HYBRID METAHEURISTIC ALGORITHMS FOR SINGLE AND MULTI-OBJECTIVE 2D BIN PACKING PROBLEM

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

BY

MUHAMMED BEYAZ

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

JANUARY 2015

Approval of the thesis:

HYBRID METAHEURISTIC ALGORITHMS FOR SINGLE AND MULTI-OBJECTIVE 2D BIN PACKING PROBLEM

submitted by **MUHAMMED BEYAZ** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

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

Prof. Dr. Adnan Yazıcı
Head of Department, **Computer Engineering** _____

Prof. Dr. Ahmet Coşar
Supervisor, **Computer Engineering Department, METU** _____

Dr. Tansel Dökeroğlu
Co-supervisor, **Computer Engineering Department, METU** _____

Examining Committee Members:

Prof. Dr. Özgür Ulusoy
Computer Engineering Dept., Bilkent University _____

Prof. Dr. Ahmet Coşar
Computer Engineering Dept., METU _____

Assoc. Prof. Dr. Pınar Karagöz
Computer Engineering Dept., METU _____

Assoc. Prof. Dr. Ertan Onur
Computer Engineering Dept., METU _____

Asst. Prof. Dr. Selim Temizer
Computer Engineering Dept., METU _____

Date: _____

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: MUHAMMED BEYAZ

Signature :

ABSTRACT

HYBRID METAHEURISTIC ALGORITHMS
FOR SINGLE AND MULTI-OBJECTIVE 2D BIN PACKING PROBLEM

Beyaz, Muhammed
M.S., Department of Computer Engineering
Supervisor : Prof. Dr. Ahmet Coşar
Co-Supervisor : Dr. Tansel Dökeroğlu

January 2015, 86 pages

2D Bin packing problem (2DBPP) is an NP-hard combinatorial optimization problem. Objects with different width and length sizes are packed in order to minimize the number of unit-capacity bins according to an objective function. Single or multi-objective versions of this well-known industrial engineering problem can be faced frequently in real life situations. There have been several heuristics proposed for the solution of 2DBPP until now where it is not possible to find the exact solutions for large problem instances. Next fit, First Fit, Best Fit, Unified Tabu Search and Genetic Algorithms are some of these algorithms. Recently, Memetic Algorithms have put themselves forward as a new area of research in evolutionary computation with their ability to combine different heuristics and local search mechanisms together for higher quality solutions. In this thesis, we propose a set of single and multiobjective memetic and genetic algorithms that make use of the state-of-the-art metaheuristics and local search techniques for the solution of 2DBPP. We analyze the optimization time and the resulting solution quality of the algorithms on a 2DBPP benchmark offline problem set with 500 instances. Through results of exhaustive experiments and with the aid of a novel visual analyzer developed in this study, we conclude that the proposed memetic and hybrid genetic algorithms are robust with their ability to obtain very high percentage of the optimal solutions for the given benchmark problem instances.

Keywords: 2D Bin packing, memetic, genetic, load balancing, multiobjective, heuristic

ÖZ

TEK VE ÇOK AMAÇLI İKİ BOYUTLU KUTU PAKETLEME PROBLEMİ İÇİN MELEZ METASEZGİSEL ALGORİTMALAR

Beyaz, Muhammed
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Ahmet Coşar
Ortak Tez Yöneticisi : Dr. Tansel Dökeroğlu

Ocak 2015 , 86 sayfa

İki boyutlu kutu paketleme problemi çözümü polinom zamanlı olmayan (NP) kombinatoriyal bir problemdir. Farklı genişlik ve uzunluktaki nesnelerin en az kutu kaplayacak şekilde birim kapasiteli kutulara bir nesnel fonksiyonun en az sonuç üretecek şekilde yerleştirilmesidir. Çok bilinen bu endüstri probleminin çok amaçlı versiyonlarıyla gündelik hayatta sıkça karşılaşılmaktadır. Kesin çözüm bulmanın mümkün olmadığı büyük ölçekli 2 boyutlu kutu paketleme problemleri için şimdiye kadar birkaç sezgisel yöntem önerilmiştir. Sonraki Ekleme, İlk Ekleme, En İyi Ekleme, Birleşik Tabu Araması ve Genetik Algoritmalar bu algoritmaların en çok kullanılan birkaçıdır. Son zamanlarda, farklı sezgisel ve lokal aramaları birleştirerek daha yüksek kalitede sonuç üreten Memetik Algoritmalar evrimsel hesaplamalarda yeni bir araştırma alanı olmuştur. Bu tezde, 2 boyutlu kutu paketleme problemi için metasezgisel ve lokal aramayı kullanan tek ve çok amaçlı memetik ve genetik algoritmalardan oluşan bir çözüm kümesi sunduk. 500 problem içeren çevrimdışı bir problem kümesi ile algoritmalarımızın optimizasyon zamanlarını ve sonuç kalitelerini analiz ettik. Bu projede geliştirdiğimiz görsel analiz aracı ile de yapılan yoğun testler sonucunda, geliştirdiğimiz memetik ve genetik algoritmaların kullanılan problem kümeleri üzerinde yüksek oranda optimum sonuç ürettiği neticesine vardık.

Anahtar Kelimeler: 2B Kutu paketleme, memetik, genetik, yük dengeleme, çok amaç, sezgisel

To my family and people who are reading this page

ACKNOWLEDGMENTS

Many thanks to my supervisor, Prof. Dr. Ahmet Coşar, for spending his months and effort to propose this study. It has been a great privilege to have been mentored by such an esteemed researcher.

Special thanks to Dr. Tansel Dökeroğlu who provided encouragement for this study and also for his valuable insight and his guidance about this study.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

1DBPP	One-Dimensional Bin Packing Problem
2DBPP	Two-Dimensional Bin Packing Problem
3DBPP	Three-Dimensional Bin Packing Problem
AIS	Artificial Immune System
AM	Ahuja and Murty
AM-H	Ahuja and Murty Tree Improvement Heuristic
BFB	Best Fit Bin
BF	Best Fit
BFD	Best Fit Decreasing
BFDH	Best Fit Decreasing Height
BRKGA	Biased Random-Key Genetic Algorithm
CG	Center of Gravity
CVC	California Vehicle Code
DD	Due Dates
DFTRC	Distance to the Front-Top-Right Corner
DG	Directed Graph
DJD	Djang and Finch
EA	Evolutionary Algorithm
FBSP	Finite Best Strip Packing
FF	First Fit
FFD	First Fit Decreasing
FFDH	First Fit Decreasing Height
FFF	Finite First Fit
FG-O-2DBPP	Free Guillotine Oriented Two-Dimensional Bin Packing Problem
FNF	Finite Next Fit
GA	Genetic Algorithm
GLS	Genetic Local Search
GRASP	Greedy Randomized Adaptive Search Procedure
HEA	Hybrid Evolutionary Algorithm
LB	Load Balancing
LGF	Lowest Gap Fill
LGFi	Improved Lowest Gap Fill
LGFof	Oriented Improved Lowest Gap Fill
LP	Linear Programming
MA	Memetic Algorithm

MH-MOGA	Multi-Heuristic Multi-Objective Genetic Algorithm
MH-SOMA	Multi-Heuristic Single Objective Memetic Algorithm
MHNO-MOGA	Multi-Heuristic Non-Oriented Multi-Objective Genetic Algorithm
MHNO-SOMA	Multi-Heuristic Non-Oriented Single Objective Memetic Algorithm
MHO-MOGA	Multi-Heuristic Oriented Multi-Objective Genetic Algorithm
MHO-SOMA	Multi-Heuristic Oriented Single Objective Memetic Algorithm
MOGA	Multi-Objective Genetic Algorithm
MXGA	Multi-Crossover Genetic Algorithm
NFD	Next Fit Decreasing
NFDH	Next Fit Decreasing Height
NO-2DBP	Non-Oriented Two-Dimensional Bin Packing
NO-MOGA	Non-Oriented Multi-Objective Genetic Algorithm
NO-SOMA	Non-Oriented Single Objective Memetic Algorithm
O-2DBP	Oriented Two-Dimensional Bin Packing
O-BBP	Oriented Bin Packing Problem
O-MOGA	Oriented Multi-Objective Genetic Algorithm
O-SOMA	Oriented Single Objective Memetic Algorithm
OnBP	Online Bin Packing
OffBP	Offline Bin Packing
PCA	Principle Component Analysis
PSO	Particle Swarm Optimization
QAP	Quadratic Assignment Problem
RSAM	Random Sampling Ahuja and Murty
SR	Swap and Rotate
SA	Simulated Annealing
SO-2DBPP	Single Objective Two Dimensional Bin Packing Problem
SOMA	Single Objective Memetic Algorithm
TS	Tabu Search
TSP	Travelling Sales Person
UTS	Unified Tabu Search

CHAPTER 1

INTRODUCTION

Given a set of rectangular items and a 2D bin of fixed width and variable length, the two-dimensional bin packing problem (2DBPP) consists of orthogonally placing all the pieces within the bin, without overlapping, such that the overall number of the bins is minimised [1, 2, 3, 4, 5]. The 2DBPP is an intractable optimization problem and widely faced during the industrial manufacturing processes. Textile manufacturing and newspaper page design are some of these optimization areas.

Online and offline are two different categories of 2DBPP according to the availability, at the beginning, of information about input. Online bin packing (OnBP) means that objects arrive one by one and there is no way to know complete input sequence, so it must be inserted into a bin immediately without waiting other objects. Hard drive partitioning for online storage systems usually deals with these types of problems. Offline bin packing (OffBP) means that all of the objects are known before they are packed so they can be reordered according to insertion heuristics. Plane load planning is a typical OffBPP.

Orientation is another key aspect of the 2DBPP. Rotating objects in packing creates better results, but objects may be not rotatable. Textile industry can change the orientation of single color shirts by rotating the shirts while the process is in cutting phase, because there is no difference between rotation or not. But shipping industry consider the orientation of fragile items.

