BICRITERIA BIN PACKING PROBLEM WITH DEVIATION BASED OBJECTIVES

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

BY

AYLA ÖYLEK

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

DECEMBER 2015

Approval of the thesis:

BICRITERIA BIN PACKING PROBLEM
WITH
DEVIATION BASED OBJECTIVES

submitted by **AYLA ÖYLEK** in partial fulfillment of the requirements for the degree of **Master of Science in Industrial Engineering Department, Middle East Technical University** by,

Prof. Dr. M. Gülbin Dural Ünver _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Murat Köksalan _____
Head of Department, **Industrial Engineering**

Assoc. Prof. Dr. Esra Karasakal _____
Supervisor, **Industrial Engineering Dept., METU**

Prof. Dr. Meral Azizoğlu _____
Co-Supervisor, **Industrial Engineering Dept., METU**

Examining Committee Members:

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

Assoc. Prof. Dr. Esra Karasakal _____
Industrial Engineering Dept., METU

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

Assist. Prof. Dr. Sakine Batun _____
Industrial Engineering Dept., METU

Assist. Prof. Dr. Özlem Karsu _____
Industrial Engineering Dept., Bilkent University

Date: 04.12.2015

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

Name, Last name : AYLA ÖYLEK

Signature :

ABSTRACT

BICRITERIA BIN PACKING PROBLEM WITH DEVIATION BASED OBJECTIVES

Öylek, Ayla

M. S., Department of Industrial Engineering

Supervisor : Assoc. Prof. Dr. Esra Karasakal

Co-Supervisor : Prof. Dr. Meral Azizoğlu

December 2015, 78 pages

In this thesis, two bicriteria bin packing problems are addressed. Bin packing problem is an NP-hard combinatorial optimization problem. Items with different weights are packed into bins with limited capacity in order to minimize the required number of bins. Objectives of the first problem are minimizing the number of bins and minimizing the total overdeviation. In the second problem, minimization of the number of bins and minimization of the maximum overdeviation are two conflicting objectives. For the solutions of the problems mixed integer linear programming models are formulated and used to find all nondominated objective vectors. The upper bounds and lower bounds are developed on the objective function values and bounds are incorporated into the mathematical models to increase the solution efficiency of the models. Computational results show that the problem with up to 100 items could be solved for high capacity bins. The problems with up to 75 items can be solved when the capacity is low.

Keywords: Bin Packing, Total Overdeviation, Maximum Overdeviation, Multiobjective Optimization, Nondominated Objective Vectors

ÖZ

İKİ KRİTERLİ KUTU PAKETLEME PROBLEMLERİ

Öylek, Ayla
Yüksek Lisans, Endüstri Mühendisliği Bölümü
Tez Yöneticisi : Doç. Dr. Esra Karasakal
Ortak Tez Yöneticisi : Prof. Dr. Meral Azizoğlu

Aralık 2015, 78 sayfa

Bu çalışmada, iki kriterli iki kutu paketleme problemini ele aldık. Kutu paketleme problemi çözümü polinom zamanlı olmayan (NP) kombinatoriyal bir problemdir. Farklı ağırlıktaki nesnelerin en az kutu kaplayacak şekilde sınırlı kapasiteli kutulara yerleştirilmesidir. İlk problemin birbiriyle çelişen amaç fonksiyonları kutu sayısının ve kutu kapasitesinden toplam sapmanın en azlanmasıdır. Kutu sayısının ve maksimum sapmanın en azlanması ikinci problemin birbiriyle çelişen amaç fonksiyonlarıdır.

Problemi tam sayılı karmaşık model olarak formüle ettik ve etkin çözümler elde eden kesin yöntemler kullandık. Çözümlerin kalitesini arttırmak için alt ve üst sınırlar önerdik.

Deneysel sonuçlarımız 100 nesneye kadar olan problemlerin yüksek kapasiteli kutular için çözülebildiğini, düşük kapasiteli kutular için 75 nesneye kadar olan problemlerin çözülebileceğini gösterdi.

Anahtar Kelimeler: Kutu Paketleme, Toplam Sapma, Maksimum Sapma, Çok Amaçlı Optimizasyon, Etkin Çözümler

ACKNOWLEDGMENTS

First of all, I would like to express my deepest appreciation to my supervisor Assoc. Prof. Dr. Esra Karasakal and my co-supervisor Prof. Dr. Meral Azizoğlu for their guidance, their valuable insight and their important contributions. It has been a great privilege to have been mentored by them.

Then, I should reveal that, being with me in all this exhaustive period and providing endless support, my family deserves the best wishes. I offer sincere thanks to my parents, Ayşe and Yakup Öylek, and my sister Kadriye Öylek for their love and patience.

I would like to express my deep sense of gratitude to Cem Tüfekci for his great encouragment and support throughout this study. I also would like to thank Cem Kundakçı and Yusuf Kaplan for supporting me. Without their patience, completing this work would not be possible.

I also would like to express my special thanks to the special person in my life; Erdem Çolak for his great technical support and being with me; giving his love and making me feel better in every instance during this study. I am extremely fortunate for having his love and support at the most hopeless moments. Without him, completing this work would not be possible.

Lastly, I would like to thank all the members of the examining committee and all the people I had the chance to know in the Department of Industrial Engineering.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

The bin packing problem is one of the well-recognized problems of the Operational Research (OR) literature. The problem decides on the assignment of the items to the bins so as to minimize the prespecified objective function. The items have defined capacity usages and the bins have defined capacity availabilities.

The objective functions used in the Bin Packing Problems (BPP) define the type of the problem. The classical bin packing problems minimize the number of bins used to pack all items without exceeding any bin capacity. The maximum cardinality BPP takes fixed number of bins and maximizes the number of items packed without exceeding any bin capacity. The bin covering problem maximizes the number of bins used by exceeding the capacity of each bin.

The BPPs in general and the classical BPP in particular have been taken the attention of many researchers for many years. This is due to their theoretical challenge and practical importance. They are theoretically challenging as they are shown to be hard-to-solve problems. They are practically important as they are directly and indirectly applied to many practical situations.

Logistics sector is an area where the BPPs find their direct application. Assigning items into trucks, cargo airplanes, ships while minimizing the number of trucks, cargo airplanes and ships are examples for the classical BPPs.

The BPPs find their indirect applications in manufacturing environments. The well-known cutting stock problem is a BPP where the items are assigned to the sheets so as to minimize the waste, i.e., the number of sheets used. Parallel machine scheduling problem is a BPP where the jobs (items) of specified processing times

(capacity requirements) are to be assigned to the machines (bins) of specified capacities. The classical Assembly Line Balancing Problem is another notable application area. The tasks (items) of specified task times (capacity requirements) are to be assigned to the workstations (bins) without exceeding the cycle time (bin capacity).The BPPs studied in the literature either minimize/maximize the number of bins used for a given number of items or maximize the number of items for a given number of bins. The capacities are set tight in those studies; they either play a role of upper bounds or lower bounds.

In this study, it is assumed that the bin capacities can be violated with some penalty, i.e., the capacities are soft –but not hard- constraints. The associated objective functions minimize a function of the deviations around the preset capacities, like minimizing total overdeviation or minimizing maximum overdeviation.

Dealing with bin capacity violations, i.e., deviations might be an important concern for the cases where the bins are shared by some other parties. The deviations will then be represented the amount that the user wants to increase his/her capacity share. The share of the user can be increased provided that proper negotiations are done. The proper negotiations might be done provided that the deviations around the preset capacities are found properly.

The total/maximum overdeviation problem finds its application in the assembly line. The cycle time defines the production rate of an assembly line. An increase in cycle time, capacity in BPP terminology, would lead to a reduction in the number of workstations, bins in BPP. A decision maker might accept an increase in the cycle time; thereby decrease in the production rate, provided that such an increase leads to a reduction in the number of workstations, thereby resource usages. The natural problem becomes to catch the trade-off between the deviation from the capacity and the number of workstations (Naderi et al. [27]).

In this study, a bicriteria problem that considers minimizing total number of bins used and minimizing total/maximum overdeviation around the bin capacities is studied. Our aim is to generate all nondominated objective vectors. To the best of our knowledge, our study is the first attempt for deviation based objectives, in BPPs.

The rest of the thesis is organized as follows: Chapter 2 presents a literature review on the bin packing problems then gives brief information about the types of bin packing problems and classifies the literature review according to these types. In Chapter 3, our problems are defined, their mathematical models are given and some properties of the solutions are presented. The notation used and some definitions are given in this chapter. Chapter 4 discusses our solution approaches to generate all nondominated objective vectors along with the lower bounds and upper bounds. Chapter 5 reports on our computational experiments and discusses its results. Chapter 6 gives a brief conclusion of our study.

CHAPTER 2

LITERATURE REVIEW

This section reviews the literature on the bin packing problems. The single criterion bin packing problems are reported first and then the multi-criteria bin packing problems are reviewed.

Online and offline are two different categories of bin packing according to the information about the input. Items arrive dynamically in online bin packing problems where all items are known at the beginning in offline type bin packing problems. Our study focuses on offline type bin packing problems.

The dimension of the items and bins is another important criterion for classification of the bin packing problem. Single or multiple dimensions can be used to pack items to bins. In one-dimensional bin packing problems, the weight of an item is selected mostly as the single property. In multi-dimensional bin packing problems, other properties like weight, height, length, width can be used for packing. Our problem is one-dimensional bin packing problem and the weight of items is used in our problem.

The bin packing problems are also categorized according to bin capacity. There are single sized and variable sized problems. In single sized problems, bins have equal capacities wherein variable sized version bins have different capacities. In our problem, all bins have equal capacities.

2.1. Single Criterion Bin Packing Problems (BPPs)

The basic single criterion BPPs are of four types: Classical BPP, Maximum Cardinality BPP, Bin Covering Problem and Modified BPP. The problems assume that there are n items of weight w_i for item i and m bins of identical capacity C.

The mathematical model is first given and then the related literature of each basic problem is reviewed.

2.1.1. Classical Bin Packing Problems

The classical bin packing problem minimizes the number of bins used to pack all items without exceeding the bin capacities.

The decision variables associate with the assignment of the items to the bins is stated as follows:

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

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

Constraint set (1) states that each item is assigned to one bin

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

Constraint set (2) ensures that the capacity of each bin used is not exceeded

$$\sum_i w_i x_{ij} \leq \text{c} y_j, \qquad \forall \text{ j} \qquad (2)$$

The objective function is as expressed below

$$\text{Minimize} \quad \sum_{j} y_j \qquad (3)$$

The classical BPP is shown to be strongly NP-hard by Garey and Johnson [12]. There are numerous studies on the classical BPPs. The studies are reviewed according to the solution procedures used. Firstly, the literature review about approximation algorithms is given and then literature review about exact algorithms of classical bin packing problem is given.

2.1.1.1. Approximation Algorithms

Firstly, well-known bin packing heuristics such as next fit, first fit and best fit algorithm are given then recent studies are introduced.

Johnson [17] developed a number of simple one-dimensional bin packing algorithms namely next fit, first fit, best fit and worst fit heuristics.

Next fit is the simplest approximation approach. It tries to pack arbitrarily ordered items to bins. The approach packs the item into the current bin, if the item does not fit then it creates a new bin and insert the item into the new bin.

First fit is a better algorithm than next fit. The algorithm tries to pack an item into the first available bin; if the item does not fit into any bin then it creates a new bin and inserts the item into the new bin.

Best fit is another well-known approximation algorithm obtained from first fit algorithm. Best fit algorithm calculates remaining capacity of all bins. It tries to pack an item into a feasible bin having smallest remaining capacity. If the item does not fit any bin then it creates a new bin and inserts the item into the new bin.

Worst fit is similar to best fit algorithm. Worst fit algorithm calculates remaining capacity of all bins then it tries to pack an item into a feasible bin having largest

remaining capacity. If the item does not fit any bin then it creates a new bin and inserts the item into the new bin.

Johnson et al. [18] examined the performance of simple algorithms and they developed next fit decreasing (NFD), first fit decreasing (FFD), best fit decreasing (BFD) and worst fit decreasing (WFD) algorithms. These four algorithms are similar to the algorithms of Johnson [17]. NFD, FFD, BFD and WFD algorithms start with sorting the items in nonincreasing order of their weights. Other steps of the algorithms are similar to the steps of Johnson[17] ‘s algorithms.

Mladenovic and Hansen [26] developed an effective meta-heuristic by variable neighborhood search method (VNS). The heuristic is based upon the strategy of using more than one neighborhood structure and of changing those structures systematically during the local search. VNS explores distant neighborhoods from the current solution and to jump to a new one if and only if an improvement was made. A local search routine is also applied to get from new solutions to local optima.

Gupta and Ho [15] presented a new algorithm to bin packing problem; minimum bin slack heuristic (MBS). At each step, the algorithm tries to find a set of items to pack that fits the bin capacity as much as possible. All possible subsets of items are tested to pack in order to use bin capacity better. Result part of the study shows that MBS is better than FFD and BFD in terms of solution quality.

Fleszar and Hindi [10] enhanced the approach of Gupta and Ho and proposed four new algorithms based on MBS. The first algorithm, MBS’, is modified version of the original algorithm. Before the search procedure is started, an item is chosen and fixed in the bin. Other three heuristics are based on MBS’. In relaxed MBS’, the authors accept some packing with positive slack. The third one is Perturbation MBS, starting from an initial solution, the heuristic finds a new solution by perturbing the current one. Sampling MBS’ algorithm is the last one; the algorithm applies MBS’ several times by changing the order of unassigned items. Results show that MBS based heuristics give good results in reasonably short solution times.

