of Unsolved Instances of Problem I

n = 50						
c	**w_i**	**# of unsolved problems**	**# of ins. by MIP I & MIP II**	**# of efficient solutions**	**% unsolved problems**	**% of soln. by Theorem 4.4**
100	[1, 50]	0	20	130	0	84.6
150	[1, 50]	0	20	90	0	77.7
200	[1, 50]	0	20	70	0	71.4
100	[1, 100]	8	40	272	2.9	85.2
150	[1, 100]	3	20	171	1.7	88.3
200	[1, 100]	1	20	133	0.7	84.9
n = 75						
100	[1, 50]	2	30	200	1	85
150	[1, 50]	0	20	128	0	84.3
200	[1, 50]	0	20	98	0	79.5
150	[1, 100]	7	40	267	2.6	85
200	[1, 100]	3	30	190	1.6	84.2
n = 100						
100	[1, 50]	3	40	281	1.1	85.7
150	[1, 50]	0	20	164	0	87.8
200	[1, 50]	0	20	132	0	84.8
200	[1, 100]	7	40	270	2.6	85.1

The percentages of efficient solutions found by Theorem 4.4, show that on the average 85 % of all efficient solutions are found by Theorem 4.4. As the problem gets harder; that is as capacity decreases or the number of items and the average weight of items increases, this percentage also increases. That is

because, as problem gets harder we need more bins to be filled and this leads to a drastic increase in the total number of efficient solutions. However, there are not many efficient solutions until all the bins are fully loaded.

As can be seen from Table 5.3, when capacity increases, the total number of efficient solutions decreases. This is because, as capacity increases we need fewer bins to assign the items. As average weight of items increases we need more bins to be filled, hence the total number of solutions increases with increasing the average weight of items. As mentioned in Chapter 2, the ranges of efficient solutions are related with the necessary number of bins. Because of the same reason, the number of problem instances solved by MIP I and MIP II increases as the number of bins to be filled increases.

The number of unsolved problems shows that as capacity increases or the average weight of items decreases, the number of unsolved problems decreases. So, our previous conclusion, as the necessary number of bins increases, the problem gets harder to solve, holds here as well.

We also observe that, due to the same reason, as the number of items increases, the number of unsolved instances increases too.

We also investigate the performance of the upper and lower bounds on the number of bins for 150 and 200 items. In Table 5.4, the average values of these bounds and the average and maximum differences between the bounds are given.

Table 5.4 The Lower and Upper Bounds when n = 150 and 200 for Problem I

n = 150					
c	**w_i**	**AVG. LB**	**AVG. UB**	**AVG. DIF.**	**MAX. DIF**
100	[1, 50]	55.3	60	4.7	16
150	[1, 50]	55.5	57.5	2	6
200	[1, 50]	55.7	56.5	0.8	2
100	[1, 100]	80.5	103.8	23.3	83
150	[1, 100]	139	193.7	54.7	144
200	[1, 100]	92.3	102.9	10.6	47
n = 200					
100	[1, 50]	56.8	66.6	9.8	44
150	[1, 50]	54	58.1	4.1	28
200	[1, 50]	57.6	59.2	1.6	13
100	[1, 100]	116.9	145.3	28.4	116
150	[1, 100]	129.9	202.1	72.2	220
200	[1, 100]	102.7	117.3	14.6	63

When the weights are generated between [1, 50], as the capacity increases the bounds get closer. This may be explained by the fact that as the capacity increases, we need fewer bins, therefore the size of the problem reduces. Thus, the performance of our bounds improves. However, when weights are generated between [1, 100], the largest gap between the bounds is observed when the capacity is 150. Recall that we could not solve the problems with more than 100 items when the weights are generated between [1, 100] and the bin capacities are set to 100 and 150. Hence, we cannot expect a smooth relationship

for these instances. We also observe that when the weights are generated between [1, 100], the bounds are the closest when bin capacity is 200.

Computational Experiments with and without Upper Bound:

To see the effect of upper bounds on the solution speed, we perform experiments with and without upper bounds for 50 items and report the results in Table 5.5. The table gives the number of unsolved problems, number of nodes and CPU times with and without upper bounds cases.

Table 5.5 Comparison of Results with and without Upper Bound

		# of unsolved Pr.		**# of nodes**		**CPU time (s)**	
c	**w_i**	**with UB**	**w/o UB**	**with UB**	**w/o UB**	**with UB**	**w/o UB**
100	[1, 50]	0	0	90	590	63	90
150	[1, 50]	0	0	6	69	3	24
200	[1, 50]	0	0	0	27	3	11
100	[1, 100]	8	11	210	800	32	82
150	[1, 100]	3	4	181	5283	30	154
200	[1, 100]	1	2	215	775	60	79

The positive effect of the upper bounds on the solution speed is clearly seen from Table 5.5. When the weights are generated between [1, 100], the role of the upper bounds increases in the solution process. This is not surprising because when the average weight of the items increases, the problem becomes harder to solve.