In this thesis we study mechanisms, which are inspired by natural evolution, known as Evolutionary Algorithms (EA). Reproduction, mutation, recombination and selec-

tion are key mechanisms of EA that are used to solve optimization problems. Bin Packing, Travelling Sales Person and Quadratic Assignment Problem [5, 6] are well-known problems that are solved with EAs. Genetic Algorithm (GA) and Memetic Algorithm (MA) are the most well-known approaches of EAs. The objective of these optimization problems can be single or multi-objective. GA mimics the natural evolution process and has the ability to find optimal solution in a large search space. In nature, survival of the fittest is a rule allowing the fittest solutions in each iteration to converge a (near-) optimal solution in a short time. In GA, the fittest of individual is the solution of optimization problems:

- Parents mate and produce offsprings
- Some of individuals are mutated
- Best individuals are selected to survive to the next generation

MA is another growing are of EA. It mimics natural evolution process but it may differ from GA by performing individual learning which is also known as meme(s). As in GA, MA applies below list to find (near-) optimal solution of optimization problems:

- Parents mate and produce offsprings
- Some of individuals are mutated
- Individual learning is performed (local search)
- Best individuals are selected to survive

In this thesis, we propose four different techniques to solve two different 2D offline BPPs. In each technique, we use some of well known heuristics: Finite Next Fit (FNF), Finite First Fit (FFF), Best Fit Decreasing Height (BFDH), Unified Tabu Search(UTS)Improved Left Gap Fill (LGFi) and oriented Improved Left Gap Fill (LGFof).

The objective of our first problem is minimizing the number of bins. Orientation is crucial part of bin packing problems so we developed two different multi-heuristic single objective memetic algorithms (MH-SOMA). Our first algorithm, MHO-SOMA, uses heuristics: FNF, FFF, BFDH, UTS and LGFof. It tries to minimize the number

of bins for orientated 2D offline BPP. Our second algorithm, MHNO-SOMA, uses heuristics: FNF, FFF, BFDH, UTS, LGFof and LGFi in order to minimize the number of bins for non-orientated 2D offline BPP.

Our second optimization problem tries to find a pareto-optimal solution for both the minimal number of bins and the most efficiently load-balanced placing of the rectangular items. We propose two different multi-heuristic multi-objective genetic algorithms (MH-MOGA). The first proposed multi-objective technique, MHO-MOGA, uses heuristics: FNF, FFF, BFDH, UTS and LGFof to solve oriented multi objective 2D offline BPPs. The second proposed multi-objective technique, MHNO-MOGA, uses heuristics: FNF, FFF, BFDH, UTS, LGFof and LGFi. It tries to find pareto-optimal solution (minimal number of bins and most effective load-balancing of items) for non-oriented multi objective 2D offline BPPs.

Through exhaustive experiments with 500 benchmark problem instances, we analyze the performance of our novel algorithms.

CHAPTER 2

RELATED WORK

In this chapter, we give information about the well known metaheuristics/heuristics such as Next Fit, First Fit, Best Fit, Unified Tabu Search, Left Gap Fill, Genetic Algorithm and Memetic Algorithm which are used to solve 2DBPP.

Next Fit is a level-oriented packing heuristic. It keeps a current level and tries to pack item into current level, if item does not fit then it creates a new level as current level and insert the item into current level.

First Fit is another level-oriented packing heuristic. It keeps all levels in all bins and tries to pack an item into first available level, if the item does not fit into any level then it creates new level and insert the item into new level

Best Fit is another level-oriented packing heuristic. It keeps all levels in all bins. First it calculates remaining gaps of all level even if we pack the item to that level and then tries to pack the item into minimum remaining gap level, if the item does not fit into any level then it creates new level and insert the item into new level.

Unified Tabu Search is a variant of Tabu Search Algorithm. First it packs the items into bins and then tries to replace items in order to minimize the number of bins. While minimizing the number of bins, it keeps a list known as tabu list and keeps track of last replacement movements.

Left Gap Fill tries to pack items into left most position of bin by calculating the best item which can fit into current gap.

Genetic Algorithm is an evolutionary algorithm. It tries to exploit the global search

space by mating best solution of problem. It uses inheritance, mutation, selection and crossover of natural evolution.

Memetic Algorithm is another evolutionary algorithm. It combines global search with an individual learning method.

2.1 Next Fit

Meir and Moser introduced a new packing algorithm in 1968 [32]. This algorithm packs the largest square cube into the left most corner of the hyberbox, and then it picks the second largest cube. It checks the remaining part of ground level of the hypercube. If the remaining part is enough then it inserts the second largest square cube into the position which it can touch the first largest cube. If the second largest cube cannot fit then it creates a new level with height of first largest cube and tries to insert the second largest cube at that level. This new introduced algorithm is known as Next Fit Decreasing (NFD) algorithm.

Coffman et al. studied on the NFD in order to pack the rectangles into rectangles or strips. This new extended algorithm is called Next Fit Decreasing Height (NFDH) algorithm [28].

Berkey and Wang enhanced Coffman's approach and proposed a new algorithm known ad Finite Next Fit (FNF). This level-oriened algorithm packs the item into finite bin set in one phase [35].

2.2 First Fit

Johnson et al. examined the good placement heuristics. One of the heuristics is known as First Fit (FF) [40]. In FF, items are packed into the first bin which has available space for item. First Fit Decreasing (FFD) algorithm is another variant of FF. It first sorts the items in descending order and then applies FF.

Coffman et al. applied non-increasing height sorting to FFD for 2DBPP and introduces a new algorithm First Fit Decreasing Height (FFDH) [28].

Berkey and Wang enhanced Coffman's first fit approach and proposed a new algorithm Finite First Fit (FFF) [35]. This level-oriened algorithm packs the item into finite bin set in one phase.

2.3 Best Fit

Johnson et al. described another good placement heuristics Best Fit (BF) algorithm [40]. It is similar to FF algorithm but items are inserted by remaining space in the bin. It calculates all the remaining spaces and inserts the item into bin which has smallest remaining space. Best Fit Decreasing (BFD) algorithm is an extended version of BF. First it sorts the items and then items are inserted.

Berkey and Wang enhanced BF and proposed a new algorithm Finite Best Strip Packing (FBSP) [35]. FBSP is a two level algorithm. At first level, items are inserted according to BF. At second level, prepared strip result is divided into levels and levels are merged into bins by using remaining horizontal space.

Coffman and Shor [23] described a new method similar to FBSP algorithm. It first sorts the items according to decreasing height and then inserts them by using BFD algorithm. This algorithm is Best Fit Decreasing Height (BFDH).

2.4 Unified Tabu Search

In 1986, Glover developed a new algorithm which is known as Tabu Search (TS) [19]. Glover described the details of algorithm [20, 21]. TS is composed of stopping condition, keeping tabu list, neighborhood search and aspiration condition. Stopping condition checks the endpoint of search. Tabu list keeps the track of last known moves in order to not to stuck in local optima. Neighborhood search is minor changes (swaps) of items. Aspiration condition or fitness function calculates the value of current search.

Lodi et al. developed new heuristics for different 2DBPP and used these heuristics within TS [11]. Lodi et al. explained the detail of their Unified Tabu Search (UTS)

algorithm [12] .

2.5 Left Gap Fill

Burke et al. introduced a new best fit strategy in order to pack 2D rectangles into strip [29]. New heuristic is using dynamic selection of best fit strategy in order to choose next rectangle. It used two stages processing of rectangles. First stage is preprocessing which rotates items whose height is greater than width and then sorts the item by decreasing width. Second stage it inserts items by using leftmost, tallest neighbor or shortest neighbor heuristics.

Bennell et al. enhanced the Burke strip packing strategy and used it to solve two dimensional bin packing problem [9]. It used the same preprocessing strategy and packing best fit rectangle into bin by using waste space insertion. This algorithm is known as Best Fit Bin (BFB). Lee changed BFB algorithm to Lowest Gap Fill (LGF) algorithm by using left most position search instead of waste space insertion [31]. Wong and Lee improved LGF into Improved Lowest Gap Fill (LGFi) by changing the shortest edge as remaining gap and also developed a new version of LGFi in order to pack oriented bin packing known as Oriented Improved Lowest Gap Fill (LGFof) [30].

2.6 Genetic Algorithms

Bremermann offers a mathematical model for solving problems by using gene (solutions), mutation and crossover [26]. This is the start point of Genetic Algorithms (GA). Holland extended GA and developed a framework for it [25]. This study made GA more applicable in computing enviroment. Goldberg introduced GA terminology and application areas [16]. Goldberg solves some problems in order to show the application of GA in a real problem. He also describes the implementation of the functionalities of genetic algorithm.

Smith tried to find the maximum number of rectangles that can be packed into a bin by using GA [14]. He used permutation of rectangles in order to encode the

chromosomes so the sequence of rectangles becomes important. The sequence gives the solution.

Hopper and Turton found solution to rotatable objects bin packing problems by use of BL and BLF heuristics [17]. He used order-based (permutation) encoding schema. Crossover and mutation generate valid chromosomes. There are two types of mutation operation. First one is order changing mutation and second one is rotation.

Kröger et al. creates a new vision of genetic algorithm encoding [13]. Directed Graph (DG) based encoding is used with top and right insertion technique. If a problem arises, then indices which shows priority are used to solve problem.

2.7 Recent Studies

Thapatsuwan et al. aim to find an optimal solution for multiple container packing problem [39]. Their study is composed of three different algorithms which are Artificial Immune System (AIS), GA and Particle Swarm Optimization (PSO). Result part of the study shows that according to found parameter sets AIS is better than GA and PSO but its run time is at least four times greater than GA.

Lim et al. encountered a practical problem California Vehicle Code (CVC) [10] . Containers are prepared by an oversea company and delivered in USA. Trucks has to obey axle weight constraints. A Greedy Randomized Adaptive Search Procedure (GRASP) is used in order to meet criteria.