Ross et al. [32] suggested an approach based on messy genetic algorithm in order to solve one-dimensional bin packing problem. Hyper-heuristics uses some combination of well-known heuristics to find a better solution to problems. The approach applied four basic bin packing heuristics and four improvement heuristics trying to pack an item in any open bins rather than in a new bin. GA-based algorithm applied to a large set of benchmark problems. Experiments showed that the algorithm found the optimal solution for nearly 80% of them and for others found a solution very close to optimal.

Alvim et al. [1] described a hybrid improvement procedure for the bin packing problem. The approach is based on feasible solutions to dual bin packing problem using fixed number of bins. Progressive increase in the number of bins is used by a possibly feasible solution. Reduction techniques lower and upper bounds, an algorithm using lower bounding strategies, load redistribution and an improvement process utilizing tabu search are used in the procedure. Experiments with benchmark problem sets showed that the procedure improved the best-known solutions for many of the benchmark instances and found the largest number of optimal solutions with respect to the other available approximate algorithms. The hybrid algorithm outperforms any known heuristics.

Singh and Gupta [34] proposed an approach combining two heuristics for the one-dimensional bin packing problem. The authors applied a hybrid steady-state grouping genetic algorithm to bin packing problem and developed an improved version of Perturbation MBS heuristic [10]. The approach combines hybrid steady-state grouping genetic algorithm and improved Perturbation MBS heuristic. The hybrid algorithm combines steady state grouping genetic algorithm and improved better fit heuristic. Improved better fit heuristic allocates unassigned items to the bins after the application of the genetic algorithm.

Stawowy [35] presented a simple, non-specialized and non-hybridized evolutionary based heuristic to the bin packing problem. The algorithm does not have a pure evolutionary strategy. Modified encoding scheme for the permutation with separators, the concept of separators' movements during mutation, and separators' removal for problem size reduction are used to solve the one-dimensional bin

packing problem. Computational experiments with benchmark problem sets showed that the procedure is comparable to much more complicated algorithms.

Loh et al. [23] presented a new way of one-dimensional bin packing problem with weight annealing procedure. Proposed algorithm is easy to understand and straightforward. The algorithm finds an initial solution by use of first fit decreasing heuristic and calculates bin load and residual capacity for each bin. In improvement phase, exchanging operations are derived between all possible pairs of bins. Experiments of benchmark instances showed that procedure found high-quality solutions within very low computing times and found new optimal solutions.

Kim and Wy [20] introduced a new packing algorithm which considers the use of the last two fit (L2F) augmentation to next fit decreasing, first fit decreasing, and best fit decreasing algorithms. L2F augmentation improves the solution of original algorithms checking to pack an additional item into the bins or to replace an item with a pair of unpacked items whose total weights are larger than the item. The authors studied on four simple algorithms; NFD_L2F, FFD_L2F, BBB_FFD_L2F, and BFD_L2F. Computational results showed that the algorithm improved the solutions of the benchmark problem sets.

Pérez et al. [30] described a hybrid algorithm in order to find the optimal solution to one dimensional bin packing problem. Hybrid algorithm is based on a heuristic and a mathematical model. Heuristic method uses FFD_L2F algorithm first, if the number of bins is equal to lower bound then the algorithm terminates, otherwise an exact method is used. Lower bound is derived using a metaheuristic and the mathematical model is based on flow arcs technique introduced by de Carvalho [8] called as Valerio model. Valerio model is modified by defining new constraint with the lower bound. Computational results showed that the proposed algorithm finds the optimal solution for all instances using less time than Valerio model.

2.1.1.2. Exact Algorithms

In this subsection, the exact algorithms developed for solving Classical BPPs are introduced.

Eilon and Christofides [9] presented a branch and bound procedure with a simple depth-first search and a best-fit decreasing branching strategy. In each node, alternative subproblems are generated by assigning the selected item to all initialized bins in increasing order of their residual capacities if it fits. If there is no available bin, a new bin is initialized by assigning the current item to it.

Hung and Brown [16] proposed a branch and bound algorithm for the generalization of the BPP with different bin capacities. The algorithm performs a branching strategy similar to that in the procedure of Eilon and Christofides [9]. The algorithm is based on a characterization of equivalent assignments, thereby reducing the number of explored nodes. Results of the study show that only small-sized problem instances can be solved.

Martello and Toth [24] developed an algorithm, MTP, based on a "first fit decreasing" branching strategy. The items are indexed in nonincreasing order of their weights and MTP applies a reduction algorithm. 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. Computational results indicate that MTP algorithm is the most effective branch and bound procedure.

Scholl et al. [33] proposed a fast hybrid procedure BISON for solving the classical BPP. BISON composed of different known and new bound arguments and reduction procedures, several heuristics, and a branch and bound procedure. BISON calculated lower bound and upper bound first if they are equal the procedure terminates, if they are not equal, the procedure continues with tabu search. BISON tries to find a feasible solution with tabu search. After tabu search, if lower and upper bounds are still not equal then an algorithm that uses a depth-first search branch and bound procedure is implemented.

Carvalho [8] studied the one-dimensional bin packing problem. They make use of its similarity with the cutting stock problem and present an exact solution algorithm to the cutting stock problem which actually solves the application of it; the bin packing problem (the demand for each item –stock size- is 1). At each node in a branch and bound tree subproblems with different structures are produced. In order to overcome this difficulty an arc flow formulation is introduced which includes a set of flow constraints and set of constraints to ensure demand is satisfied. The arc flow formulation allows column generation at any node in the tree. The subproblem generates a set of arcs which mean cutting patterns in cutting stock problem (or set of items to fit in a bin in the bin-packing problem). They also make use of LP relaxation to define lower bounds at nodes and generate attractive columns.

2.1.2. Maximum Cardinality Bin Packing Problem

The maximum cardinality BPP maximizes the number of items packed without exceeding the capacity of a prespecified number of bins (Labbé et al. [22]).

The decision variable associated with the assignment of the items to the bins is stated as follows

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

The mathematical formulation of the problem is as stated below:

$$\text{Maximize} \quad \sum_i \sum_j x_{ij} \tag{4}$$

$$\text{s.t.} \quad \sum_i w_i x_{ij} \leq c \qquad \forall j \ (j=1,\ldots,m) \tag{5}$$

$$\sum_j x_{ij} \leq 1 \qquad \forall i \ (i=1,\ldots,n) \tag{6}$$

$$x_{ij} = 0 \text{ or } 1 \qquad \forall \ i \text{ and } j \ (i=1,\ldots,n), \ (j=1,\ldots,m) \tag{7}$$

The objective function expressed in (4) maximizes the number of items packed. The constraint set (5) ensures that the capacity of each bin is not exceeded. Each item is assigned to at most one bin as stated by constraint sets (6) and (7).

2.1.2.1. Approximation Algorithms

Coffman et al. [6] analyzed very fast heuristics for maximum cardinality type bin packing problem. They developed an algorithm, First- Fit Increasing (FFI) algorithm and analyzed its running time and performance. The algorithm sorts the items according to the increasing order of their weights, and an item is placed into the lowest indexed bin which it fits. The algorithm tries to find a maximum subset of smaller pieces.

Foster and Vohra [11] presented the probabilistic analysis of First- Fit Increasing heuristic for the maximum cardinality bin packing problem. Results of the study indicate that for the independent and identically distributed items, the relative error of the first fit increasing heuristic approaches to zero where the number of items approaches to infinity.

Peeters and Degraeve [29] studied on branch and price algorithms in order to solve the maximum cardinality bin packing problem and the bin covering problem. The authors introduced a new formulation for both problems. This formulation can be solved with column generation and tight upper bounds are derived from the LP relaxation of this formulation. They introduced a branch and price algorithm for both problems. The branch-and-price algorithm is applied whenever the upper bound derived from the LP relaxation differs from the heuristics. Experimental results showed that mostly the upper bound found by LP relaxation is equal to the optimal solution. The algorithm can solve large instances and difficult data sets.

Labbé, Laporte and Martello [22] developed an algorithm for maximum cardinality bin packing problem. The objective of the problem is to maximize the number of items packed bins without exceeding bin capacities. The algorithms is a heuristic approach using simple bin packing heuristics of the Johnson et al. [18]. The authors

described a reduction procedure for the heuristic approach. Tight upper bounds are derived. The algorithm guarantees a feasible solution and computational results showed that algorithm finds an optimal solution within very low computing times.

2.1.3. Bin Covering Problems

The bin covering problem assumes that there are unlimited number of bins and maximizes the number of bins used. The problem takes the capacity of each bin as a lower bound on the total weight assigned to that bin. (Chen and Yao [5])

The decision variables associate with the assignment of the items to the bins are stated as follows

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

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

The mathematical formulation of the problem is as stated below:

Maximize $\sum_j y_j$ (8)

s.t. $\sum_j x_{ij} \leq 1$ $\forall$ I (9)

$\sum_i w_i x_{ij} \geq c y_j$ $\forall$ j (10)

$x_{ij} = 0$ or 1 $\forall$ i and j (11)

$y_j = 0$ or 1 $\forall$ j (12)

The objective unction expressed in (8) maximizes the number of bins used. The constraint sets (9) and (11) ensure that each item is assigned to at most one bin. Constraint set (10) states the lower limit on the capacity usage.

In this section, the approximation algorithms are reviewed first and then a few exact algorithms are presented to solve bin covering problems.

2.1.3.1. Approximation Algorithms

Bruno and Downey [2] applied First-Fit Increasing heuristic for bin covering problem. Item sizes are chosen uniformly. The authors showed that 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.

Csirik et al. [7] developed two simple algorithms for bin covering problems; Heuristic Simple (SI) and Improved Heuristic Simple (ISI). Items are sorted in nonincreasing order with respect to their weights and first k items are packed where the sum of the weights up to (k+1)th item is greater than or equal to one in the heuristic simple algorithm. Algorithm tries to fill slack of the bin by adding items from the end of the list. Improved Simple Heuristic (ISI) is the improved version of Heuristic Simple. The heuristic divides the items into three groups according to their weights. The algorithm compares the item subsets from different groups and selects the best combination. The authors aimed to assign larger items to empty bins so as to use more bins.

2.1.3.2. Exact Algorithms

Labbé et al. [21] studied on an exact algorithm for the bin covering problem. They presented some reduction criteria, upper bounds and an enumerative algorithm. The authors developed a depth first branch and bound algorithm. At the first step of the algorithm, two reduction criteria are applied and an upper bound is derived. A lower bound is computed and if the upper bound is equal to lower bound procedure terminates. The branching strategy of the algorithm is based on finding more promising nodes. Experimental results of the study showed that the combined effect

of the reduction criteria and the upper bounds made a hard problem relatively easy to solve one, in most cases

Chen and Yao [5] study the bin covering problem which is a dual problem of the bin packing problem. They formulate the problem as a set partitioning problem and develop a branch and bound strategy using column generation. In order to overcome the difficulty of subproblems with different structures, they use the idea of Carvalho[8] and use a flow model. At the branch and bound tree, they also use the LP relaxation method to either to continue the depth-first branching or fathoming a node. They introduce some criterion to produce attractive and feasible paths (the columns to be added) which is the main difference of the study from Carvalho[8].

2.1.4. Modified Bin Packing Problems

Brusco et al. [4] described a variant of the classical bin packing problem as modified bin-packing problem. The modified bin packing problem decides on the assignment of items to the fixed number of bins of unlimited capacity so as to minimize the squared deviation of the bins from the average weight.

The decision variable associate with the assignment of the items to the bins is stated as follows

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

The mathematical formulation of the problem is as stated below:

$$\text{Minimize} \quad \sum_j [\, T - \sum_i x_{ij} w_i]^2 \qquad (13)$$

$$\text{s.t.} \quad \sum_j x_{ij} = 1 \qquad \forall \text{ i} \qquad (14)$$

$$x_{ij} = 0 \text{ or } 1 \qquad \forall \text{ i and j} \qquad (15)$$

The objective function (13) gives the total squared deviation around an average weight $T = \sum_i w_i / m$.

The constraint sets (14) and (15) ensure that each item is assigned to exactly one bin.

Rao and Iyengar [31] studied a different version of classical bin packing problem; the modified bin packing problem. Bins have unlimited capacity and there is fixed number of bins available. The objective of the problem is to minimize the sum of squared deviation between the average weight of a bin calculated by dividing the total weight of items to the number of bins and sums of the item weights in that bin. The authors proposed an algorithm based on simulated annealing. The algorithm starts from a randomly generated initial solution and uses a completely random neighborhood search procedure. Experimental results showed that the algorithm is much better than any of the heuristic methods of bin packing in terms of solution quality.

Brusco, Thompson and Jacobs [4] enhanced Rao and Iyengar's algorithm and presented a heuristic based on simulated annealing for modified bin packing problem. Their procedure used a morph-based search procedure in order to find better allocation. Rao and Iyengar's neighborhood search procedure evaluates many interchanges that have little chance of improving the objective function. The neighborhood search procedure of Brusco et al. [4] evaluates interchanges that are more likely to improve the objective function by limiting such interchanges to similarly-sized items. Results of the study indicate that the morphing process improves the solution of simulated annealing heuristics for these problems.