For the problems where the weights are generated between [1, 100], the number of unsolved problems and the number of nodes reduce significantly when upper bounds are used. Hence, CPU times decrease drastically.

5.2.2 Problem II – Minimization of Maximum Overdeviation & Number of Bins

In this subsection, we present the results of our computational experiments for Problem II. The same notation with Problem I are used.

Solution Times:

We report the CPU times of Problem II in Table 5.6.

Table 5.6 The CPU Times of Problem II

n = 50					
c	**w_i**	**Max(1)**	**Avg(1)**	**Max(2)**	**Avg(2)**
100	[1, 50]	3600	124	1024	21.3
150	[1, 50]	4	2	4	2
200	[1, 50]	3	2	3	2
150	[1, 100]	3600	587	1014	32
200	[1, 100]	3600	326	900	25
n = 75					
100	[1, 50]	3600	488	2194	54.7
150	[1, 50]	3600	143.5	548	15.3
200	[1, 50]	46	5	46	5
200	[1, 100]	3600	638.5	2071	95.4
n = 100					
150	[1, 50]	3600	290	72	11
200	[1, 50]	152	5.3	152	5.3

The CPU times of Problem II show that this problem is harder to solve than Problem I. However all trends observed for Problem I are also valid for Problem II. For instance, the average CPU times decrease with increasing capacity for 50 items when weights are generated between [1, 50]. For 50 items when weights are generated between [1,100], Avg(1) values show the same relationship. For 75 and 100 items, the CPU times decrease with increasing capacity or decreasing average weight of items. With increasing capacity, we need less number of bins; hence, problem size decreases. As we mentioned above, Problem II is harder to solve as the model tries to make deviations of bins close to each other. Note that Problem I did not have such a balance concern.

The Lower Bound and Upper Bound Performances:

Table 5.7 reports the lower, upper bound and optimal solution values for Problem II.

Table 5.7 The Average Lower Bound, Upper Bound and Optimal Values

AVG. UB, OPT		**n = 50**		**n=75**		**n=100**	
c	**w_i**	**UB**	**OPT**	**UB**	**OPT**	**UB**	**OPT**
100	[1, 50]	128.9	128.1	162.4	161.7		
150	[1, 50]	153.2	152.9	192.5	191.9	236.7	221.2
200	[1, 50]	142.9	142.6	208.9	208.4	283.2	259.7
150	[1, 100]	241.4	227.5				
200	[1, 100]	272.7	254.9	367.3	306.3		
AVG. LB, OPT		**LB**	**OPT**	**LB**	**OPT**	**LB**	**OPT**
100	[1, 50]	127.5	128.1	160.2	161.7		
150	[1, 50]	152.4	152.9	189	191.9	220.2	221.2
200	[1, 50]	142.2	142.6	207.6	208.4	256.7	259.7
150	[1, 100]	224.2	227.5				
200	[1, 100]	251.5	254.9	302.1	306.3		

When n = 50, and weights are generated between [1, 50], as the capacity increases, the bounds get closer. This observation is valid when weights are generated between [1, 100]. We also observe that, as the number of items decreases, the bounds get closer. Although we observe improvements in the bounds with increase in the capacity or decrease in the number of items, the improvements are not significantly different from each other. This is the main difference between Problem I and Problem II. Decrease in the necessary number of bins is not as effective as that of Problem I. Because in Problem II, we assign items to the bins so that the deviation amounts over all bins are close to each

other. Hence, the number of bins required is no longer a dominant factor in determining the performance of the bounds.

When n = 100, and weights are generated between [1, 50], we do not observe the above relations. If the number of items is high, then it is more difficult to assign items to the bins with very close overdeviations. Hence for these cases, increase in capacity does not improve the performance of the bounds.

Number of Unsolved Instances:

Table 5.8 shows the number of unsolved instances for Problem II.