Gonçalves and Resende presented a new way of 2DBP and 3DBP with Biased Random -Key Genetic Algorithm (BRKGA) [22]. He developed two different insertion heuristics which are based of Distance to the Front-Top-Right Corner (DFTRC) rule. DFTRC1 is calculating the distance according to the current presentation of box so it is available for oriented bin packing and DFTRC2 is calculating the distance for all the possible orientation of box so it is good to use it for orientation free problem.

Lopez-Camacho et al. suggested a heuristic and hyperheuristic method for packing problems [18]. Heuristics are FFD, Filler, BFD, Djang and Finch with initial fullness of $DJD\frac{1}{2}$, $DJD\frac{1}{3}$ and $DJD\frac{1}{2}$. Hyperheuristic finds the best combination of these

heuristic in order to solve one instance. GA is used to find hyperheuristics. Principle Component Analysis (PCA) is used to analyse BPPs.

Bennell et al. proposed a MultiCrossover Genetic Algorithm (MXGA) with BFB heuristic for NO-2DBPP with Due Dates (DD) [8]. BFB is a variant of BF heuristic and it has two phases. Phase one rotates the bins in order to maximize width and phase two sorts items with decreasing height. GA assigns the bin number of rectangles. Fitness function considers minimum number and minimum latency.

Fernández et al. considered rotation of items and Load Balancing (LB) of bins as multi-objective [4]. Multi-Objective Memetic Algorithm (MOMA) is developed. Insertion heuristic is made of two phase. First phase is using BLF heuristic and second phase is dividing bin into many bins in order to meet level criteria. Two types of local optimizers are used. First local optimizer tries to minimize the number of bins by trying the items from emptiest bin to insert to most occupied bin. Second local optimizer tries to gathers the items from most occupied zones of bins and insert them into new bins.

Blum and Schmid described an evolutionary algorithm (EA) for free guillotine oriented two-dimensional bin packing problem (FG-O-2DBPP) [3] . The proposed algorithm is based on LGFi heuristic. Crossover operator is a variant of uniform order-based crossover.

Bansal and Khan used Round and Approximation Framework to pack 2D rectangles with and without rotation [46] . The proposed technique divides rectangles into five group: big, wide, long, medium and small. It creates containers with wide and long group of rectangles. First, it packs big, containers and medium by brute force approach. Then, it packs small rectangles by Next Fit.

Recent studies shows that bin packing problems do not have exact solutions. Recent optimization techniques such as GA, MA, PSO and AIS are tried to be applied for packing problems as soon as possible. Real life applications of BPP are not just about minimizing the number of bins so recent studies not only try to minimize the number of bins but also tries to keep balance of bins. Industrial engineering is curious about the solution of BPP.

2.8 Memetic Algorithms

Dawkins unfolded a new idea of gene-centered view of evolution. He coined a new phrase “meme” which means a behavior is passed from one individual to another individual. Moscato inspired by the book “The Selfish Gene” and developed a term Memetic Algorithm (MA) [37] . MA is also known as Hybrid Evolutionary Algorithms (HEA). It is a hybridization of population-based search and local search technique. In this study, he used MA as a combination of GA and Simulated Annealing (SA) for Travelling Sales Person (TSP) and some other problems. Moscato and Cotta described terms about MA and gave guidelines about the MA implementation [38] .

MA uses two different learning methods. First one is Lamarckian Learning. In Lamarckian Learning, individuals learn during their lifetime and keep the values of local search improvement. In selection mechanism, fitness of individuals are evaluated according to their local search improvements, so best improved fitnesses are transferred to next generation with their improvement information in genotype. Second one is Baldwinian Learning. In Baldwinian Learning, local search is applied to individual and fitness value of individual is calculated by the use of local learning, but local improvements are not part of genotype, so local search improvements are passed indirectly to next generation. Whitley et al. argued that Lamarckian Learning is faster than Baldwinian Learning but results of Lamarckian Learning can converge to local optima faster than Baldwinian Learning [15]. In 2005, Yao et al. showed that there is no significant difference between Lamarckian and Lamarckian Learnings [45] .

Development of MA can be divided into three generations. First generation describes the basis of Memetic Algorithm. It introduced Memetic Algorithm as marriage between a global search techique with local learning. Hyper-heuristic [24], Meta-Lamarckian MA [42] and Multi-meme [33] are classfied as second generation of MA. In Hyper-heuristic and Meta-Lamarckian MA, a reward mechanism is applied to individuals. Past values of memes are considered for reward value and individuals with higher reward values have more chances to survive and be replicated. In Multi-Meme, memes are considered as a part of individual (genotype) so inheritance is used to passed memetic behavior of individuals to their offsprings. Third generation of MA

uses Co-evolution [27] and self-generation [34]. Rule-based local search is applied to heuristic so repeated patterns are captured and individuals are nourished by captured patterns.

Fischer and Merz developed a MA for cost-based communication problem. Recombination and mutation of population is proposed for global search and Ahuja and Murty (AM) tree improvement heuristic (AM-H) and the Random Sampling Ahuja and Murty (RSAM) algorithm are used for local search heuristics [44].

Merz and Freisleben proposed Genetic Local Search (GLS) algorithm for quadratic assignment problem (QAP). Proposed algorithm uses GA for global search and swapping items for local search [36].

Wolf and Merz described a new algorithm for supply chain problems. Linear programming (LP) is used as local search heuristic and mutation based filter checks offspring generation is applied for global search technique [43] .

CHAPTER 3

PROPOSED ALGORITHMS

2DBPP is an NP combinatorial problem. In daily life, humans encounter this problem directly or indirectly. Minimizing the textile usage for a manufacturer and planning the position of advertisements in a newspaper are well known problems. Some packing problems must be considered to solve with other objectives such as load balancing, due dates, etc.

In this thesis, we tried to solve two different packing problems. For each problem, we developed two different algorithms.

Our first problem is minimizing the number of bins. We developed MHO-SOMA for oriented 2D offline BPP and MHNO-SOMA for non-oriented 2D offline BPP.

Minimizing number of bins and increasing load balance is our second problem. We proposed MHO-MOGA for oriented 2D offline BPP and MHNO-MOGA for non-oriented 2D offline BPP.

In first section, mathematical formulation of problems are described. In second section, we defined orientation in packing problems. In third section, we gave information about heuristics (FNF, FFF, BFDH, UTS, LGFof and LGFi) which are used in proposed algorithms. In fourth and fifth sections, we described details of our proposed algorithms (MHO-SOMA, MHNO-SOMA, MHO-MOGA and MHNO-MOGA).

3.1 Problem Definition

Load balancing for 2DBPP means that trying to balance total moments of rectangles which are on the left of centre of gravity (CG) of bins with total moments of rectangles which are on the right of CG of bins according to CG of bins. In our case, Euclidean Center of bin is considered as CG of bin. CG of bin and CG of rectangle is shown in Figure 3.1:

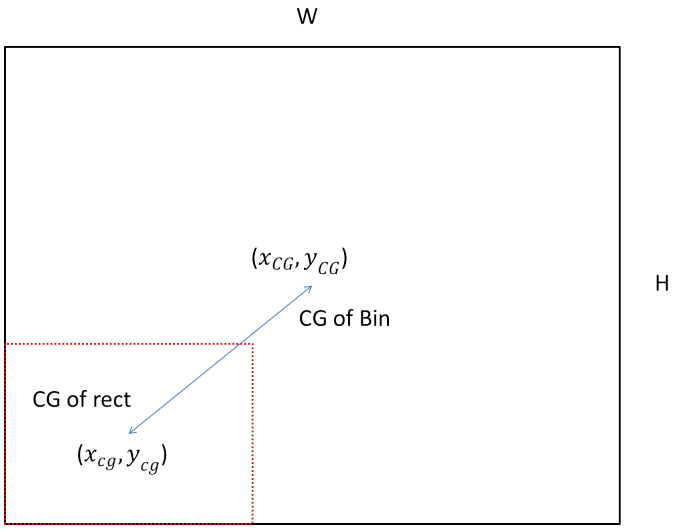

Figure 3.1: CG of bin and CG of rectangle

Two different load balancing of two different bins with same rectangle list is shown in Figure 3.2 and Bin2 has better load balance than Bin1.

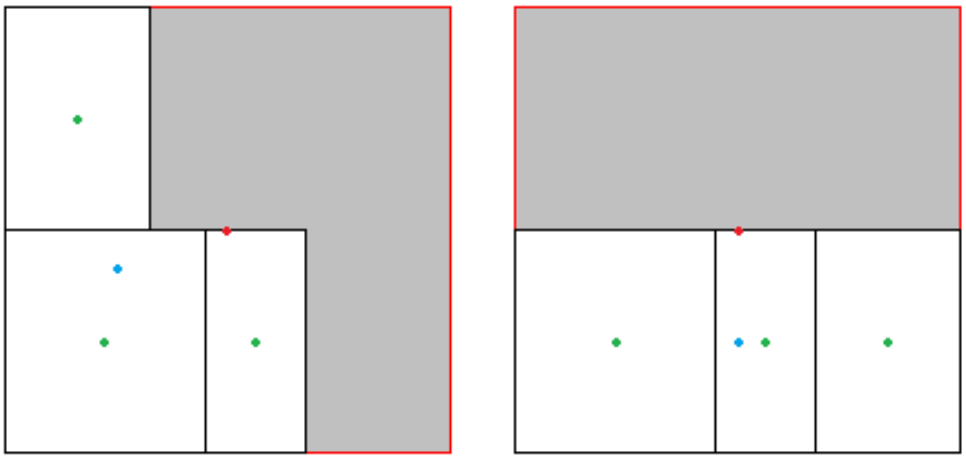

Figure 3.2: Bin1 (left) and Bin2 (right), red dot for CG of bin, green dots for CG of rects and blue dot for CG of load

Definition of concept and mathematical formulation of single objective and multi-objective bin packing problem will be explained in two subsections.