Brusco, Köhn and Steinley [3] suggested a new way to solve modified bin packing problem. They combined minimax bin packing problem and modified bin packing problem. The objective function of the problem is minimizing the maximum sum of the weights within each bin. The authors formulated a mixed zero-one integer linear programming model and developed a heuristic procedure based on the simulated annealing algorithm. Results showed that the simulated annealing heuristic generally provided better solutions than commercial mathematical programming software package solution to the mathematical formulation.

2.2. Multi-Criteria Bin Packing Problems

The majority of the BPP studies consider single criterion problems. The studies on the multicriteria BPPs are scarce and of relatively recent origin. The most noteworthy of these studies are due to Geiger [13], Mezghani et al.[25], Naderi et al. [27], and Patel et al. [28].

Geiger [13] proposed a heuristic approximation approach for the multi-objective bin packing problem. Minimizing the number of bins and minimizing the heterogeneousness of the elements in each bin are two conflicting objectives of the problem. Geiger presents a heuristic approach using modified best fit algorithm based on the principle of the conventional method. The approach computes an approximation of the set of efficient solutions when controls the heterogeneousness of the bins. His computational results compared with benchmark instances and random-fit algorithm showed the applicability of the heuristic approach to the problem.

Mezghani et al. [25] considered manager's preferences in the multi-objective bin packing problem and proposed a goal programming model for the problem where the objectives are minimizing the number of bins and minimizing the conflict between items among the bin. Goal programming model includes satisfaction function to integrate different types of manager's preferences.

Naderi et al. [27] developed a local search-based heuristic for a real case study of multi-objective bin packing problem. They formulated the problem as a mixed integer linear programming model and proposed a heuristic procedure for large sized instances of the problem. The objectives of the car manufacturer are related to improving logistic activities, which are to minimize the number of required transportation vehicles and to minimize the maximum workload difference among the transportation vehicles. Results of the model and the procedure showed that the heuristic procedure outperforms the mixed integer linear programming model in large sized instances.

Patel et al. [28] suggested a memetic algorithm for the one-dimensional multi-objective bin packing problem. The algorithm used local search on each chromosome so the algorithm is guaranteed to give near optimal solutions. The first objective is minimizing the number of bins and the second objective is maximizing total profit. The memetic algorithm calculates the fitness function first, selects the bin of items according to the fitness function, apply crossover to items and apply local search method in order to improve the solution.

CHAPTER 3

BICRITERIA BIN PACKING PROBLEM

In this chapter, we define two bicriteria bin packing problems. The first problem considers the number of bins and the total overdeviation as two objectives. The objectives of the second problem are the minimization of the number of bins and minimization of maximum overdeviation. Then, the associated mathematical models are given.

The section is organized as follows: Section 3.1 defines the problem and some basics are given in Section 3.2 In Section 3.3 the mathematical model for the problems are given and Section 3.4 discuss a property associated with efficient solutions.

3.1. Problem Definition

In the classical bin packing problem, given the weights of the items and bin capacity, the items are packed into a minimum number of bins without exceeding the bin capacity.

Our study allows violation of the bin capacity and penalizes the amount of overdeviation. The overdeviation of a bin is the total weight of items in the bin minus the capacity of the bin.

We first study the problem of minimizing the number of bins and minimizing the total overdeviation, and then consider the problem of minimizing the number of bins and minimizing the maximum overdeviation. Total overdeviation is described as the sum of overdeviation of all of the bins. Maximum overdeviation is the maximum

overdeviation among all bins. Overdeviation, total overdeviation and maximum overdeviation are calculated as;

Overdeviation of bin j = d_j = 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 = $\sum_j d_j = \sum_j$ Max {0, $\sum_i w_i\ x_{ij} - c$}

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

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

The problem assigns n items into m bins. Item i (i =1,2,…,n) has a weight of w_i units and the capacity of each bin j is c units.

As in Karasakal et al. [19] the problem of minimizing total overdeviation is defined as Problem I and problem of minimizing maximum overdeviation is defined as Problem II.

We first solve the number of bins problem to find the minimum number of bins. We then find the total and the maximum overdeviation. Starting from the minimum number of bins calculated, the number of bins is decreased by one and the total / the maximum overdeviation is found. As the number of bins decreases, the total / the maximum overdeviation increases, therefore, they are conflicting. We find the set of nondominated objective vectors.

3.2. Some Definitions

A multi-criteria problem with p objectives can be defined as follows:

Min $\{Z_1(x) , Z_2(x),\ldots,Z_p(x)\}$

subject to

$x \in X$

where

$Z_i(x)$: value of the objective i at x

x : decision vector

X : feasible solution decision space

A solution $x \in X$ is said to be efficient if there is no other x' such that :

$Z_i(x) \leq Z_i(x')$ $\forall$ i and

$Z_k(x) < Z_k(x')$ for at least one k.

$Z(x) = (Z_1(x) , Z_2(x),\ldots,Z_p(x))$ is the image of x in the objective function space.

If x is efficient then $Z(x)$ is a nondominated objective vector.

In this study, all nondominated objective vectors, and an efficient solution corresponding to each nondominated objective vector are generated.

3.3. Mathematical Model

Recall that x_{ij} is the main decision variable that is defined as

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

We let

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

Problem I :

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

$$\text{Minimize} \quad \sum_j y_j \tag{3.1}$$

$$\text{Minimize} \quad \sum_j d_j \tag{3.2}$$

The constraints of the problem are explained next.

Each item should be assigned to exactly one bin.

$$\sum_j x_{ij} = 1, \qquad \forall \text{ i} \tag{3.3}$$

If an item is assigned to a bin, then the bin should be opened.

$$x_{ij} \leq y_j, \qquad \forall \text{ i and j} \tag{3.4}$$

Constraint (3.5) gives the overdeviation due to bin capacity;

$$\sum_i w_i x_{ij} - d_j \leq c y_j, \qquad \forall \text{ j} \qquad (3.5)$$

The overdeviation of a bin should be greater than or equal to 0.

$$d_j \geq 0 \qquad \forall \text{ j} \qquad (3.6)$$

x_{ij} values are binary.

$$x_{ij} = 0, 1 \qquad \forall \text{ i and j} \qquad (3.7)$$

y_j values are binary. Note that y_j takes value 1 if x_{ij} is 1.

$$y_j = 0, 1 \qquad \forall \text{ j} \qquad (3.8)$$

Problem II:

Minimizing the number of bins used and minimizing the maximum overdeviation are two objectives for the second problem. The objectives are expressed as:

$$\text{Minimize} \quad \sum_j y_j \qquad (3.9)$$

$$\text{Minimize} \quad f \qquad (3.10)$$

The constraints of the problem are explained next.

Each item should be assigned to exactly one bin.

$$\sum_j x_{ij} = 1, \qquad \forall \text{ i} \qquad (3.3)$$

If an item is assigned to a bin, then the bin should be opened.

$$x_{ij} \leq y_j, \qquad \forall \text{ i and j} \qquad (3.4)$$

Constraint (3.5) calculates the overdeviation due to bin capacity.

$$\sum_i w_i x_{ij} - d_j \leq c y_j, \qquad \forall \text{ j} \qquad (3.5)$$

Constraint (3.12) calculates the maximum overdeviation f.

$$f \geq d_j, \qquad \forall \text{ j} \qquad (3.11)$$

The overdeviation of a bin should be greater than or equal to 0.

$$d_j \geq 0 \qquad \forall \text{ j} \qquad (3.6)$$

x_{ij} 's are binary.

$$x_{ij} = 0, 1 \qquad \forall \text{ i and j} \qquad (3.7)$$

y_j values are binary. Note that y_j takes value 1 if x_{ij} is 1.

$$y_j = 0, 1 \qquad \forall \text{ j} \qquad (3.8)$$

3.4. Properties of Efficient Solutions

In this section, three properties of the efficient solutions are given. Karasakal et al. [19] define the range of nondominated objective vectors for the number of bins and total/maximum overdeviation problems. Karasakal et al. [19] find that efficient solutions lie between the solution with one bin and the solution with zero total/maximum overdeviation.

We define a two-step approach to generate the nondominated objective vectors. The solution with one bin gives the total/maximum overdeviation. The upper bound on the total/maximum overdeviation of all nondominated objective vectors is ($\sum_i w_i$ – c) and it gives an efficient solution (1, $\sum_i w_i$ – c).

Therefore; two extreme efficient solutions are; (z*,0) and (1, $\sum_i w_i - c$)

We now present the range for the number of bins and range for the total / maximum overdeviation.

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

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

This follows an upper bound on the number of efficient solutions is

Min{z*-1 +1, $\sum_i w_i - c$ - 0 +1} = Min {z* , $\sum_i w_i - c + 1$}

Note that the number of efficient solutions is bounded by z*, hence n.

Property 1.

Karasakal et al. [19] proposed an approach to find total overdeviation easily when the remaining capacities of all bins are equal to zero.

Recall that the total overdeviation is the sum of the deviations over the capacity of all bins. In the second stage of Problem I, total overdeviation is calculated by decreasing the number of bins by one each time. If there is no slack in any one of the bins, decreasing the number of bins by one increases the total ovedeviation by c units.

Let t be the minimum total overdeviation where the number of bins is equal to m.

Decreasing the number of bins by one unit increases the minimum total overdeviation by c units.

t (m-1) = t (m) + c if all bins are fully loaded, i.e.,

if $\sum_i w_i x_{ij} \geq c$ $\forall$ j

CHAPTER 4

SOLUTION APPROACHES

Our bicriteria bin packing problems aim to find the set of efficient solutions for minimizing the number of bins and minimizing the total/maximum overdeviation objectives. In this chapter, we present our solution approaches. In Section 4.1 the stages of our approach are given. Section 4.2 and Section 4.3 present our lower bounds and upper bounds respectively.

4.1. A Two Stage Solution Procedure

A two stage solution procedure is applied to generate all efficient solutions. Two-stage procedure for Problem I is presented first and then the solution procedure for Problem II is described.

Problem I :

Minimization of the number of bins and minimization of the total overdeviation are the objectives of the Problem I. In stage I of Problem I, the aim is minimizing the number of bins for zero deviation over capacity. In stage II of Problem I, minimum total overdeviation for the given number of bins is found. All efficient solutions for the number of bins and total overdeviation problem are generated.

Stage I of Problem I:

In this stage, the classical bin packing problem is solved.

The objective function is as stated below:

Minimize

$$z = \sum_j y_j \qquad (4.1)$$

The constraints, are as stated below:

An upper bound for the optimal number of bins y_j, should not be exceeded.

$$\sum_j y_j \leq UB_1 \qquad (4.2)$$

A lower bound for the optimal number of bins y_j, is no bigger than the optimal number of bins.

$$\sum_j y_j \geq LB_1 \qquad (4.3)$$

The constraint set (4.4) requires the assignment of each item.

$$\sum_j x_{ij} = 1 \qquad \forall \text{ i} \qquad (4.4)$$

In this stage, any deviation over the capacity is not allowed; therefore constraint (4.5) is modified. The sum of the weights of items in a bin should be no bigger than the bin capacity.

$$\sum_i w_i x_{ij} \leq c y_j \qquad \forall \text{ j} \qquad (4.5)$$

The constraint set (4.6) relates the assignment variable to the bin opening variable.

$$x_{ij} \leq y_j \qquad \forall \text{ i and j} \qquad (4.6)$$

Constraint set (4.7) states the binary nature of the variable.

$$0 \leq x_{ij} \leq 1 \text{ and } x_{ij} \text{ is integer} \qquad \forall \text{ i and j} \qquad (4.7)$$

Constraint set (4.8) is for nonnegativity.

$$y_j \geq 0 \qquad \forall \text{ j} \qquad (4.8)$$

An optimal solution to the above model is found in stage I and let z* be the optimal number of bins. Stage II finds the minimum total overdeviation using the optimal number of bins, z* value. The first efficient solution is (z*,0). In stage II of Problem I, all efficient solutions are generated by decreasing the number of bins by one each time.

Stage II of Problem I:

This stage aims to minimize total overdeviation for given number of bins.

The objective function is as stated below:

Minimization of total overdeviation is as expressed below.

Minimize

$$t = \sum_j d_j \qquad (4.9)$$

The constraints, are as stated below:

Number of bins is k* for the first iteration of stage II. The constraint set (4.10) states that the number of bins is decreased by one in each iteration.

$$\sum_j y_j = k^* - 1 \qquad (4.10)$$

The following constraint set imposes an upper bound for the optimal number of total overdeviation d_j.

$$\sum_j d_j \leq UB_2 \qquad (4.11)$$

A lower bound for the optimal number of total overdeviation d_j, is used via the following constraint set.

$$\sum_j d_j \geq LB_2 \qquad (4.12)$$

The constraint sets (4.4), (4.6), (4.7), and (4.8) used in the first stage are also formulated this stage.

In this stage, deviation over the capacity is allowed. The sum of the weights of items in a bin should be no bigger than the bin capacity and deviation over the capacity.

$$\sum_i w_i x_{ij} - d_j \leq c y_j \qquad \forall j \qquad (4.13)$$

where

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

Problem II:

Minimization of the number of bins and minimization of the maximum overdeviation are the objectives of Problem II. In stage I of Problem II, the aim is to minimize the number of bins for zero deviation over capacity. Stage II of Problem II is the minimax problem and the minimum of the maximum overdeviation is found for the given number of bins. All nondominated objective vectors are generated for the number of bins and maximum overdeviation problem.