Table 5.8 Number of Unsolved Instances for Problem II

n = 50				
c	**w_i**	**# of unsolved problems**	**Total # of solutions**	**% unsolved problems**
100	[1, 50]	2	130	1.5
150	[1, 50]	0	90	0
200	[1, 50]	0	70	0
150	[1, 100]	15	171	8.8
200	[1, 100]	8	133	6
n = 75				
100	[1, 50]	22	200	11
150	[1, 50]	4	128	3.1
200	[1, 50]	0	98	0
200	[1, 100]	27	190	14.2
n = 100				
150	[1, 50]	10	164	6.1
200	[1, 50]	0	132	0

The percentage of the number of unsolved instances is more than that of Problem I as expected. Problem II is more difficult even when the capacity is increased because in Problem II, items should be assigned to the bins so that overdeviations are nearly equal. The other relations for Problem I, hold for Problem II.

Finally we investigate the upper and lower bound performances for 150 and 200 items. In Table 5.9, the average of these bound values and the differences between these bounds are given.

Table 5.9 The Lower and Upper Bounds when n = 150 and 200 for Problem II

n = 150					
c	**w_i**	**AVG. LB**	**AVG. UB**	**AVG. DIF.**	**MAX. DIF**
100	[1, 50]	194.4	226.7	32.3	311
150	[1, 50]	248.9	294.6	45.7	264
200	[1, 50]	281.9	345.9	64	214
100	[1, 100]	222	346.5	124.5	745
150	[1, 100]	278.2	455.3	177.1	600
200	[1, 100]	388.3	663.7	275.4	512
n = 200					
100	[1, 50]	218.5	280	61.5	410
150	[1, 50]	289.5	384.9	95.4	462
200	[1, 50]	333.6	459.3	125.7	710
100	[1, 100]	310.9	562.1	251.2	796
150	[1, 100]	317.8	607.2	289.4	849
200	[1, 100]	444	890.5	446.5	901

It can be observed from Table 5.9 that, none of the relations we observed for Problem I hold. Actually, that is an expected result because when we examined lower and upper bound performances for Problem II, there was not a smooth relationship for the problems with 100 items. When the number of items is large (more than 100 items) the problem becomes so hard that the capacity loses its significance.

To summarize, Problem II shows the same relations with Problem I, according to the solution times and the number of unsolved instances. However, when we examine lower and upper bound performances, when there are more than 100 items, the observations differ. By Theorem 4.4, Problem I shows better performance on the average because after some point we guarantee to find efficient solutions without solving the problem. However, in Problem II, we have to solve all instances. That increases the possibility of not finding an efficient point and decreases the performances for large n.

CHAPTER 6

CONCLUSIONS

In this study, we analyze a bin packing problem with multiple objectives. We aim to minimize the number of bins used and the deviations over the bin capacity. We generate two versions of this problem. We minimize the total overdeviation in the first problem and the maximum overdeviation in the second one. To the best of our knowledge, our study is the first attempt to solve the multiobjective bin packing problem, to minimize the number of bins and the deviation over the bin capacity.

We first show that the two objectives of our problems are conflicting, i.e. improvement in one detoriates the other. We also show that our problems are analogous to the parallel machine scheduling problems.

For both problems, we develop mathematical models and propose some solution procedures to generate efficient points. To enhance the efficiency of our procedures, we generate some lower and upper bounds. Lower bounds are obtained from the LP relaxations of the problems. We generate some heuristics to obtain upper bounds by using the procedures developed for the parallel machine scheduling problems.

We develop some properties of the efficient sets for both problems. For minimizing total overdeviation, by this property, we generate some of the efficient solutions trivially.

To measure the performance of the algorithms and investigate the effects of different parameters on problem difficulties, an experiment is designed. The computational results reveal that CPU time is effected by the capacity of the bins, the weight of the items and the number of the items for the problem minimizing total overdeviation. The proposed procedure provides satisfactory methods of solving problems up to 100 items when weights are generated uniformly between [1, 100] and [1, 50]. For the problem of minimizing maximum overdeviation, the proposed procedure provides satisfactory methods of solving problems up to 75 items when the weights are generated uniformly between [1, 100] and up to 100 items when the weights are generated uniformly between [1, 50]. So we can conclude that, the problem of minimizing maximum overdeviation is harder to solve.

We also perform an experiment to test the effect of upper bounds. Our experiments show that incorporation of the upper bounds drastically reduce the CPU time and the number of unsolved instances.

The computational tests also reveal that the capacity, the number of items and the weight of the items play a significiant role on the difficulty of the problems. The problem gets harder with decrease in capacity, increase in average weight and increase in the number of items. Hence, we may conclude that our

solution procedure is very effective especially for problems up to 100 items when the weights are generated uinformly between [1, 100] and [1, 50].

This study may open some new research areas. Some examples that can be studied in the future are listed below.