3.1.1 Single Objective 2DBPP

Single objective two dimensional bin packing [39, 3] tries to

minimize

$$C = \sum_{j=1}^{n} c_j \tag{3.1}$$

subject to

$$x_i + (w_i w_i^x) + (h_i h_i^x) \leq x_k + (1 - le_{ik}) \quad , \forall i, k, i < k \tag{3.2}$$

$$x_k + (w_k w_k^x) + (h_k h_k^x) \leq x_i + (1 - ri_{ik}) \quad , \forall i, k, i < k \tag{3.3}$$

$$y_i + (w_i w_i^x) + (h_i h_i^x) \leq y_k + (1 - un_{ik}) \quad , \forall i, k, i < k \tag{3.4}$$

$$y_k + (w_k w_k^x) + (h_k h_k^x) \leq y_i + (1 - ab_{ik}) \quad , \forall i, k, i < k \tag{3.5}$$

$$le_{ik} + ri_{ik} + un_{ik} + ab_{ik} \leq p_{ij} + p_{kj} - 1 \quad , \forall i, k, i < k \tag{3.6}$$

$$\sum_{j=1}^{C} p_{ij} = 1 \quad , \forall i \tag{3.7}$$

$$\sum_{i=1}^{B} p_{ij} \leq M c_j \quad , \forall j \tag{3.8}$$

$$x_i + (w_i w_i^x) + (h_i h_i^x) \leq W_j + (1 - p_{ij})M \quad , \forall i, j \tag{3.9}$$

$$y_i + (w_i w_i^y) + (h_i h_i^y) \leq H_j + (1 - p_{ij})M \quad , \forall i, j \tag{3.10}$$

$$w_i^x, w_i^y, h_i^x, h_i^y, le_{ik}, ri_{ik}, ab_{ik}, un_{ik}, p_{ij}, c_j \in 0, 1 \quad , \forall i, k, i < k \tag{3.11}$$

$$x_i, y_i \geq 0 \quad , \forall i \tag{3.12}$$

Variables of formulation are described as

C	total number of bin
w_i, h_i	indicates width and height of rectangle i
W_j, H_j	indicates width and height of bin j
x_i, y_i	indicates left-bottom corner of rectangle i as coordinate
w_i^x, w_i^y	binary variables indicate width of rectangle i is parallel to X and Y axis
h_i^x, h_i^y	binary variables indicate height of rectangle i is parallel to X and Y axis
le_{ik}	a binary variable indicates rectangle i is placed on the left side of rectangle k
ri_{ik}	a binary variable indicates rectangle i is placed on the right side of rectangle k
ab_{ik}	a binary variable indicates rectangle i is placed above rectangle k
un_{ik}	a binary variable indicates rectangle i is placed under rectangle k
p_{ij}	a binary variable indicates; $p_{ij} = 1$ if rectangle i is placed in bin j otherwise $p_{ij} = 0$
c_j	a binary variable indicate; $c_j = 1$ if bin j is used otherwise $c_j = 0$
M	an arbitrarily large number used in Bin-M constraints

According to above definitions

- constraints (Eq. 3.2-3.5) ensure that none of the rectangles overlaps each other
- constraint (Eq. 3.6) makes sure non-overlapping rectangles
- constraint (Eq. 3.7) guarantees that each rectangle is packed only in single bin

- constraint (Eq. 3.8) indicates that a bin is used iff a rectangle is packed into it
- constraints (Eq. 3.9-3.10) guarantee that all rectangles in the bin do not exceed the dimensions of the bin
- constraints (Eq. 3.11-3.12) describes the range of variables

3.1.2 Multi-Objective 2DBPP

Multi-objective two dimensional bin packing with load balancing [39, 3, 4] tries to minimize

$$(C/2 + LB/2) \tag{3.13}$$

when

$$C = \sum_{j=1}^{n} c_j \tag{3.14}$$

$$LB = \sum_{j=1}^{C} \left| \sum_{i=1}^{B} p_{ij} d_i m_{ij} \sqrt{(x_{ij} + (w_{ij}/2) - x_{CG})^2 + (y_{ij} + (h_{ij}/2) - y_{CG})^2} \right| \tag{3.15}$$

subject to

$$x_i + (w_i w_i^x) + (h_i h_i^x) \leq x_k + (1 - le_{ik}) \quad , \forall i, k, i < k \tag{3.16}$$

$$x_k + (w_k w_k^x) + (h_k h_k^x) \leq x_i + (1 - ri_{ik}) \quad , \forall i, k, i < k \tag{3.17}$$

$$y_i + (w_i w_i^x) + (h_i h_i^x) \leq y_k + (1 - un_{ik}) \quad , \forall i, k, i < k \tag{3.18}$$

$$y_k + (w_k w_k^x) + (h_k h_k^x) \leq y_i + (1 - ab_{ik}) \quad , \forall i, k, i < k \tag{3.19}$$

$$le_{ik} + ri_{ik} + un_{ik} + ab_{ik} \leq p_{ij} + p_{kj} - 1 \quad , \forall i, k, i < k \tag{3.20}$$

$$\sum_{j=1}^{C} p_{ij} = 1 \quad , \forall i \tag{3.21}$$

$$\sum_{i=1}^{B} p_{ij} \leq Mc_j \quad , \forall j \tag{3.22}$$

$$x_i + (w_i w_i^x) + (h_i h_i^x) \leq W_j + (1 - p_{ij})M \quad , \forall i, j \tag{3.23}$$

$$y_i + (w_i w_i^y) + (h_i h_i^y) \leq H_j + (1 - p_{ij})M \quad , \forall i, j \tag{3.24}$$

$$w_i^x, w_i^y, h_i^x, h_i^y, le_{ik}, ri_{ik}, ab_{ik}, un_{ik}, p_{ij}, c_j \in 0, 1 \quad , \forall i, k, i < k \tag{3.25}$$

$$x_i, y_i \geq 0 \quad , \forall i \tag{3.26}$$

$$m_{ij} \in -1, 1 \quad , \forall i \tag{3.27}$$

$$x_{CG} \geq (W/2) \tag{3.28}$$

$$y_{CG} \geq (H/2) \tag{3.29}$$

Variables of formulation are described as

B	total number of rectangle
C	total number of bin
LB	total sum of load balancing
w_i, h_i	indicates width and height of rectangle i
d_i	indicates weight of rectangle i
W_j, H_j	indicates width and height of bin j
x_i, y_i	indicates left-bottom corner of rectangle i as coordinate
x_{CG}	indicates x coordinate of center of gravity of bin which is equal to (W/2)
x_{CG}	indicates y coordinate of center of gravity of bin which is equal to (H/2)
w_i^x, w_i^y	binary variables indicate width of rectangle i is parallel to X and Y axis
h_i^x, h_i^y	binary variables indicate height of rectangle i is parallel to X and Y axis
le_{ik}	a binary variable indicates rectangle i is placed on the left side of rectangle k
ri_{ik}	a binary variable indicates rectangle i is placed on the right side of rectangle k
ab_{ik}	a binary variable indicates rectangle i is placed above rectangle k
un_{ik}	a binary variable indicates rectangle i is placed under rectangle k
p_{ij}	a binary variable indicates; $p_{ij} = 1$ if rectangle i is placed in bin j otherwise $p_{ij} = 0$
m_{ij}	a variable indicates; $m_{ij} = 1$ if $(x_{ij} + (w_{ij}/2) - x_{CG}) \geq 0$ otherwise $m_{ij} = -1$
c_j	a binary variable indicates; $c_j = 1$ if bin j is used otherwise $c_j = 0$
M	an arbitrarily large number used in Bin-M constraints

According to above definitions

- constraints (Eq. 3.16-3.19) ensure that none of the rectangles overlaps each other
- constraint (Eq. 3.20) makes sure non-overlapping rectangles
- constraint (Eq. 3.21) guarantees that each rectangle is packed only in single bin
- constraint (Eq. 3.22) indicates that a bin is used iff a rectangle is packed into it
- constraints (Eq. 3.23-3.24) guarantee that all rectangles in the bin do not exceed the dimensions of the bin

- constraints (Eq. 3.25-3.29) describes the range of variables

3.2 Orientation

Orientation means that an object can be rotated (90 angle) or not, while the object is being packed into bin. If an object of problem can be rotated while it is being packed into bin, then it is called as non-oriented or orientation-free. If an object of problem cannot be rotated, then it is called as oriented or orientation-fix. Therefore, if objects of problem can be rotated, then this problem is a non-oriented bin packing problem. Otherwise, the problem is an oriented bin packing problem. Textile industry can change the orientation of single color shirts by rotating the shirts while the process is in cutting phase, because there is no difference between rotation or not so it is a non-oriented bin packing problem. But shipping industry must consider the orientation of loads which are fragile, so it is an oriented packing bin problem.

3.3 State-of-the-art Algorithms

Insertion of items into bins is achieved by the use of heuristics. Each technique can result in a different arrangement of items so heuristic is one of the key part of BPP. Large numbers of methods have been proposed for bin packing problem. We choose six well known heuristics, which are FNF, FFF, BFDH, LGFi, LGFof and UTS, to pack items. In this study, FNF, FFF, BFDH, LGFi and LGFof heuristics are implemented and TSpack implementation of UTS is used.

3.3.1 Finite Next Fit

Berkey and Wang developed FNF algorithm [35]. The proposed FNF is described below in Algorithm 1.