Stage I of Problem II:

In this stage as in stage I of Problem I, the classical bin packing problem is solved.

Stage II of Problem II:

In this stage, the aim is to minimize maximum overdeviation for a given number of bins.

Objective Function, is as stated below:

Minimization of maximum overdeviation is as expressed below.

Minimize

$$f \tag{4.14}$$

The constraints, are as stated below:

The constraint set (4.19) states that k value is decreased by 1.

$$\sum_{j} y_j = k^* - 1 \tag{4.15}$$

The following constraint set imposes an upper bound for the optimal number of maximum overdeviation *f*.

$$f \leq UB_3 \tag{4.16}$$

A lower bound for the optimal number of maximum overdeviation *f*, is used via the following constraint set.

$$f \geq LB_3 \tag{4.17}$$

Constraint set (4.22) calculates the maximum overdeviation. Maximum overdeviation will be greater than or equal to the total overdeviation of all bins.

$$f \geq d_j \qquad \forall \text{ j} \tag{4.18}$$

The constraint sets (4.4), (4.6), (4.7), and (4.8) used in the first stage and are also formulated in this stage. Constraint set (4.13) are also formulated in this stage of Problem II.

where

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

4.2. Lower Bounding Procedures

Three lower bounds on the optimal objective function values are presented. First one is the lower bound on the optimal number of bins, the second one is the lower bound on the optimal total overdeviation and the last lower bound is on the optimal maximum overdeviation. We make rounding as all parameters are integers.

Lower Bound I

Lower bound I, LB_1 is calculated for stage I of Problem I and Problem II solution procedures. The first stage is the same for Problem I and Problem II and LB_1 is valid for both problems.

Martello and Toth [24] developed a lower bound for the minimum number of bins. Their lower bound is based on the total weight of items. Total weight of items divided by the capacity of bins [$\sum_i w_i$ / c] is a lower bound for the optimal number of bins with zero total/maximum overdeviation.

$$LB_1 = \left\lceil \sum_i wi \,/\, c \right\rceil \tag{4.19}$$

Therefore, the constraint (4.3) becomes

$$\sum_j y_j \geq \left\lceil \sum_i wi \,/\, c \right\rceil \tag{4.20}$$

Lower Bound II

Lower bound II, LB_2 is calculated for the stage II of Problem I solution procedure. Recall that stage II of Problem I is finding the minimum total overdeviation for the given number of bins.

Karasakal et al. [19] proposed a lower bound for the minimum total overdeviation. Max $\{0, [\sum_i w_i - (\text{number of bins} \times c)]\}$ is a lower bound for the optimal total overdeviation with given number of bins.

$$LB_2 = \left\lceil \text{Max}\{0, [\sum_i wi - (number\ of\ bins \times c)]\} \right\rceil \quad (4.21)$$

Therefore, the constraint (4.12) becomes

$$\sum_j d_j \geq \left\lceil \text{Max}\{0, [\sum_i wi - (number\ of\ bins \times c)]\} \right\rceil \quad (4.22)$$

Lower Bound III

Lower bound III, LB_3 is calculated for the stage II of Problem II solution procedure. The objective of stage II of Problem II is to minimize the maximum overdeviation for the given number of bins.

Karasakal et al. [19] present a lower bound for the maximum overdeviation. Max $\{0, [[\sum_i w_i - (\text{number of bins} \times c)] / \text{number of bins}]\}$ is a lower bound for the minimum of maximum overdeviation with given number of bins.

$$LB_3 = \left\lceil \text{Max}\{0, [[\sum_i wi - (\text{number of bins} \times c)] / \text{number of bins}]\} \right\rceil \quad (4.23)$$

Therefore, the constraint (4.17) becomes

$$f \geq \left\lceil \text{Max}\{0, [[\sum_i wi - (\text{number of bins} \times c)] / \text{number of bins}]\} \right\rceil \quad (4.24)$$

These three lower bounds are the rounded up values of the optimal.

4.3. Upper Bounding Procedures

Three upper bounds are presented on the objective function values. First one is the upper bound on the optimal number of bins, the second one is on the optimal total overdeviation and the last one is on the optimal maximum overdeviation.

Upper Bound I

The first stage of Problem I and Problem II is the classical bin packing problem. Therefore, the upper bound on the optimal number of bins is derived from a well-known good placement approximation algorithm. The upper bound is calculated using best fit decreasing (BFD) algorithm developed by Johnson [17]. BFD is used, as it is easy to implement and the algorithm has a good performance.

In the BFD algorithm, items are first sorted according to the decreasing order of their weights. BFD calculates the remaining capacity of all bins and inserts an item into a feasible bin with smallest remaining capacity. BFD creates a new bin if the item is not assigned to any bin and inserts the item into that bin.

Finding UB_1:

Step I: Sort the items according to their decreasing weights.

Step II: Open one bin for the first time.

Step III: Calculate the remaining capacity of the open bins.

Step IV: Insert the item into a feasible bin having the smallest remaining capacity. If there is no feasible bin to insert, open a new bin. Insert the item to that new bin.

Step V: Apply Step III and Step IV for all items until all items are inserted to a bin.

Upper Bound II:

An upper bound is derived for the second stage of Problem I, UB_2. Recall that stage II of Problem I is minimizing the total overdeviation. Karasakal et al. [19] modified the best fit decreasing algorithm for the total overdeviation problem. Steps of their constructive heuristic are as follows.

Step I: Sort the items according to their decreasing weights.

Step II: Assign the first item to the feasible bin having the smallest remaining capacity. If there is no feasible bin to insert, assign the item to an arbitrarily selected bin.

Step III: Apply Step II for all items until all items are inserted to a bin.

Karasakal et al. [19] applied an improvement heuristic for the solution of the constructive heuristic if $(UB_2\text{-}LB_2)/UB_2 \geq 0{,}4$. Improvement heuristic starts from the solution of the constructive heuristic. They aim to decrease the problem size by eliminating some bins and their assigned items. Their improvement heuristic is applied for all solutions of the constructive heuristic to find UB_2 in our study. The improvement heuristic sorts the bins with no overdeviation in nonincreasing order of their assigned weights and eliminates a number of bins having no overdeviation. The bins with total assigned weight equal to the capacity of bins are eliminated first. The steps of the improvement heuristic:

Step I: Select the bins with zero overdeviation

Step II: Sort the selected bins in nonincreasing order of their assigned weights.

Step III: Eliminate first b bins and the assigned items of that bins

Step IV: Solve the total overdeviation problem (stage II of Problem I) for the remaining bins and items.

In step III of the improvement heuristic, Karasakal et al. [19] define a method to calculate b value based on their computational results. They set;

b = [y x 10(average weight of items/ capacity of bins)]

where

$$y = \begin{cases} 1 & \text{if number of items} = 50 \\ 1.5 & \text{if number of items} = 75 \\ 2 & \text{if number of items} = 100 \end{cases}$$

average weight of items = $\sum w_i$ / n

Figures 4.1 , 4.2 and 4.3 show schematic view of improvement heuristic. Value of b is taken to be 3 in this example. Figure 4.1 shows elimination procedure of improvement heuristic. Bins are first sorted in nonincreasing order of their assigned weights and then 3 eliminated bins are selected and reduced configuration of bins are shown.

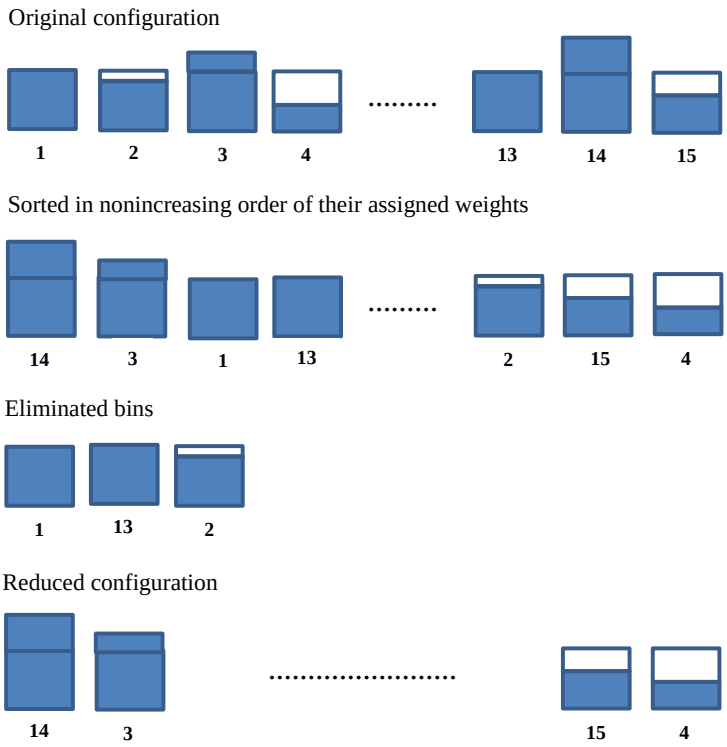

Figure 4.1. Elimination procedure of improvement heuristic

Figure 4.2 shows the solution of total overdeviation problem for reduced configuration.

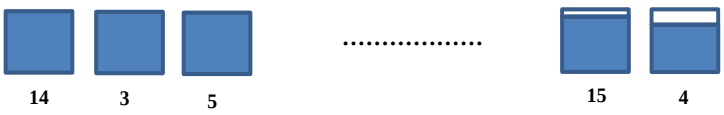

Figure 4.2. Solution of reduced configuration

Upper bound III is found by the solution of reduced configuration and the eliminated bins. Figure 4.3 shows the final solution of problem.

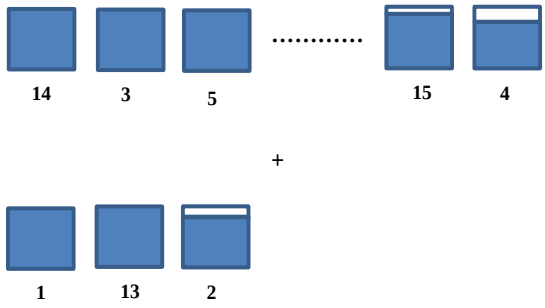

Figure 4.3. Final solution by adding two configurations

Upper Bound III:

UB_3 is an upper bound for the maximum overdeviation. The second stage of problem II is similar to the minimizing makespan with parallel machines problem, therefore, the heuristic of Graham [14] is used to calculate the upper bound. The steps of the Graham [14] algorithm are stated below:

Step I: Sort the items according to their decreasing weights.

Step II: Assign the first item to the feasible bin having the smallest remaining capacity. If there is no feasible bin to insert, assign the item to the bin with smallest overdeviation including that item, if all bins are overloaded, assign the item to the bin with smallest overdeviation.

Apply the step for all items until all items are inserted to a bin.

These three upper bounds are the rounded down results of the calculated bounds.

CHAPTER 5

COMPUTATIONAL EXPERIMENTS

In this chapter, computational results of the experiments are presented. In Section 5.1, generation of the test problems is explained. In Section 5.2, the results are given and discussed.

Mathematical models are solved by GAMS 24.2 using CPLEX MIP solver with zero absolute and relative gaps. The algorithms are coded in C programming language. Runs are performed on a computer with an Intel Core i7 3,10 GHz processor and 16GB of RAM.

5.1. Problem Sets

The problem data used in the experiments are generated randomly. 12 problem sets, identified by the number of items, item weights and bin capacities, are used. Three values, 50, 75 and 100 are used for the number of items. Problem sets with 50 and 100 items are taken from Scholl, Klein, and Jurgens [33]. Data for item weights are generated from discrete uniform distributions within ranges [1,100] and [30,100]. Two values, 100 and 150 are used for bin capacities. For each problem set, 10 problem instances are solved, hence a total of 120 problem instances are solved in the experiments. For each problem instance, many problems are solved to find all nondominated sets.

5.2. Analysis of the Results

The solution times are expressed in Central Processing Unit (CPU) seconds. For each problem instance and each total/maximum overdeviation problem to generate an efficient solution, CPU time limits are set to 3600 seconds for the number of bins and the total overdeviation problems (Problem I), and 9000 seconds for the number of bins and the maximum overdeviation problems (Problem II). The execution of the algorithms is terminated if the optimal solution is not found in the time limits.

The results of the computational experiments for problems I and II are given in the following sections.

5.2.1. Problem I: Minimization of Number of Bins and Total Overdeviation

The solution times of the number of bins problem, the total overdeviation problem and the improvement heuristic for the upper bound of the total overdeviation problem are presented in Tables 5.1, 5.2 and 5.3 respectively. For each problem set, maximum and average CPU times are reported. Maximum and average CPU times of all problems are given in columns titled “Max(1)” and “Avg(1)”, CPU times of all optimally solved problems in the prespecified time limits (i.e. excluding the unsolved problems) are reported in columns titled “Max(2)” and “Avg(2)”.

Table 5.1 reports the CPU times of finding the optimal solution to the number of bins for Problem I.