- Variable Bin Sizes: Instead of using bins with equal capacities, bins with different capacities can be used.
- Problem with Underdeviation: Another multi-objective problem can be to maximize the number of bins used and to minimize the underdeviation (slack of the bins).
- Problem with Underdeviation and Overdeviation: Instead of minimizing the number of bins, an alternative multi-objective problem can be to minimize the underdeviation and overdeviation of the bins.

REFERENCES

1. Azar, Y., Regev O. (2001), "On-line Bin Stretching", Theoretical Computer Science, Vol. 268, pp. 17-41.

2. Bruno, J. L., Downey J. P. (1985), "Probabilistic Bounds for Dual Bin Packing", Acta Informatica, Vol. 22, pp. 333-345.

3. Csirik, J., Frenk J. B. G., Labbe M., Zhang S. (1999), "Two Simple Algorithms for Bin Covering", Acta Cybernetica, Vol. 14, pp. 13-25.

4. Csirik, J., Johnson D. S., Kenyon C. (2001), "Better Approximation Algorithms for Bin Covering", 12th ACM-SIAM Symp. On Discrete Algorithms, Washington D.C.

5. Csirik, J., Frenk J. B. G., Galambos G., Kan A. H. G. (1991), "Probabilistic Analysis of Algorithms for Dual Bin Packing Problems", Vol. 12, pp. 189-203.

6. Coffman, E. G., Garey M. R. and Johnson D. S. (1984), “Approximation Algorithms for Bin Packing – An Updated Survey”, Algorithm Design for Computer Systems Design, Springer-Verlag, Vienna.

7. Coffman, E. G., Leung J. Y-T, Ting D.W. (1978), “Bin Packing: Maximizing the Number of Pieces Packed”, Acta Informatica, Vol. 9, pp. 263-271.

8. Eilon, S., Christofides N. (1971), “The Loading Problem”, Management Science, Vol. 17, pp. 259-267.

9. Flezar, K., Hindi K. S. (2002), “New Heuristics for One Dimensional Bin Packing”, Computers and Operations Research, Vol. 29, pp. 821-839.

10. Foster, D. P., Vohra V. R. (1989), “Probabilistic Analysis of A Heuristics For The Dual Bin Packing Problem”, Information Processing Letters, Vol. 31, pp. 287-290.

11. Friesen, D. K., Langston M. A. (1986), “Variable Sized Bin Packing”, SIAM Journal of Computing, Vol.15, pp. 222-229.

12. Garey, M.R. and Johnson, D.S. (1979), "A Guide to the Theory of NP-Completeness", Computers and Intractability, A Series of Books in the Mathematical Sciences, W.H. Freeman and Company.

13. Graham, R. L., (1969), "Bounds on Multiprocessing Timing Anomalies", SIAM J. Applied Mathematics, Vol.17, pp. 416-429.

14. Gupta, J. N. D., Ho J. C. (1999), "A New Heuristic Algorithm for The One Dimensional Bin Packing Problem", Production Planning and Control, Vol.10, pp. 598-603.

15. Hansen, P., Mladenovic N. (1997), "Variable Neighbourhood Search", Computers and Operations Research, Vol. 24, pp. 97-100.

16. Hung, M. S., Brown J. R. (1978), "An Algorithm for A Class of Loading Problems", Naval Research Logistics Quarterly, Vol. 25, pp. 289-297.

17. Johnson, D. S., Demers A., Ullman J. D., Garey M. R., and Graham R.L.(1974), "Worst-case performance bounds for simple one-dimensional packing algorithms", SIAM Journal on Computing, Vol.3, pp. 299-325.

18. Johnson, D. S. (1974), "Fast Algorithms for Bin Packing", Journal of Computer System Sciences, Vol.8, pp. 272-314.

19. Labbe, M., Laporte G., Martello S. (1995), “An Exact Algorithms for the Dual Bin Packing Problem”, Operations Res. Letters, Vol.17, pp.9-18.

20. Martello, S. and Toth, P.(1990), “Knapsack Problems: Algorithms and Computer Implementations”, John Wiley and Sons, Chichester, England.

21. Scholl, A., Klein R., Jurgens C. (1997), “BISON: A Fast Hybrid Procedure for Exactly Solving The One Dimensional Bin Packing Problem”, Computers and Operations Research, Vol. 24, pp. 627-645.

22. Xing, W. (2002), “A Bin Packing Problem with Over Sized Items”, Operations Research Letters, Vol. 30, pp. 83-88.

23. Zhang, G. (1997), “A New Version of On-line Variable Sized Bin Packing”, Discrete Applied Mathematics, Vol. 72, pp. 193-197.

24. Zhang G. (2001), “An On-line Bin Batching Problem”, Discrete Applied Mathematics, Vol. 108, pp. 329-333.