Algorithm 1 Finite Next Fit (FNF) algorithm

```
1:  function FNF(rectList, binWidht, binHeight)
2:      sort rectList by decreasing height
3:      currentBin ← CreateNewBin(binWidth, binHeight)
4:      currentLevel ← CreateNewLevel(currentBin)
5:      n ←rectList.length
6:      for i ← 1, n do
7:          if rect(i) fits into currentLevel then
8:              pack rect(i) into currentLevel
9:          else
10:             currentLevel ← CreateNewLevel(currentBin)
11:             if rect(i) fits into currentLevel then
12:                 pack rect(i) into currentLevel
13:             else
14:                 currentBin ← CreateNewBin(binWidth, binHeight)
15:                 currentLevel ← CreateNewLevel(currentBin)
16:             end if
17:         end if
18:     end for
19: end function
```

An example of proposed algorithm is shown in Table 3.1 and Figure 3.3:

Table 3.1: Input List with Bin Width = 20 and Bin Height = 20

Rect	1	2	3	4	5	6	7	8	9	10
Height	9	3	5	3	15	10	15	8	4	2
Width	10	8	6	10	2	6	4	3	6	12

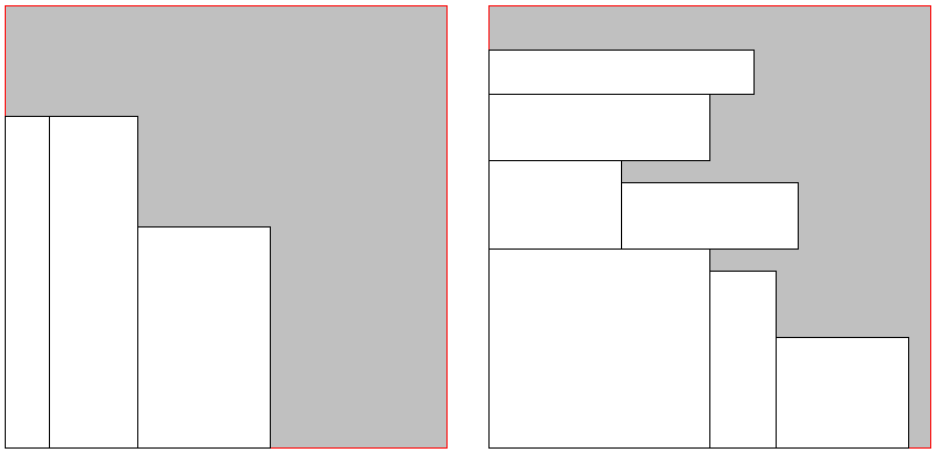

Figure 3.3: Finite Next Fit Output

3.3.2 Finite First Fit

Berkey and Wang introduced FFF algorithm [35]. The proposed FFF is described below in Algorithm 2.

Algorithm 2 Finite First Fit(FFF) algorithm

```
1:  function FFF(rectList, binWidht, binHeight)
2:      sort rectList by decreasing height
3:      n ←rectList.length
4:      for i ← 1, n do
5:          m ←bins.length
6:          for j ← 1, m do
7:              n ←bin(j).levels.length
8:              for k ← 1, n do
9:                  if rect(i) fits into bin(j).level(k) then
10:                     pack rect(i) into bin(j).level(k)
11:                 end if
12:             end for
13:         end for
14:         if rect(i) does not fit into any bin then
15:             create new bin
16:             pack rect(i) into new bin
17:         end if
18:     end for
19: end function
```

An example of proposed algorithm is shown in Table 3.1 and Figure 3.4:

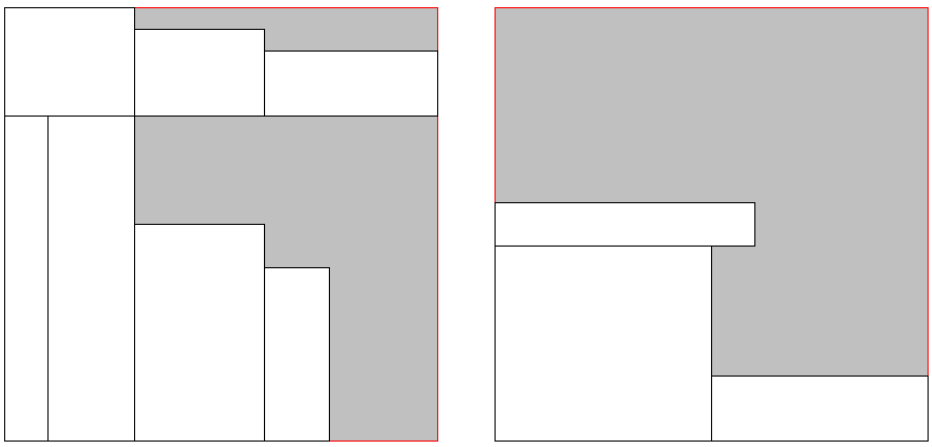

Figure 3.4: Finite First Fit Output

3.3.3 Best Fit Decreasing Height

Coffman and Shor described BFDH algorithm [23]. The proposed BFDH is described below in Algorithm 3.

Algorithm 3 Best Fit Decreasing Height(BFDH) algorithm

```
1:  function BFDH(rectList, binWidht, binHeight)
2:      sort rectList by decreasing height
3:      n ←rectList.length
4:      for i ← 1, n do
5:          minRemainingSpace ← binWidth
6:          m ←bins.length
7:          for j ← 1, m do
8:              n ←bin(j).levels.length
9:              for k ← 1, n do
10:                 if rect(i) fits into bin(j).level(k) then
11:                     pack rect(i) into bin(j).level(k)
12:                     if remainingSpace < minRemaningSpace then
13:                         minRemainingSpace ← remainingSpace
14:                     end if
15:                 end if
16:             end for
17:         end for
18:         if rect(i) fits fit into any bin then
19:             packs the item into minRemainingSpace bin level
20:         else
21:             create new bin
22:             pack rect(i) into new bin
23:         end if
24:     end for
25: end function
```

An example of proposed algorithm is shown in Table 3.1 and Figure 3.5:

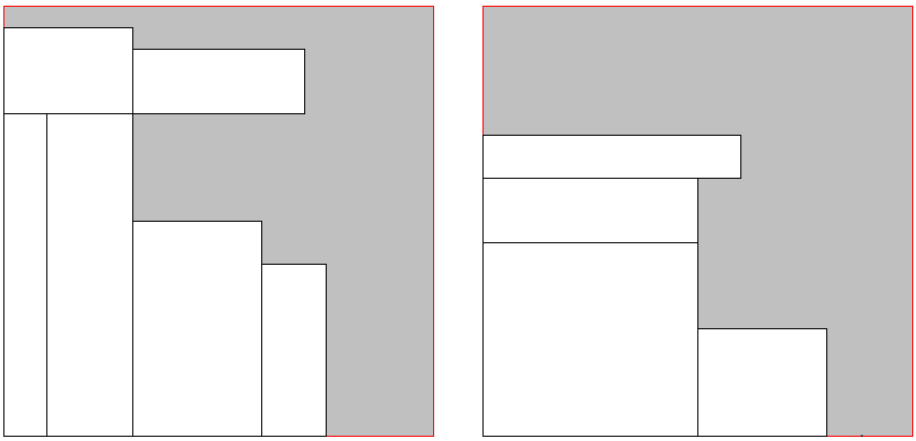

Figure 3.5: Best Fit Decreasing Height Output

3.3.4 Unified Tabu Search

Lodi et al explained the detail of their UTS and implemented the algorithm [12]. UTS algorithm of Lodi is explained below in Algorithm 4.

Algorithm 4 Unified Tabu Search(UTS) algorithm

```
1:  function TSpack(rectList, binWidht, binHeight)
2:      z* ← A(f1; : : : ; ng)
3:      L ← alowerboundontheoptimalsolutionvalue
4:      if z* = L then
5:          stop
6:      end if
7:      initialize all tabu lists to empty
8:      pack each item into a separate bin
9:      z ← n                                   ▷ Tabu Search solution value
10:     d ← 1;
11:     determine the target bin t
12:     while dotimelimitisnotreached
13:         diversity ← false
14:         k ← 1
15:         while dodiversify = falseandz* > L
16:             kin ← l SEARCH(t, k, diversify, z)
17:             z ← min(z, z*)
18:             if then k <= kin
19:                 determine the new target bin t
20:             end if
21:         end while
22:         if z* = L then
23:             stop
24:         elseDIVERSIFICATION(d, z, t)
25:         end if
26:     end while
27: end function
```

An example of proposed algorithm is shown in Table 3.1 and Figure 3.6:

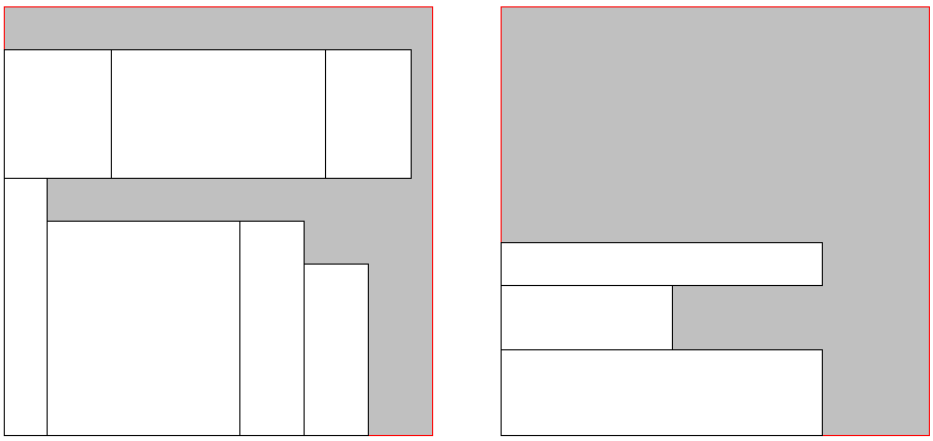

Figure 3.6: Unified Tabu Search Output

3.3.5 Improved Left Gap Fill

Wong and Lee improved LGF into LGFi [30]. LGFi is used for non-oriented bin packing problem. The proposed LGFi is described below in Algorithm 5.