Since the number of bins problem is strongly NP-hard one should expect that the solution times increase exponentially with the problem size. From Table 5.1, it can be seen that the solution times increase exponentially with the problem size but there are some exceptions due to the lower and upper bounds used in the MIPs. When capacity is 150; for both 75 items and 100 items; the average CPU times are negligible. Thanks to the strong lower and upper bounds; for all these 20 problems the optimal solutions are found using only the bounds.

Moreover, the average CPU time for the problems increases when the item weights are generated within the range [30,100]. Looking at all the instances (i.e. without excluding the unsolved ones) for each problem set this inference can be made. This is due to the fact that when the minimum weight is 30, the weights of all items lie in a smaller range and the trade-off between the items is more difficult to handle.

Table 5. 1. The CPU times for the number of bins problem when time limit is 3600 seconds

n = 50					
C	**w_i**	**Max(1)***	**Avg(1)***	**Max(2)****	**Avg(2)****
100	[1, 100]	13.56	2.13	13.56	2.13
150	[1, 100]	58.45	5.99	58.45	5.99
100	[30, 100]	3600.00	1086.69	0.72	0.48
150	[30, 100]	3600.00	634.93	2733.01	305.38
n = 75					
C	**w_i**	**Max(1)***	**Avg(1)***	**Max(2)****	**Avg(2)****
100	[1, 100]	5.02	2.28	5.02	2.28
150	[1, 100]	0.00	0.00	0.00	0.00
100	[30, 100]	4.84	2.67	4.84	2.67
150	[30, 100]	3600.00	1463.52	129.50	38.85
n = 100					
C	**w_i**	**Max(1)***	**Avg(1)***	**Max(2)****	**Avg(2)****
100	[1, 100]	3600.00	917.12	1907.21	246.29
150	[1, 100]	0.00	0.00	0.00	0.00
100	[30, 100]	3600.00	2524.30	8.48	6.11
150	[30, 100]	3600.00	1545.58	661.46	173.23

* CPU times in seconds of all problems
** CPU times in seconds of all optimally solved problems

In Table 5.2 the average and maximum CPU times for the total overdeviation problem are presented. The problems that are solved by Property 1 are not taken into account, as they require negligible time.

Note that for the total overdeviation problem as the problem size increases, (i.e., as the number of items or bins increase) the average CPU time (i.e. without excluding the unsolved ones) also increases. Also, as the minimum weight increases to 30; the

total overdeviation problem becomes harder to solve. Another result is that as the capacity increases the problem becomes easier to solve.

Table 5. 2. The CPU times for the total overdeviation problem

n = 50					
C	**w_i**	**Max(1)***	**Avg(1)***	**Max(2)****	**Avg(2)****
100	[1, 100]	3600.00	1599.72	2746.37	167.85
150	[1, 100]	0.77	0.08	0.77	0.08
100	[30, 100]	3600.00	2736.58	2954.29	151.90
150	[30, 100]	3600.00	1423.45	15.52	2.25
n = 75					
C	**w_i**	**Max(1)***	**Avg(1)***	**Max(2)****	**Avg(2)****
100	[1, 100]	3600.00	1786.28	236.59	18.69
150	[1, 100]	0.00	0.00	0.00	0.00
100	[30, 100]	3600.00	3075.03	3251.95	345.66
150	[30, 100]	3600.00	1432.27	162.49	14.77
n = 100					
C	**w_i**	**Max(1)***	**Avg(1)***	**Max(2)****	**Avg(2)****
100	[1, 100]	3600.00	2508.08	54.29	4.59
150	[1, 100]	3600.00	329.00	0.00	0.00
100	[30, 100]	3600.00	3394.75	1199.12	286.12
150	[30, 100]	3600.00	2068.13	0.00	0.00

* CPU times in seconds of all problems
** CPU times in seconds of all optimally solved problems

Not surprisingly, the improvement heuristic shows the same properties with the total overdeviation problem, as it is a smaller sized version of the problem as, can be seen from Table 5.3.

Moreover, for the improvement heuristic, it is seen that as the number of bins increases, the average CPU time (Avg(1)) also increases. Also, one can see that from Table 5.3 Avg(1) column as the minimum weight increases to 30; the problem is harder to solve except for problem set with 100 items and capacity of 100.

Table 5. 3. The CPU times for the improvement heuristic for the total overdeviation problem

n = 50					
C	**w_i**	**Max(1)***	**Avg(1)***	**Max(2)****	**Avg(2)****
100	[1, 100]	3600.00	1340.36	311.18	61.92
150	[1, 100]	1.01	0.34	1.01	0.34
100	[30, 100]	3600.00	1509.59	1234.66	62.73
150	[30, 100]	3600.00	1055.04	584.48	75.37
n = 75					
C	**w_i**	**Max(1)***	**Avg(1)***	**Max(2)****	**Avg(2)****
100	[1, 100]	3600.00	2201.92	3323.91	747.62
150	[1, 100]	3271.65	341.69	3271.65	341.69
100	[30, 100]	3600.00	2224.78	980.74	53.85
150	[30, 100]	3600.00	2333.71	3254.04	1318.81
n = 100					
C	**w_i**	**Max(1)***	**Avg(1)***	**Max(2)****	**Avg(2)****
100	[1, 100]	3600.00	2970.28	3288.56	1335.25
150	[1, 100]	3600.00	719.47	978.67	431.15
100	[30, 100]	3600.00	2496.92	1486.02	67.77
150	[30, 100]	3600.00	3240.71	3274.68	2770.97

* CPU times in seconds of all problems
** CPU times in seconds of all optimally solved problems

Table 5.4 reports the number of unsolved problems in 3600 seconds for the number of bins problem. Recall that 10 problem instances are solved for each problem set. The inferences made for the CPU times are valid here. When the problem size increases, it is more difficult to obtain the optimal solutions. There are only 4 unsolved instances out of 40 instances when there are 50 items, whereas there are 13 instances out of 40 when there are 100 items.

For the minimum weight case, the situation is similar; only 2 out of 60 instances are unsolved when minimum weight is 1; while there are 19 unsolved instances out of 60 when minimum weight is 30. There are 12 unsolved instances out of 60 when the capacity is 100, whereas there are 9 unsolved instances out of 60 when capacity is 150.

Table 5. 4. The number of unsolved instances for the number of bins problem when time limit is 3600 seconds

		# of unsolved instances*		
C	**w_i**	**n = 50**	**n = 75**	**n = 100**
100	[1, 100]	0	0	2
150	[1, 100]	0	0	0
100	[30, 100]	3	0	7
150	[30, 100]	1	4	4

*Out of 10 problem instances

In Table 5.5 upper bounds on the number of efficient solutions for total overdeviation problem are presented. Maximum and average number of efficient solutions per problem instance are reported. Recall that 10 instances are solved for each problem set.

Table 5. 5. The number of efficient solutions for total overdeviation problem

n = 50				
C	**w_i**	**Upper bound on the # of efficient solutions**	**Max # of efficient solutions per problem instance**	**Avg # of efficient solutions per problem instance**
100	[1, 100]	254	30	25.40
150	[1, 100]	161	19	16.10
100	[30, 100]	358	39	35.80
150	[30, 100]	213	23	21.30
n = 75				
C	**w_i**	**Upper bound on the # of efficient solutions**	**Max # of efficient solutions per problem instance**	**Avg # of efficient solutions per problem instance**
100	[1, 100]	403	44	40.30
150	[1, 100]	252	26	25.20
100	[30, 100]	538	56	53.80
150	[30, 100]	318	33	32.00
n = 100				
C	**w_i**	**Upper bound on the # of efficient solutions**	**Max # of efficient solutions per problem instance**	**Avg # of efficient solutions per problem instance**
100	[1, 100]	524	61	52.40
150	[1, 100]	336	36	33.60
100	[30, 100]	736	81	73.60
150	[30, 100]	431	46	43.10

Table 5.6 reports upper bound on the number of efficient solutions for each problem set, number and percentage of exact solutions found and number and percentage of solutions found by Property 1. For example when there are 100 items, capacity is 100 and minimum weight is 30, 511 exact solutions out of 736 (i.e., at least 69% of the problems) are found. The rest could not be found within the specified time limit.

Moreover, 493 of them (i.e. at least 67% of the problems) are found by Property 1. It can be concluded that Property 1 is quite useful.

In summary, in 120 problem instances, upper bound on the total number of efficient solutions is 4526; and Property 1 finds 3726 of the efficient points which correspond to at least 82.32%. 29.50% of the remaining efficient solutions could be solved in the time limit.

Table 5.6 shows that Property 1 performs the worst when capacity is 100 and minimum weight is 30; regardless of the number of items. For those problem sets, the problem is dense; i.e., there are more bins than those of any other problem sets since capacity is low and average weight is high; and weights of the items are closer to each other.

We observe that the capacity is 150, Property 1 works better. The percentage of the solutions found by Property 1 increases to 94.57%. This is due to the fact that; when the number of bins decreases by 1, one bin will be emptied, and items in that bin will be distributed to the other bins which will result in filling more bins at a time.

Any increase in the number of items, decreases in capacity or increases in minimum weight of items results in a decrease in the percentage of finding exact solutions in vast majority of the instances.

Table 5. 6. Results on the efficient solutions for the total overdeviation problem

n = 50						
C	**w_i**	**Upper bound on the # of efficient solutions**	**# of exact solutions found**	**% of exact solutions found**	**# of soln. found by Property 1**	**% of soln. found by Property 1**
100	[1, 100]	254	239	94	218	86
150	[1, 100]	161	161	100	151	94
100	[30, 100]	358	272	76	243	68
150	[30, 100]	213	206	97	195	92
n = 75						
C	**w_i**	**Upper bound on the # of efficient solutions**	**# of exact solutions found**	**% of exact solutions found**	**# of soln. found by Property 1**	**% of soln. found by Property 1**
100	[1, 100]	403	377	93	357	89
150	[1, 100]	252	252	100	242	96
100	[30, 100]	538	397	74	368	68
150	[30, 100]	318	313	98	299	93
n = 100						
C	**w_i**	**Upper bound on the # of efficient solutions**	**# of exact solutions found**	**% of exact solutions found**	**# of soln. found by Property 1**	**% soln. found by Property 1**
100	[1, 100]	524	481	92	463	88
150	[1, 100]	336	335	100	325	97
100	[30, 100]	736	511	69	493	67
150	[30, 100]	431	418	97	408	95

The model returns non-optimal solution to all unsolved instances of the number of bins problem within the specified time limit. Table 5.7 reports the quality of those solutions, i.e., average and maximum percentage deviations from the lower and upper bounds. It is seen that the quality of solutions of unsolved problems is not better than the upper bounds. Average percentage deviation from the upper bound is

0.5% the model solutions are close to the upper bounds for the number of bins problem.

Table 5.7. The performance of the model solutions for unsolved instances - The Number of Bins Problem

n = 50						
C	**w_i**	**# of unsolved problem instances**	**Avg % deviation from UB1**	**Max % deviation from UB1**	**Avg % deviation from LB1**	**Max % deviation from LB1**
100	[1, 100]	0	-	-	-	-
150	[1, 100]	0	-	-	-	-
100	[30, 100]	3	0.00	0.00	13.14	15.00
150	[30, 100]	1	0.00	0.00	4.35	4.35
n = 75						
C	**w_i**	**# of unsolved problem instances**	**Avg % deviation from UB1**	**Max % deviation from UB1**	**Avg % deviation from LB1**	**Max % deviation from LB1**
100	[1, 100]	0	-	-	-	-
150	[1, 100]	0	-	-	-	-
100	[30, 100]	0	-	-	-	-
150	[30, 100]	4	0.00	0.00	3.06	3.13
n = 100						
C	**w_i**	**# of unsolved problem instances**	**Avg % deviation from UB1**	**Max % deviation from UB1**	**Avg % deviation from LB1**	**Max % deviation from LB1**
100	[1, 100]	0	-	-	-	-
150	[1, 100]	0	-	-	-	-
100	[30, 100]	7	0.00	0.00	12.27	15.85
150	[30, 100]	4	2.27	2.33	2.27	2.33

Table 5.8 shows the quality of model solutions within the specified time limits to unsolved problems for total overdeviation problem. The number of model solutions and average and maximum percentage deviations from lower bound and upper bound are reported in the table. The model can find more solution when the number of

items is 50. For 50 items, the solutions are closer to the upper bound whereas solutions are closer to the lower bound for problems with 75 and 100 items.

Table 5.8. The performance of the model solutions for unsolved problems - Total Overdeviation Problem

n = 50							
C	**w_i**	**Upper bound on # of unsolved problems**	**# of solution found by model to unsolved problems**	**Avg % deviation from UB2**	**Max % deviation from UB2**	**Avg % deviation from LB2**	**Max % deviation from LB2**
100	[1, 100]	15	7	3.35	9.84	11.53	31.97
150	[1, 100]	0	0	-	-	-	-
100	[30, 100]	86	27	5.60	11.98	10.31	76.74
150	[30, 100]	7	1	0.00	0.00	4.44	4.44
n = 75							
C	**w_i**	**Upper bound on # of unsolved problems**	**# of solution found by model to unsolved problems**	**Avg % deviation from UB2**	**Max % deviation from UB2**	**Avg % deviation from LB2**	**Max % deviation from LB2**
100	[1, 100]	26	0	-	-	-	-
150	[1, 100]	0	0	-	-	-	-
100	[30, 100]	141	20	8.21	14.72	0.97	3.30
150	[30, 100]	7	0	-	-	-	-
n = 100							
C	**w_i**	**Upper bound on # of unsolved problems**	**# of solution found by model to unsolved problems**	**Avg % deviation from UB2**	**Max % deviation from UB2**	**Avg % deviation from LB2**	**Max % deviation from LB2**
100	[1, 100]	43	0	-	-	-	-
150	[1, 100]	1	0	-	-	-	-
100	[30, 100]	225	11	8.45	11.32	0.69	2.97
150	[30, 100]	13	0	-	-	-	-

Table 5.9. presents the performance of the lower and upper bounds for the number of bins problem. It can be seen that as the number of bins and minimum weight increase the number of bins thereby the complexity of the problem increases. When capacity increases, especially when the minimum weight is 1, the bounds perform well together. Out of 30 instances where capacity of the bins is 150 and minimum weight is 1, the lower and upper bounds are equal for all instances, except two.