Algorithm 5 Improved Left Gap Fill Non-Oriented(LGFi) algorithm

```
1:  function LGFi(rectList, binWidht, binHeight)
2:      rotate rects to make width greater than height
3:      sort rectList by decreasing height
4:      currentBin ← CreateNewBin(binWidth, binHeight)
5:      currentPoint(x, y) ← 0, 0
6:      while rectList.length! = 0 do
7:          rect ← GetNextRectangle(rectList)
8:          gapWidth ← binWidth − currentPointX
9:          gapHeight ← binHeight − currentPointY
10:         gap ← min(gapWidth, gapHeight)
11:         if gap = 0 then
12:             currentBin ← CreateNewBin(binWidth, binHeight)
13:             currentPoint(x, y) ← 0, 0
14:         else if gap < min(allRemainingRectHeight) then
15:             mark are as waste space
16:             set currentPoint(x, y) to lowest-leftmost point
17:         else if gap = rect.heightorgap = rect.width then
18:             pack rect into currentPoint(x, y)
19:             set currentPoint(x, y) to lowest-leftmost point
20:             remove rect from rectList
21:         else if norectfitintogap then
22:             rect ← GetNextRectangle(rectList)
23:             pack rect into currentPoint(x, y)
24:             set currentPoint(x, y) to lowest-leftmost point
25:             remove rect from rectList
26:         end if
27:     end while
28: end function
```

An example of proposed algorithm is shown in Table 3.1 and Figure 3.7:

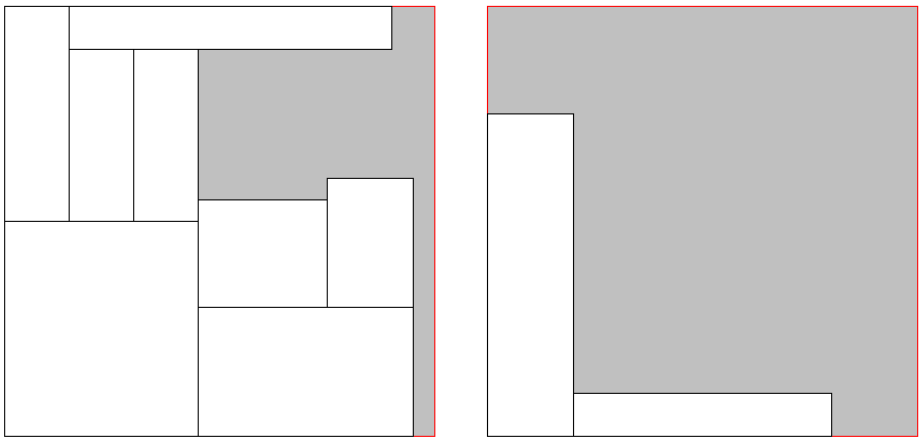

Figure 3.7: LGFi Output

3.3.6 Oriented Improved Left Gap Fill

Wong and Lee improved LGFi for O-BPP and developed LGFof [30]. The proposed LGFof is described below in Algorithm 6.

Algorithm 6 Improved Left Gap Fill Oriented(LGFof) algorithm

```
1:  function LGFof(rectList, binWidht, binHeight)
2:      sort rectList by non-increasing area
3:      currentBin ← CreateNewBin(binWidth, binHeight)
4:      currentPoint(x, y) ← 0, 0
5:      while rectList.length! = 0 do
6:          rect ← GetNextRectangle(rectList)
7:          gapWidth ← binWidth − currentPointX
8:          gapHeight ← binHeight − currentPointY
9:          gap ← min(gapWidth, gapHeight)
10:         if gap = 0 then
11:             currentBin ← CreateNewBin(binWidth, binHeight)
12:             currentPoint(x, y) ← 0, 0
13:         else if gap < min(allRemainingRectHeight) then
14:             mark are as waste space
15:             set currentPoint(x, y) to lowest-leftmost point
16:         else if gap = rect.heightorgap = rect.width then
17:             pack rect into currentPoint(x, y)
18:             set currentPoint(x, y) to lowest-leftmost point
19:             remove rect from rectList
20:         else if norectfitintogap then
21:             rect ← GetNextRectangle(rectList)
22:             pack rect into currentPoint(x, y)
23:             set currentPoint(x, y) to lowest-leftmost point
24:             remove rect from rectList
25:         end if
26:     end while
27: end function
```

An example of proposed algorithm is shown in Table 3.1 and Figure 3.8:

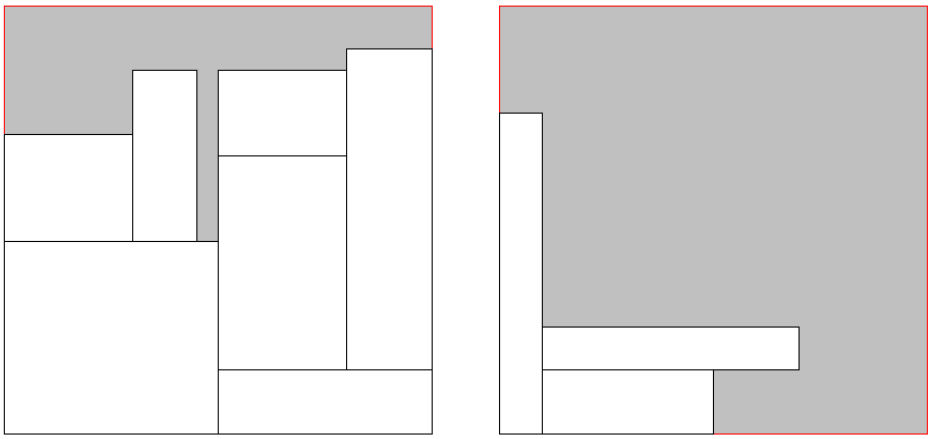

Figure 3.8: LGFof Output

3.3.7 Genetic Algorithm

GA mimics the behavior of the natural evolution process. GA has got the power of natural evolution process in order to find optimum solution for a large search space. In GAs, a chromosome means a possible solution for the given problem set. A list of chromosome generates the population of GA. Each chromosome consists of genes and has an, encoding schema. Heuristic can be part of a chromosome or genetic algorithm. Each item of the problem means a gene in the chromosome. Encoding keeps the identification of items. Heuristic is used to find the fitness value of the chromosome.

GA has three operators:

- Selection: survival of the individual
- Crossover: mating the parents in order to get offspring
- Mutation: random modification of individual

GA works as follows in Algorithm 7.

Algorithm 7 Genetic Algorithm (GA) algorithm

1: Initialize population
2: Calculate fitness of population
3: **while** Best individual (chromosome) does not meet the criteria **do**
4: Select parents
5: Crossover operation in order to get offsprings
6: Mutation operation over population
7: Determine fitness of population
8: **end while**
9: Best individual is the solution of problem

3.3.8 Chromosome Structure

Chromosome is a candidate solution of BPP. For 2DBPP, rectangular items are the genes of a chromosome. In this study, chromosome is not only composed of genes also it has two heuristics to pack the genes into bins. We have used permutation encoding to keep the identification of rectangles. Permutation encoding is a form of keeping width-height of rectangular items and sequence between rectangles.

Chromosome Structure of 2DBPP Explanation of structure is in Figure 3.9:

- W: width of rectangle
- H: height of rectangle:
- Heu1: heuristic one
- Heu2: heuristic two

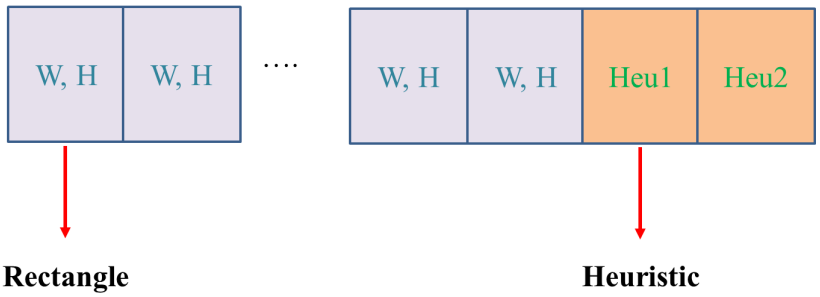

Figure 3.9: Chromosome Structure

An example of chromosome is shown Figure 3.10:

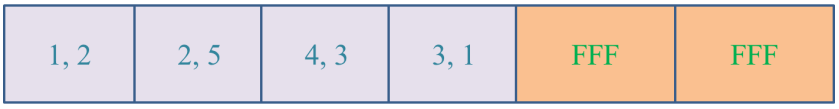

Figure 3.10: Example Chromosome

3.3.9 Selection

We have used Elitism as the selection of parents in GA. Elitism means that best chromosomes in the population have the chance to mate in order to create new offsprings.

3.3.10 Crossover

Single point crossover is applied technique in our study. One point is chosen. From beginning of the chromosome to the crossover point is copied from first parent and the rest of the chromosome comes from the second parent. Our crossover point is half of the chromosome. We also get first heuristic from one parent and second heuristic from other parent.