Table 5. 9. The lower and upper bound values for the number of bins problem

n = 50						
C	**w_i**	**Avg LB1**	**Max LB1**	**Avg UB1**	**Max UB1**	**# of instances UB1=LB1**
100	[1, 100]	25.20	28	26.50	31	2
150	[1, 100]	17.10	20	17.30	20	8
100	[30, 100]	32.60	35	36.90	40	0
150	[30, 100]	22.00	23	22.90	24	1
n = 75						
C	**w_i**	**Avg LB1**	**Max LB1**	**Avg UB1**	**Max UB1**	**# of instances UB1=LB1**
100	[1, 100]	39.00	41	41.30	45	0
150	[1, 100]	26.20	27	26.20	27	10
100	[30, 100]	48.70	51	54.80	57	0
150	[30, 100]	32.60	34	33.60	35	1
n = 100						
C	**w_i**	**Avg LB1**	**Max LB1**	**Avg UB1**	**Max UB1**	**# of instances UB1=LB1**
100	[1, 100]	51.10	58	53.40	62	1
150	[1, 100]	34.60	37	34.60	37	10
100	[30, 100]	65.40	69	74.60	82	0
150	[30, 100]	43.60	46	45.00	47	0

Table 5.10 reports the performance of the upper bounds for the number of bins problem. In 79 out of 99 optimally solved instances the upper bound is equal to the optimal value. Average percentage deviation of the upper bound from the optimal solution is only 0.75% for the number of bins problem. The worst performance of the upper bound is when capacity is 150 and minimum weight is 30 regardless of the number of items.

Table 5. 10. The upper bound values on the number of bins and the optimal number of bins

n = 50									
C	**w_i**	**# of optimally solved instances**	**Avg UB1**	**Max UB1**	**Avg Opt**	**Max Opt**	**# of instances UB1= Opt**	**Avg % deviation from optimal**	**Max % deviation from optimal**
100	[1, 100]	10	26.50	31	26.40	31	9	0.50	5.00
150	[1, 100]	10	17.30	20	17.10	20	8	1.21	6.25
100	[30, 100]	7	36.43	40	36.29	40	6	0.39	2.70
150	[30, 100]	9	22.89	24	22.22	24	3	3.08	4.76
n = 75									
C	**w_i**	**# of optimally solved instances**	**Avg UB1**	**Max UB1**	**Avg Opt**	**Max Opt**	**# of instances UB1= Opt**	**Avg % deviation from optimal**	**Max % deviation from optimal**
100	[1, 100]	10	41.30	45	41.30	45	10	0.00	0.00
150	[1, 100]	10	26.20	27	26.20	27	10	0.00	0.00
100	[30, 100]	10	54.80	57	54.80	57	10	0.00	0.00
150	[30, 100]	6	33.33	35	32.50	34	1	2.54	3.23
n = 100									
C	**w_i**	**# of optimally solved instances**	**Avg UB1**	**Max UB1**	**Avg Opt**	**Max Opt**	**# of instances UB1= Opt**	**Avg % deviation from optimal**	**Max % deviation from optimal**
100	[1, 100]	8	54.50	62	54.50	62	8	0.00	0.00
150	[1, 100]	10	34.60	37	34.60	37	10	0.00	0.00
100	[30, 100]	3	73.33	75	73.33	75	3	0.00	0.00
150	[30, 100]	6	45.00	47	44.17	47	1	1.91	2.38

Table 5.11 shows the performance of the lower bounds for the number of bins problem. In 52 out of 99 optimally solved problems the lower bound is equal to the optimal value. So, it can be claimed that lower bound is not as strong as the upper bound, the average percentage deviation of the lower bound from the optimal solution is 3.68% for the number of bins problem. The worst performance of the lower bound is observed when capacity is 100 and item weights are generated from the range [30,100] regardless of the number of items. Deviations from the optimal solution is smaller when capacity is 150.

Table 5. 11. The lower bound values on the number of bins and the optimal number of bins

n = 50									
C	**w_i**	**# of optimally solved instances**	**Avg LB1**	**Max LB1**	**Avg Opt**	**Max Opt**	**# of instances LB1=Opt**	**Avg % deviation from optimal**	**Max % deviation from optimal**
100	[1, 100]	10	25.20	252	26.40	31	3	4.19	9.68
150	[1, 100]	10	17.10	171	17.10	20	10	0.00	0.00
100	[30, 100]	7	32.43	227	36.29	40	0	10.50	13.51
150	[30, 100]	9	22.00	198	22.22	24	7	0.95	4.35
n = 75									
C	**w_i**	**# of optimally solved instances**	**Avg LB1**	**Max LB1**	**Avg Opt**	**Max Opt**	**# of instances LB1=Opt**	**Avg % deviation from optimal**	**Max % deviation from optimal**
100	[1, 100]	10	39.00	390	41.30	45	0	5.42	15.56
150	[1, 100]	10	26.20	262	26.20	27	10	0.00	0.00
100	[30, 100]	10	48.70	487	54.80	57	0	11.10	14.04
150	[30, 100]	6	32.50	195	32.50	34	6	0.00	0.00
n = 100									
C	**w_i**	**# of optimally solved instances**	**Avg LB1**	**Max LB1**	**Avg Opt**	**Max Opt**	**# of instances LB1=Opt**	**Avg % deviation from optimal**	**Max % deviation from optimal**
100	[1, 100]	8	51.88	415	54.50	62	1	4.51	10.00
150	[1, 100]	10	34.60	346	34.60	37	10	0.00	0.00
100	[30, 100]	3	64.33	193	73.33	75	0	12.25	13.33
150	[30, 100]	6	44.00	264	44.17	47	5	0.36	2.13

Table 5.12 shows the effect of the improvement heuristic on the upper bound. On the average, the improvement is 28.79%. The heuristic improves the upper bound at most when capacity is150 and item weights are generated from [30,100] by 50.10% on the average.

Table 5.12. The performance of the improvement heuristic on the total overdeviation problem

Total Overdeviation Problem		**% improvement**		
C	**w_i**	**n=50**	**n=75**	**n=100**
100	[1, 100]	29.28	31.14	36.23
150	[1, 100]	12.07	12.00	19.90
100	[30, 100]	25.54	26.50	26.19
150	[30, 100]	41.47	56.82	52.98

Table 5.13 illustrates the upper and lower bound for the total overdeviation problem. The bounds are evaluated for the problems that could not be solved by Property 1. Problems for which lower bound is equal to 0 are excluded. In 83 out of 761 problems, that is 10.91% of the problems, upper and lower bounds are equal. The bound values are closer when the capacities of bins are larger. Also when the minimum weight is 1, the bounds perform better.

Table 5.13. The lower and upper bound values for total overdeviation problem

n = 50									
C	**w_i**	**# of solns (LB2≠0)**	**Avg LB2**	**Max LB2**	**Avg UB2**	**Max UB2**	**# of solns UB2= LB2**	**Avg % deviation**	**Max % deviation**
100	[1, 100]	24	150.13	383	157.96	384	7	8.47	45.61
150	[1, 100]	10	84.40	141	84.50	142	9	0.07	0.70
100	[30, 100]	73	385.12	878	458.85	952	0	23.01	91.82
150	[30, 100]	15	126.40	318	129.00	323	7	3.52	28.30
n = 75									
C	**w_i**	**# of solns (LB2≠0)**	**Avg LB2**	**Max LB2**	**Avg UB2**	**Max UB2**	**# of solutions UB2= LB2**	**Avg % deviation**	**Max % deviation**
100	[1, 100]	30	155.17	406	170.63	406	10	18.61	92.68
150	[1, 100]	10	61.60	106	61.60	106	10	0.00	0.00
100	[30, 100]	109	552.55	1226	651.64	1302	0	23.28	92.93
150	[30, 100]	14	110.21	276	111.21	276	10	3.12	33.33
n = 100									
C	**w_i**	**# of solns (LB2≠0)**	**Avg LB2**	**Max LB2**	**Avg UB2**	**Max UB2**	**# of solutions UB2= LB2**	**Avg % deviation**	**Max % deviation**
100	[1, 100]	38	269.29	823	298.82	823	10	16.73	88.38
150	[1, 100]	11	83.55	151	84.64	151	10	8.39	92.31
100	[30, 100]	151	748.30	1707	879.07	1844	0	23.03	98.37
150	[30, 100]	18	133.22	354	139.44	354	10	12.41	74.29

Table 5.14 illustrates the strength of the upper bound (the improved upper bound). The results are reported for the problems whose optimal total overdeviation is found by the model. For 151 out of 197 problems, the upper bound values were equal to the optimal objective function values. The average deviation from the optimal value for all problems is 1.95%.

When the capacity increases, the upper bound performs better. Note that when capacity is 150, average percentage deviation from the exact solution is 0.12% whereas when capacity is 100 the average percentage deviation is 2.75%. When the problem space is more restricted (i.e. capacity is 100 and weight of item are generated from the range [30,100]) the results found by the algorithm deviates more from the exact solution regardless of the number of items. For this case, the

percentage deviation from the exact solution is 3.94% while for the case where the capacity is 150 and item weights are generated from the range [1,100] the percentage deviation is only 0.02%.

Table 5.14. The upper bound values on the total overdeviation and the optimal total overdeviation

n = 50									
C	**w_i**	**# of exact solns found by the model**	**Avg UB2**	**Max UB2**	**Avg exact**	**Max exact**	**# of solutions UB2= exact**	**Avg % deviation**	**Max % deviation**
100	[1, 100]	21	103.38	384	103.00	383	16	0.91	14.29
150	[1, 100]	10	84.50	142	84.40	141	9	0.07	0.71
100	[30, 100]	29	276.14	952	249.62	878	16	5.29	22.22
150	[30, 100]	11	142.55	323	141.09	318	8	0.61	4.33
n = 75									
C	**w_i**	**# of exact solns found by the model**	**Avg UB2**	**Max UB2**	**Avg exact**	**Max exact**	**# of solutions UB2= exact**	**Avg % deviation**	**Max % deviation**
100	[1, 100]	27	95.70	406	95.44	406	24	2.65	26.67
150	[1, 100]	10	61.60	106	61.60	106	10	0.00	0.00
100	[30, 100]	29	391.72	1302	362.83	1226	18	3.04	12.97
150	[30, 100]	11	122.64	276	122.64	276	11	0.00	0.00
n = 100									
C	**w_i**	**# of exact solns found by the model**	**Avg UB2**	**Max UB2**	**Avg exact**	**Max exact**	**# of solutions UB2= exact**	**Avg % deviation**	**Max % deviation**
100	[1, 100]	18	182.89	823	182.89	823	18	0.00	0.00
150	[1, 100]	10	91.80	151	91.80	151	10	0.00	0.00
100	[30, 100]	18	857.61	1844	810.61	1707	8	3.19	9.56
150	[30, 100]	10	187.70	354	187.70	354	10	0.00	0.00

The performances of the lower bound are given in Table 5.15 It is seen that in all problem sets the lower bound is equal to the exact value for 10 sets except one. 10 exact solutions for each problem set are the ones for which Property 1 becomes effective. The condition of the Property 1 holds when the exact value is equal to the lower bound. (Lower Bound=Total Weight-Total Capacity where Total Weight= Total Deviation + Total Capacity in order for the algorithm in Property 1 to apply).

Table 5.15. The lower bound values on the total overdeviation and the optimal total overdeviation

n = 50									
C	**w_i**	**# of exact solns found by the model (LB2≠0)**	**Avg LB2**	**Max LB2**	**Avg exact**	**Max exact**	**# of solutions LB2= exact**	**Avg % deviation**	**Max % deviation**
100	[1, 100]	11	181.36	383	103.00	383	10	4.15	45.61
150	[1, 100]	10	84.40	141	84.40	141	10	0.00	0.00
100	[30, 100]	10	690.80	878	690.80	878	10	0.00	0.00
150	[30, 100]	10	154.80	318	154.80	318	10	0.00	0.00
n = 75									
C	**w_i**	**# of exact solns found by the model (LB2≠0)**	**Avg LB2**	**Max LB2**	**Avg exact**	**Max exact**	**# of solutions LB2= exact**	**Avg % deviation**	**Max % deviation**
100	[1, 100]	10	241.60	406	241.60	406	10	0.00	0.00
150	[1, 100]	10	61.60	106	61.60	106	10	0.00	0.00
100	[30, 100]	10	1034.8	1226	1034.8	1226	10	0.00	0.00
150	[30, 100]	10	134.80	276	134.80	276	10	0.00	0.00
n = 100									
C	**w_i**	**# of exact solns found by the model (LB2≠0)**	**Avg LB2**	**Max LB2**	**Avg exact**	**Max exact**	**# of solutions LB2= exact**	**Avg % deviation**	**Max % deviation**
100	[1, 100]	10	324.10	823	324.10	823	10	0.00	0.00
150	[1, 100]	10	91.80	151	91.80	151	10	0.00	0.00
100	[30, 100]	10	1448.0	1707	1448.0	1707	10	0.00	0.00
150	[30, 100]	10	187.70	354	187.70	354	10	0.00	0.00

Table 5.14 shows that there are 204 exact solutions found by the model. In 120 of these solutions (1 for every problem instance) Property 1 starts to apply; so the lower bound is equal to upper bound (i.e. it is optimal). For the 84 remaining solutions for only one problem, lower bound is found to be different than 0, this shows that the lower bound has very little effect on the performance of the total overdeviation problem. Table 5.15 shows that the upper bound is equal to the lower bound only when the upper bound hits the optimal so Property 1 becomes effective.

5.2.2. Problem II: Minimization of the Number of Bins and Maximum Overdeviation

The solution times of the number of bins problem and the maximum overdeviation problem are reported in Tables 5.16 and 5.17 respectively. For each problem set, maximum and average CPU times are reported. In the tables, "Max(1)" and "Avg(1)" stand for the maximum and average CPU times of all problem instances, respectively. On the other hand, "Max(2)" and "Avg(2)" are respective CPU times, for all solved instances.

Table 5.16 reports the CPU times for the number of bins for Problem II. The difference between this table and Table 5.1 is the time limit. Recall that the CPU time limit is set to 3600 seconds for the number of bins and the total overdeviation problems, 9000 seconds for the number of bins and the maximum overdeviation problems.

The number of problems that can be solved within the limit changes and so do the average and maximum times. The change can only be seen for the large size problem instances. For the small size problems, i.e. for the problem sets for which all problems are solved to optimality within 3600 seconds, (for example 50 items, capacity is 150 and minimum weight is 1) average and maximum CPU times are the same with those of Table 5.1.

Table 5.16. The CPU times for the number of bins problem when time limit is 9000 seconds

n = 50					
C	**w_i**	**Max(1)***	**Avg(1)***	**Max(2)****	**Avg(2)****
100	[1, 100]	13.56	2.13	13.56	2.13
150	[1, 100]	58.45	5.99	58.45	5.99
100	[30, 100]	9000.00	2762.79	0.72	0.48
150	[30, 100]	9000.00	1174.80	2733.01	305.38
n = 75					
C	**w_i**	**Max(1)***	**Avg(1)***	**Max(2)****	**Avg(2)****
100	[1, 100]	5.02	2.28	5.02	2.28
150	[1, 100]	0.00	0.00	0.00	0.00
100	[30, 100]	4.84	2.67	4.84	2.67
150	[30, 100]	9000.00	3614.14	129.50	38.85
n = 100					
C	**w_i**	**Max(1)***	**Avg(1)***	**Max(2)****	**Avg(2)****
100	[1, 100]	6788.10	1251.52	6788.10	1251.52
150	[1, 100]	0.00	0.00	0.00	0.00
100	[30, 100]	9000.00	6307.86	8.48	6.11
150	[30, 100]	9000.00	3707.62	661.46	173.23

* CPU times in seconds of all problems
** CPU times in seconds of all optimally solved problems

In Table 5.17 the CPU times for the maximum overdeviation problem are presented. In the Max(1) column, it can be seen that for all the problem sets there is at least one unsolved problem. The results show that as the number of items (so the number of variables) increases; CPU times increase.

As the minimum weight increases; CPU times also increase. For all three problem sets, when the gap between the minimum weighted item and the maximum weighted item is smaller; it is more difficult to assign the items to the bins.

The results revealed that the capacity of the bins does not have any effect on the solution times.

Table 5.17. The CPU times for the maximum overdeviation problem

n = 50					
C	**w_i**	**Max(1)***	**Avg(1)***	**Max(2)****	**Avg(2)****
100	[1, 100]	9000.00	347.86	6274.20	138.35
150	[1, 100]	9000.00	206.43	988.97	39.27
100	[30, 100]	9000.00	2260.60	8881.79	561.68
150	[30, 100]	9000.00	3158.54	8808.40	586.51
n = 75					
C	**w_i**	**Max(1)***	**Avg(1)***	**Max(2)****	**Avg(2)****
100	[1, 100]	9000.00	729.92	8183.57	181.79
150	[1, 100]	9000.00	732.60	8268.75	282.11
100	[30, 100]	9000.00	3773.01	8323.06	342.33
150	[30, 100]	9000.00	4210.34	8974.81	662.16
n = 100					
C	**w_i**	**Max(1)***	**Avg(1)***	**Max(2)****	**Avg(2)****
100	[1, 100]	9000.00	2348.00	8628.23	451.77
150	[1, 100]	9000.00	1384.89	7933.46	351.43
100	[30, 100]	9000.00	4740.86	8643.84	450.51
150	[30, 100]	9000.00	5136.67	8965.34	613.09

* CPU times in seconds of all problems
** CPU times in seconds of all optimally solved problems

Table 5.18 shows that when the time limit is set to 9000 seconds, two more problems are solved optimally in the problem set where the capacity is 100, number of items is 100 and item weights are generated from the range [1,100].

Table 5.18. The number of unsolved instances for the number of bins problem when time limit is 9000 seconds

		# of unsolved instances*		
C	**w_i**	**n = 50**	**n = 75**	**n = 100**
100	[1, 100]	0	0	0
150	[1, 100]	0	0	0
100	[30, 100]	3	0	7
150	[30, 100]	1	4	4

*Out of 10 problem instances

In Table 5.19, the upper bounds on the number of efficient solutions for maximum overdeviation problem are presented. The maximum and average number of efficient solutions per problem instance are reported. Recall that 10 instances are solved for each problem set.

Table 5.19. The number of efficient solutions for maximum overdeviation problem

n = 50				
C	**w_i**	**Upper bound on the # of efficient solutions**	**Max # of efficient solutions per problem instance**	**Avg # of efficient solutions per problem instance**
100	[1, 100]	254	30	25.40
150	[1, 100]	161	19	16.10
100	[30, 100]	358	39	35.80
150	[30, 100]	213	23	21.30
n = 75				
C	**w_i**	**Upper bound on the # of efficient solutions**	**Max # of efficient solutions per problem instance**	**Avg # of efficient solutions per problem instance**
100	[1, 100]	403	44	40.30
150	[1, 100]	252	26	25.20
100	[30, 100]	538	56	53.80
150	[30, 100]	318	33	32.00
n = 100				
C	**w_i**	**Upper bound on the # of efficient solutions**	**Max # of efficient solutions per problem instance**	**Avg # of efficient solutions per problem instance**
100	[1, 100]	524	61	52.40
150	[1, 100]	336	36	33.60
100	[30, 100]	736	81	73.60
150	[30, 100]	431	46	43.10

Table 5.20 shows that the upper bound on the number of efficient solutions, number of exact solutions found for the maximum overdeviation problem. The effect of the number of items on the number of exact solutions found is clearly seen in the table. As the number of items increases; the percentage of exact solutions found decreases. The increase in the minimum weight from 1 to 30 also decreases the number of exact solutions found by the model. But the effect of capacity on the number of exact solutions is not clear.

The upper bound on the number of efficient solutions and the number of exact solutions found are reported in Table 5.20 There are 19 dominated solutions for the maximum overdeviation problem. The number of nondominated solutions is reported as the number of exact solutions. The number of dominated solutions is 1 for 50 items, 5 for 75 items and 13 for 100 items.

Table 5.20. Results on the efficient solutions for the maximum overdeviation problem

n = 50				
C	**w_i**	**Upper bound on the # of efficient solutions**	**# of exact solutions found**	**% of exact solutions found**
100	[1, 100]	254	247	98
150	[1, 100]	161	158	98
100	[30, 100]	358	286	80
150	[30, 100]	213	148	69
n = 75				
C	**w_i**	**Upper bound on the # of efficient solutions**	**# of exact solutions found**	**% of exact solutions found**
100	[1, 100]	403	377	93
150	[1, 100]	252	239	95
100	[30, 100]	538	320	60
150	[30, 100]	318	183	58
n = 100				
C	**w_i**	**Upper bound on the # of efficient solutions**	**# of exact solutions found**	**% of exact solutions found**
100	[1, 100]	524	404	78
150	[1, 100]	336	296	88
100	[30, 100]	736	358	50
150	[30, 100]	431	199	46

In Table 5.21 the quality of the model solutions for the unsolved problems of maximum overdeviation problem is presented. The number of the model solutions and average and maximum percentage deviations from lower and upper bounds are reported. The model can find a solution to 1273 out of 1290 in 9000 seconds. The solutions are close to the lower bounds for all problems.

Table 5.21. The performance of the model solutions for unsolved problems – Maximum Overdeviation Problem

n = 50							
C	**w_i**	**Upper bound on # of unsolved problems**	**# of solution found by model to unsolved problems**	**Avg % deviation from UB3**	**Max % deviation from UB3**	**Avg % deviation from LB3**	**Max % deviation from LB3**
100	[1, 100]	6	6	14.97	50.00	5.68	25.00
150	[1, 100]	3	3	80.97	216.67	6.71	16.67
100	[30, 100]	72	72	20.68	68.75	2.66	31.82
150	[30, 100]	65	65	80.16	525.00	7.25	50.00
n = 75							
C	**w_i**	**Upper bound on # of unsolved problems**	**# of solution found by model to unsolved problems**	**Avg % deviation from UB3**	**Max % deviation from UB3**	**Avg % deviation from LB3**	**Max % deviation from LB3**
100	[1, 100]	26	26	13.37	29.03	1.91	8.11
150	[1, 100]	13	13	34.18	100.00	10.54	50.00
100	[30, 100]	213	209	22.38	76.92	7.00	92.86
150	[30, 100]	135	132	101.98	1400.00	9.46	66.67
n = 100							
C	**w_i**	**Upper bound on # of unsolved problems**	**# of solution found by model to unsolved problems**	**Avg % deviation from UB3**	**Max % deviation from UB3**	**Avg % deviation from LB3**	**Max % deviation from LB3**
100	[1, 100]	116	116	19.64	400.00	4.08	50.00
150	[1, 100]	40	40	30.98	200.00	5.85	33.33
100	[30, 100]	369	365	17.86	71.43	7.39	92.86
150	[30, 100]	232	226	61.30	800.00	10.77	75.00

Table 5.22 shows the performance of the lower and upper bounds for the maximum overdeviation problem. On the average, the lower bound deviates 16.04% from the upper bound. For 9.22% of the problems upper bound is equal to the lower bound, hence no optimization effort is needed.

When the item weights are generated in range [30, 100], the gaps between the bounds are larger. For example, in the problem sets where item weights are generated from range [30, 100]; the average percentage deviation of the lower bound from the

upper bound is 19.51% whereas in the problem sets where item weights are generated from range [1,100]; the average percentage deviation is 11.59%.

When the capacity is 150; the bounds perform better and get closer. The average percentage deviation for problems with capacity of 150 is 14.00% whilst it is 17.37% when the capacity is 100. The number of items seems to have no significant effect on the performance of the bounds.

Table 5. 22. The lower and upper bound values for the maximum overdeviation problem

n = 50									
C	**w_i**	**# of solutions (LB3≠0)**	**Avg LB3**	**Max LB3**	**Avg UB3**	**Max UB3**	**# of solutions UB3= LB3**	**Avg % deviation**	**Max % deviation**
100	[1, 100]	241	288.04	2683	292.66	2683	26	12.39	89.47
150	[1, 100]	161	376.79	2765	380.80	2765	29	9.33	75.00
100	[30, 100]	316	312.96	3378	326.66	3378	19	20.25	92.86
150	[30, 100]	210	411.16	3286	425.26	3286	18	18.00	93.75
n = 75									
C	**w_i**	**# of solutions (LB3≠0)**	**Avg LB3**	**Max LB3**	**Avg UB3**	**Max UB3**	**# of solutions UB3= LB3**	**Avg % deviation**	**Max % deviation**
100	[1, 100]	380	327.93	3906	333.05	3906	47	13.13	87.50
150	[1, 100]	252	433.43	3856	436.94	3856	47	9.17	85.71
100	[30, 100]	472	349.92	4926	364.02	4926	15	20.81	93.33
150	[30, 100]	314	466.18	4831	479.75	4831	24	17.36	95.00
n = 100									
C	**w_i**	**# of solutions (LB3≠0)**	**Avg LB3**	**Max LB3**	**Avg UB3**	**Max UB3**	**# of solutions UB3= LB3**	**Avg % deviation**	**Max % deviation**
100	[1, 100]	497	354.78	5623	360.09	5623	39	13.40	91.67
150	[1, 100]	336	475.90	5317	479.84	5317	53	9.49	77.78
100	[30, 100]	635	378.28	6707	392.76	6707	37	20.94	94.44
150	[30, 100]	426	508.49	6654	522.84	6654	37	17.74	95.45

In Tables 5.23 and 5.24, the bounds are compared with the exact solutions for the problems that can be solved in 9000 seconds.