An example of crossover is shown in Figure 3.11 and Figure 3.12:

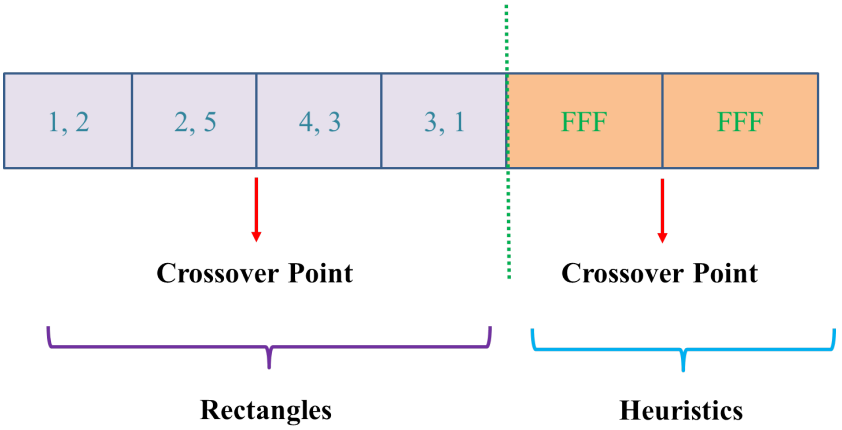

Figure 3.11: Single Point Crossover Chromosome Structure

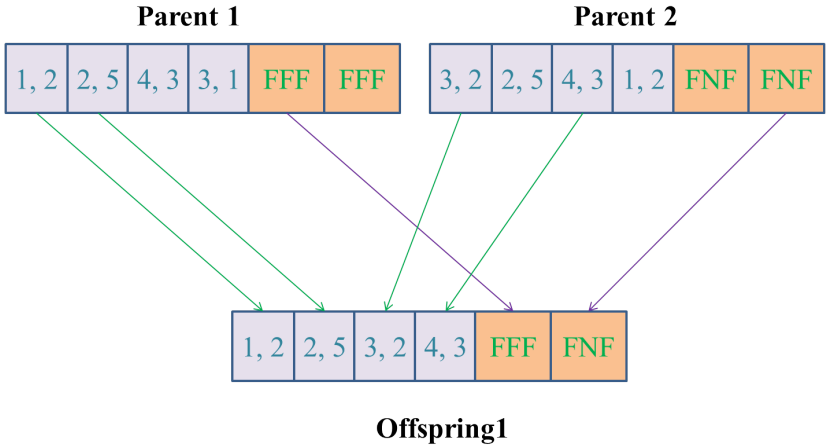

Figure 3.12: Single Point Crossover Example

3.3.11 Mutation

Mutation ensures the diversity of population. Invalid genes can be generated as a result of mutation operation. Mutation is only done in the concept of rectangular items and heuristics are not mutated. In this study, we do not want to create invalid genes so three types of mutation operations are implemented:

- Swap : Only swaps selected genes
- Rotate : Only rotate object, the value of width and height is swapped
- Swap and Rotate : Selected genes are swapped and rotated

An example of swap mutation is shown in Figure 3.13:

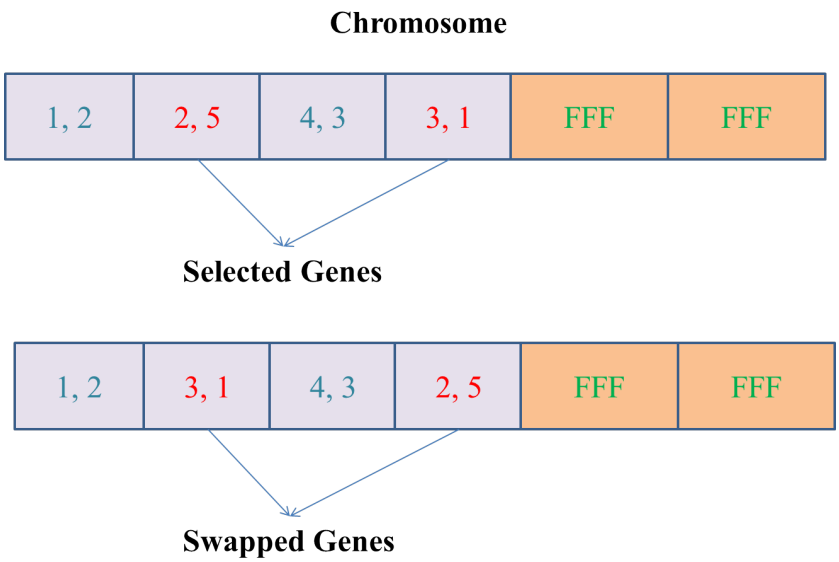

Figure 3.13: Swap Mutation Example

An example of rotation mutation is shown in Figure 3.14:

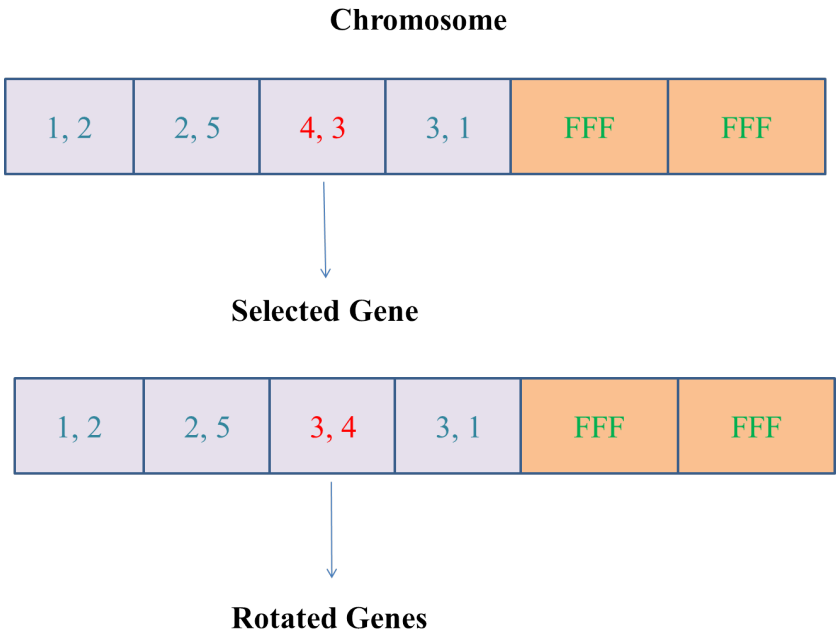

Figure 3.14: Rotation Mutation Example

An example of swap-rotation mutation is shown in Figure 3.15:

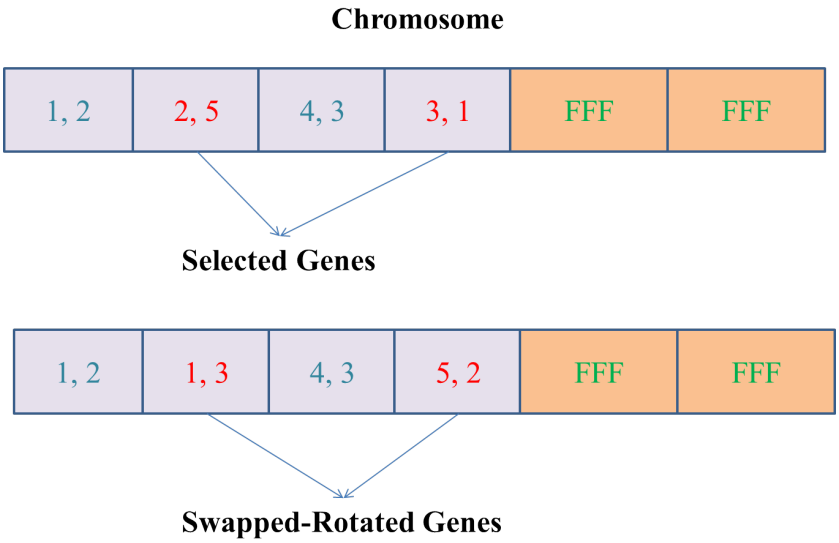

Figure 3.15: Swap-Rotation Mutation Example

3.3.12 Memetic Algorithm

MA is another growing are of EA. It also mimic natural evolution process but it may differ from GA by performing individual learning which is also known as meme(s). In this study, MA uses the same chromosome structure, selection, crossover and mutation operation of GA.

MA works as follows in Algorithm 8.

Algorithm 8 Memetic Algorithm (MA) algorithm

```
Initialize population
Calculate fitness of population
while Best individual (chromosome) does not meet the criteria do
    Select parents
    Crossover operation in order to get offsprings
    Mutation operation over population
    Individual learning is proceed (local search)
    Determine fitness of population
end while
Best individual is the solution of problem
```

3.4 Single Objective Proposed Algorithms

Bin packing problems mostly consider the number of bins so the goal of the algorithms is to pack items in minimum number of bins. We have developed a two Multi-Heuristic Single Objective Memetic Algorithms for minimizing the number of bins by considering orientation is important or not. Our fitness functions are the result of applied heuristic in the term of number of bins. We explain two algorithms in two different subsections.

3.4.1 MHO-SOMA for O-2DBPP

Minimizing the number of bins is our goal but orientation of items must be conserved for some 2D offline BPP. Advertisement packing for a newspaper can be classified as one example of these problems. To keep orientation of items, we developed Multi-Heuristic Oriented Single Objective Memetic Algorithm (MHO-SOMA). FNF, FFF, BFDH and LGFof are used as heuristics to pack items. First, genetic algorithm with heuristics is run by the use of swap mutation operation. Then, best individual of genetic algorithm is selected and UTS is applied to best individual as local search. Overall best result is the solution of oriented 2D offline BPP. MHO-SOMA is explained in Algorithm 9 and Algorithm 10:

Algorithm 9 Multi-Heuristic Oriented Single Objective Memetic Algorithm Functions

```
1: function INITALIZE(rectList, n, k)
2:     for i ← 1, n do
3:         copyList ← rectList
4:         for j ← 1, k do
5:             APPLYSWAPMUTATION(copyList)
6:         end for
7:         heuristic ← CHOOSEONE(FNF, FFF, BFDH, LGFof)
8:         heuristic1 ← heuristic
9:         heuristic2 ← heuristic
10:        CREATECHROMOSOME(copyList, heuristic1, heuristic2)
11:        PUTPOPULATION(chromosome)
12:    end for
13:    return population
14: end function
15: function FITNESS(population)
16:    for eachchromosomeinPopulation do
17:        if heuristic1 = Heuristic2 then
18:            fitness ← INSERT(heuristic1, rectList)
19:        else
20:            resHeuristic1 ← INSERT(heuristic1, firstHalfRectList)
21:            resHeuristic2 ← INSERT(heuristic2, secpmdHalfRectList)
22:            fitness ← resHeuristic1 + reHeuristic2
23:        end if
24:    end for
25:    SortChromosomesaccordingtotheirfitnessvalue
26:    return population
27: end function
```