Table 5.23 shows the performance of the upper bound with respect to the exact solution for the maximum overdeviation problem. For 958 out of 3215 problems, i.e., 29.80% of problems, the exact solution is equal to the upper bound. The average percentage deviation of the upper bound from the exact solution is 9.50%

The performance of the upper bound gets worsen when the number of items is 100. It deviates from the exact solution by 14.19% when there are 50 items; 7.76% when there are 75 items and 35.80% when there are 100 items.

When the capacity is 100, the performance of the upper bound is better and is equal to the exact solution in 37.65% of the problems, the average percentage deviation from the exact solution is 6.31%. When the capacity is 150, the upper bound is equal to the exact solution for 17.01% of the problems and average percentage deviation from the exact value is 14.69% in problems. The average percentage deviation of upper bounds is 36.00% when minimum weight is 1, 24.40% when minimum weight is 30. It is observed that as the complexity of the problem increases, the possibility of finding the exact solution via the upper bound increases but the average percentage deviation also increases.

Table 5.23. The upper bound values on the maximum overdeviation and the optimal maximum overdeviation

n = 50									
C	**w_i**	**# of exact solutions found**	**Avg UB3**	**Max UB3**	**Avg exact**	**Max exact**	**# of solns UB3= exact**	**Avg % deviation**	**Max % deviation**
100	[1, 100]	247	284.22	2683	281.08	2683	64	7.24	87.50
150	[1, 100]	158	386.76	2765	382.87	2765	29	16.89	300.00
100	[30, 100]	286	324.17	3378	336.37	3378	120	8.20	75.00
150	[30, 100]	148	566.24	3286	579.07	3286	18	34.48	1500.00
n = 75									
C	**w_i**	**# of exact solutions found**	**Avg UB3**	**Max UB3**	**Avg exact**	**Max exact**	**# of solns UB3= exact**	**Avg % deviation**	**Max % deviation**
100	[1, 100]	377	325.49	3906	330.34	3906	124	6.52	75.00
150	[1, 100]	239	455.08	3856	458.45	3856	47	15.70	600.00
100	[30, 100]	320	460.22	4926	471.14	4926	147	4.75	73.53
150	[30, 100]	183	765.02	4831	776.64	4831	24	5.20	33.82
n = 100									
C	**w_i**	**# of exact solutions found**	**Avg UB3**	**Max UB3**	**Avg exact**	**Max exact**	**# of solns UB3= exact**	**Avg % deviation**	**Max % deviation**
100	[1, 100]	404	416.47	5623	421.06	5623	103	8.84	300.00
150	[1, 100]	296	534.83	5317	538.45	5317	53	14.27	350.00
100	[30, 100]	358	583.80	6707	595.73	6707	192	2.47	59.46
150	[30, 100]	199	1009.63	6654	1022.39	6654	37	6.38	500.00

Table 5.24 presents the performance of the lower bound with respect to the exact solution. The lower bound for the maximum overdeviation problem performs much better than the lower bound for the total overdeviation problem (see Table 5.15). The average percentage deviation of the lower bound for the maximum overdeviation problem is 6.79%. In 82.31% of the problems, the lower bound is equal to the exact value which means the lower bound provides a quite good estimate for the maximum overdeviation problem.

It is seen that the bound is better for the problems where the capacity is 150. In 1217 out of 1219 exact solutions, the lower bounds are equal to the optimal maximum

overdeviations. The average percentage deviation from the optimal maximum overdeviation is 0.09% for the capacity of 150. For problems with the capacity of 100, the lower bound behaves as expected and its performance is better when minimum weight is 1. The average percentage deviation is 7.12% when the item weights are generated from the range [1,100] whereas it increases to 16.16% when item weights are generated from the range [30,100] for capacity of 100.

Compared with Table 5.23 it is observed that for the maximum overdeviation problem, the performance of the lower bound is better than the performance of the upper bound.

Table 5.24. The lower bound values on the maximum overdeviation and the optimal maximum overdeviation

n = 50									
C	**w_i**	**# of exact solutions (LB3±0)**	**Avg LB3**	**Max LB3**	**Avg exact**	**Max exact**	**# of solutions LB3= exact**	**Avg % deviation**	**Max % deviation**
100	[1, 100]	236	293.85	2683	295.14	2683	184	6.78	89.47
150	[1, 100]	158	382.87	2765	382.87	2765	158	0.00	0.00
100	[30, 100]	244	379.97	3378	383.43	3378	143	14.62	92.86
150	[30, 100]	145	577.95	3286	577.99	3286	143	0.84	71.43
n = 75									
C	**w_i**	**# of exact solutions (LB3±0)**	**Avg LB3**	**Max LB3**	**Avg exact**	**Max exact**	**# of solutions LB3= exact**	**Avg % deviation**	**Max % deviation**
100	[1, 100]	354	346.64	3906	347.72	3906	268	7.68	87.50
150	[1, 100]	239	455.08	3856	455.08	3856	239	0.00	0.00
100	[30, 100]	269	556.02	4926	559.07	4926	173	15.35	93.33
150	[30, 100]	183	765.02	4831	765.02	4831	183	0.00	0.00
n = 100									
C	**w_i**	**# of exact solutions (LB3±0)**	**Avg LB3**	**Max LB3**	**Avg exact**	**Max exact**	**# of solutions LB3= exact**	**Avg % deviation**	**Max % deviation**
100	[1, 100]	386	440.21	5623	441.32	5623	307	6.82	91.67
150	[1, 100]	296	534.83	5317	534.83	5317	296	0.00	0.00
100	[30, 100]	322	665.38	6707	669.30	6707	202	17.99	94.44
150	[30, 100]	198	1014.73	6654	1014.73	6654	198	0.00	0.00

CHAPTER 6

CONCLUSIONS

In this thesis, we consider two bicriteria problems. The first problem takes the number of bins and total overdeviation as two criteria, whereas the second problem takes the number of bins and maximum overdeviation as two criteria.

We formulate the problems as mixed integer linear programming (MILP). Using these programs, we generate the set of nondominated vectors with respect to define criteria. We enhance the efficiency of the MILPs through lower and upper bounding mechanisms. For total overdeviation problem, we define a property that finds many nondominated solutions without using MILPs.

The results of our experiments have revealed that the problem with up to 100 items could be solved for high capacity bins. The problems with up to 75 items could be solved when the capacity is low.

Computational results show that the quality of the upper bound is better than that of the lower bound in the total overdeviation problem whereas the quality of lower bound is better than that of the upper bound in the maximum overdeviation problem. The upper bound for the total overdeviation problem is equal to the optimal value for 76.65% of the problems while the lower bound for the maximum overdeviation problem is equal to the optimal maximum overdeviation for 82.31% of the problems.

To the best of our knowledge, our study is the first bin packing study that considers deviation based measures. Further research may be directed towards development of enumeration and heuristic algorithms. Moreover different deviation based measures like minimizing total squared workload might be dealt.

REFERENCES

[1] Alvim, A., Ribeiro, C., Glover, F., & Aloise, D. (2004). A Hybrid Improvement Heuristic for the One-Dimensional Bin Packing Problem. *Journal of Heuristics, 10*, 205-229.

[2] Bruno, J. L., & Downey, P. J. (1985). Probabilistic bounds for dual bin-packing.*Acta Informatica, 22*(3), 333-345.

[3] Brusco, M., Köhn, H., & Steinley, D. (2012). Exact and approximate methods for a one-dimensional minimax bin-packing problem. *Annals of Operations Research ,206,* 611-626.

[4] Brusco, M., Thompson, G., & Jacobs, L. (1997). A morph-based simulated annealing heuristic for a modified bin-packing problem. *Journal of the Operational Research Society, 48*, 433-439.

[5] Chen, F., & Yao, E. (2001). Exact Algorithm For Bin Covering. *Journal of Zhejiang University Science Jzus, 2(3)*, 241-246.

[6] Coffman Jr, E. G., Leung, J. T., & Ting, D. W. (1978). Bin packing: maximizing the number of pieces packed. *Acta Informatica, 9*(3), 263-271.

[7] Csirik, J., Frenk, J. B., Labbé, M., & Zhang, S. (1999). Two simple algorithms for bin covering. *Acta Cybernetica*, *14*(1), 13-25.

[8] de Carvalho, J. V. (1999). Exact solution of bin-packing problems using column generation and branch-and-bound. *Annals of Operations Research, 86*, 629-659.

[9] Eilon, S., & Christofides, N. (1971). The loading problem. *Management Science,17*(5), 259-268.

[10] Fleszar, K., & Hindi, K. (2002). New heuristics for one-dimensional bin-packing. *Computers & Operations Research, 29,* 821-839.

[11] Foster, D. P., & Vohra, R. V. (1989). Probabilistic analysis of a heuristics for the dual bin packing problem. *Information Processing Letters, 31*(6), 287-290.

[12] Garey, M.R. Johnson, D.S. (1979) *Computers and Intractability: A guide to the Theory of NP-Completeness*, New York : W. H. Freeman and Company.

[13] Geiger, M. J. (2008). Bin packing under multiple objectives-a heuristic approximation approach. *Fourth International Conference on Evolutionary Multi-Criterion Optimization, Matsushima, Japan,* 53-56.

[14] Graham, R. (1969). Bounds on Multiprocessing Timing Anomalies. *SIAM Journal on Applied Mathematics, 17,* 416-429.

[15] Gupta, J.N.D, & Ho, J.C. (1999). A new heuristic algorithm for the one-dimensional bin-packing problem. *Production Planning & Control, 10*, 598-603.

[16] Hung, M. S., & Brown, J. R. (1978). An algorithm for a class of loading problems. *Naval Research Logistics Quarterly*, *25*(2), 289-297.

[17] Johnson, D. (1974). Fast algorithms for bin packing. *Journal of Computer and System Sciences, 8,* 272-314.

[18] Johnson, D., Demers, A., Ullman, J., Garey, M., & Graham, R. (1974). Worst-Case Performance Bounds for Simple One-Dimensional Packing Algorithms. *SIAM Journal on Computing, 3(4),* 299-325.

[19] Karasakal, E., Azizoglu, M., & Ilicak I. (2014). Bi-objective Bin Packing Problems: Exact Approaches to Generate All Efficient Solutions. *Technical Report No: 14-01, Middle East Technical University, Ankara.*

[20] Kim, B., & Wy, J. (2010). Last two fit augmentation to the well-known construction heuristics for one-dimensional bin-packing problem: An empirical study. *The International Journal of Advanced Manufacturing Technology, 50*, 1145-1152.

[21] Labbé, M., Laporte, G., & Martello, S. (1995). An exact algorithm for the dual bin packing problem. *Operations Research Letters*, *17*(1), 9-18.

[22] Labbé, M., Laporte, G., & Martello, S. (2003). Upper bounds and algorithms for the maximum cardinality bin packing problem. *European Journal of Operational Research, 149*, 490-498.

[23] Loh, K., Golden, B., & Wasil, E. (2008). Solving the one-dimensional bin packing problem with a weight annealing heuristic. *Computers & Operations Research, 35*, 2283-2291.

[24] Martello, S., & Toth, P. (1990). *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Chichester, England.

[25] Mezghani, S., Chabchoub, H., & Aouni, B. (2013). Manager's preferences in the Bi-Objectives Bin Packing Problem. In *Modeling, Simulation and Applied Optimization, 5th International Conference*, 1-4. IEEE.

[26] Mladenović, N., & Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research, 24*, 1097-1100.

[27] Naderi, B., & Yazdani, M. (2014). A Real Multi-Objective Bin Packing Problem: A Case Study of an Engine Assembly Line. *Arabian Journal for Science and Engineering*, *39*(6), 5271-5277.

[28] Patel, K., & Panchal, M. (2014). One-Dimension Multi-Objective Bin Packing Problem using Memetic Algorithm.

[29] Peeters, M., & Degraeve, Z. (2006). Branch-and-price algorithms for the dual bin packing and maximum cardinality bin packing problem. *European Journal of Operational Research*, *170*(2), 416-439.

[30] Pérez, J., Castillo, H., Vilariño, D., Zavala, J. C., De la Rosa, R., & Ruiz-Vanoye, J. A. (2015). A hybrid algorithm with reduction criteria for the bin packing problem in one dimension. In *Proceedings of the International Conference on Numerical Analysis and Applied Mathematics 2014 (icnaam-2014)* (Vol. 1648, p. 820003). AIP Publishing.

[31] Rao, R. L., & Iyengar, S. S. (1994). Bin-packing by simulated annealing.*Computers & Mathematics with Applications*, *27*(5), 71-82.

[32] Ross, P., Marín-Blázquez, J., Schulenburg, S., & Hart, E. (2003). Learning a Procedure That Can Solve Hard Bin-Packing Problems: A New GA-Based Approach to Hyper-heuristics. *Genetic and Evolutionary Computation 2003, Lecture Notes in Computer Science, 2724*, 1295-1306.

[33] Scholl, A., Klein, R., & Jürgens, C. (1997). BISON: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers & Operations Research*, *24*(7), 627-645.

[34] Singh, A., & Gupta, A. (2006). Two heuristics for the one-dimensional bin-packing problem. *OR Spectrum, 29*, 765-781.

[35] Stawowy, A. (2008). Evolutionary based heuristic for bin packing problem. *Computers & Industrial Engineering, 55*, 465-474