Algorithm 10 Multi-Heuristic Oriented Single Objective Memetic Algorithm (MHO-SOMA) algorithm

```
1:  function MHO-SOMA(rectList, binW, binH, numGen, numPop, numIMut)
2:      population ← INITIALIZE(rectList, numPop, numIMut)
3:      FITNESS(population)
4:      for k ← 1, numGen do
5:          parents ← ELITISMSELECTPARENTS(population)
6:          offsprings ← SINGLEPOINTCROSSOVER(parents)
7:          SWAPMUTATION(population)
8:          FITNESS(population)
9:      end for
10:     rectSequence ← bestindividualrectanglesequence
11:     fitnessBest ← bestindividualfitness
12:     fitnessUTS ← INSERT(UTS, rectSequence)
13:     if fitnessUTS < fitnessBest then
14:         resultHeuristic ← UTS
15:         resultNumberOfBin ← fitnessUTS
16:     else
17:         resultHeuristic ← bestindividualheuristic
18:         resultNumberOfBin ← fitnessBest
19:     end if
20:     return resultHeuristic, resultNumberOfBin
21: end function
```

3.4.2 MHNO-SOMA for NO-2DBPP

Orientation of items is not considered for some 2D offline BPP. Textile usage of manufacturer can be classified as one example of these problems. We proposed Multi-Heuristic Non-Oriented Single Objective Memetic Algorithm (MHNO-SOMA) for these problems. Rotation and swap-rotation mutations are used to change orientation of items. First we run genetic algorithm with heuristics: FNF, FFF, BFDH and LGFof by the use of rotation and swap-rotation mutations. Then, best chromosome

of genetic algorithm is selected to apply UTS and LGFi. Overall best result is the solution of non-oriented 2D offline BPP. MHNO-SOMA is explained in Algorithm 11 and Algorithm 12:

Algorithm 11 Multi-Heuristic Non-Oriented Single Objective Memetic Algorithm Functions

```
1:  function INITALIZE(rectList, n, k)
2:      for i ← 1, n do
3:          copyList ← rectList
4:          for j ← 1, k do
5:              RotationMutation(copyList) or SwapRotationMutation(copyList)
6:          end for
7:          heuristic ← ChooseOne(FNF, FFF, BFDH, LGFof)
8:          heuristic1 ← heuristic
9:          heuristic2 ← heuristic
10:         CreateChromosome(copyList, heuristic1, heuristic2)
11:         PutPopulation(chromosome)
12:     end for
13:     return population
14: end function
15: function FITNESS(population)
16:     for eachchromosomeinPopulation do
17:         if heuristic1 = Heuristic2 then
18:             fitness ← INSERT(heuristic1, rectList)
19:         else
20:             resHeuristic1 ← INSERT(heuristic1, firstHalfRectList)
21:             resHeuristic2 ← INSERT(heuristic2, secpmdHalfRectList)
22:             fitness ← resHeuristic1 + reHeuristic2
23:         end if
24:     end for
25:     SortChromosomesaccordingtotheirfitnessvalue
26:     return population
27: end function
```

Algorithm 12 Multi-Heuristic Non-Oriented Single Objective Memetic Algorithm (MHNO-SOMA) algorithm

```
1: function MHNO-SOMA(rectList, binW, binH, numGen, numPop, numIMut)
2:     population ← INITIALIZE(rectList, numPop, numIMut)
3:     FITNESS(population)
4:     for k ← 1, numGen do
5:         parents ← ELITISMSELECTPARENTS(population)
6:         offsprings ← SINGLEPOINTCROSSOVER(parents)
7:         ROTATEMUTATION(population) or SWAPROTATEMUTATION(population)
8:         FITNESS(population)
9:     end for
10:    rectSequence ← bestindividualrectanglesequence
11:    fitnessBest ← bestindividualfitness
12:    fitnessUTS ← INSERT(UTS, rectSequence)
13:    fitnessLGFi ← INSERT(LGFi, rectSequence)
14:    resultNumberOfBin ← MIN(fitnessBest, fitnessUTS, fitnessLGFi)
15:    resultHeuristic ← resultNumberOfBinHeuristic
16:    return resultHeuristic, resultNumberOfBin
17: end function
```

3.5 Multi-Objective Proposed Algorithms

Load balancing of bins is an important issue so the goal of the algorithms is not only to pack items in minimum number of bins but also maximize the load balancing of bins. We have developed two Multi-Heuristic Multi-Objective Genetic Algorithms for minimizing number of bins and maximizing load balance.

Our fitness function is composed of two parts, fitness of number of bins and fitness of center of gravity. Number of bins is calculated directly by heuristic. Our algorithm is not directly consider load balancing so we add calculation of center of gravity to each bin. In order to calculate center of gravity, each bin's center point is selected as Center of Gravity (CG). When a rectangle box inserted, then the Euclidean distance of its center to bin's CG is calculated and multiplied by the weight of rectangle. This

calculated value is CG of rectangle. If sign of x coordinate of rectangle is minus then CG of rectangle is subtracted from total CG of Bin. If sign of x coordinate of rectangle is plus then CG of rectangle is added to total CG of Bin. When all rectangles are inserted to bins, then absolute values of total CG of bins are added. This calculated value is called as CG of Chromosome. Load balance of bin is explained in Equation 3.30 and load balance of chromosome is explained in Equation 3.31.

$$LB_{Bin} = \sum_{j=1}^{\#Rect} d_{ij} m_{ij} \sqrt{(x_{ij} + (w_{ij}/2) - x_{CG})^2 + (y_{ij} + (h_{ij}/2) - y_{CG})^2} \tag{3.30}$$

$$LB_{Chromosome} = \sum_{i=1}^{\#Bin} \left| \sum_{j=1}^{\#Rect} d_j m_{ij} \sqrt{(x_{ij} + (w_{ij}/2) - x_{CG})^2 + (y_{ij} + (y_{ij}/2) - y_{CG})^2} \right| \tag{3.31}$$

Multi-objective 2D offline BPP can be classified into two categories according to rotation. We developed two algorithms: one for oriented and another for non-oriented problems. Two proposed multi-objective algorithms are explained in two different subsections.

3.5.1 MHO-MOGA for O-2DBPP with LB

Multi-Heuristic Oriented Multi-Objective Genetic Algorithm (MHO-MOGA) is proposed to solve oriented multi-objective 2D offline BPP. FNF, FFF, BFDH, LGFof and UTS are applied as heuristics on the base of genetic algorithm. Swap mutation operation is used to keep orientation of items. Best result of GA is the result of problem. MHO-MOGA is explained in Algorithm 13, Algorithm 14 and Algorithm 15:

Algorithm 13 Multi-Heuristic Oriented Multi-Objective Genetic Algorithm Functions

```
1:  function INITALIZE(rectList, n, k)
2:      for i ← 1, n do
3:          copyList ← rectList
4:          for j ← 1, k do
5:              SWAPMUTATION(copyList)
6:          end for
7:          heuristic ← CHOOSEONE(FNF, FFF, BFDH, LGFof, UTS)
8:          heuristic1 ← heuristic
9:          heuristic2 ← heuristic
10:         CREATECHROMOSOME(copyList, heuristic1, heuristic2)
11:         PUTPOPULATION(chromosome)
12:     end for
13:     return population
14: end function
```

Algorithm 14 Multi-Heuristic Oriented Multi-Objective Genetic Algorithm Functions Part2

```
1: function FITNESS(population)
2:     maxBin ← 0
3:     minBin ← MAXINTEGER
4:     maxCG ← 0
5:     minCG ← MAXINTEGER
6:     for eachchromosomeinPopulation do
7:         if heuristic1 = Heuristic2 then
8:             chromosomeBin ← INSERT(heuristic1, rectList)
9:         else
10:            resHeuristic1 ← INSERT(heuristic1, firstHalfRectList)
11:            resHeuristic2 ← INSERT(heuristic2, secpmdHalfRectList)
12:            chromosomeBin ← resHeuristic1 + reHeuristic2
13:        end if
14:        if chromosomeBin < minBin then
15:            minBin ← chromosomeBin
16:        end if
17:        if chromosomeBin > maxBin then
18:            maxBin ← chromosomeBin
19:        end if
20:        chromosomeCG ← SUM(bin of chromosome)
21:    end for
22:    diffBin ← maxBin − minBin
23:    diffCG ← maxCG − minCG
24:    for eachchromosomeinPopulation do
25:        fitnessBin ← (chromosomeBin − minBin)/DiffBin
26:        FitnessCG ← (chromosomeCG − minCG)/DiffCG
27:        fitness ← (FitnessBin ∗ 0.5) + (FitnessCG ∗ 0.5)
28:    end for
29:    SortChromosomesaccordingtotheirfitnessvalue
30:    return population
31: end function
```

Algorithm 15 Multi-Heuristic Oriented Multi-Objective Genetic Algorithm Oriented (MHO-MOGA) algorithm

```
1: function MHO-MOGA(rectList, binW, binH, numGen, numPop, numIMut)
2:     population ← INITIALIZE(rectList, numPop, numIMut)
3:     FITNESS(population)
4:     for k ← 1, numGen do
5:         parents ← ELITISMSELECTPARENTS(population)
6:         offsprings ← SINGLEPOINTCROSSOVER(parents)
7:         SWAPMUTATION(population)
8:         FITNESS(population)
9:     end for
10:    resultHeuristic ← bestindividualheuristic
11:    resultNumberOfBin ← bestindividualfitness
12:    return resultHeuristic, resultNumberOfBin
13: end function
```

3.5.2 MHNO-MOGA for NO-2DBPP with LB

Sometimes, orientation is not part of problem so in order to solve such problems, we developed Multi-Heuristic Non-Oriented Multi-Objective Genetic Algorithm (MHNO-MOGA). Each individual uses one or two of the heuristics: FNF, FFF, BFDH, UTS, LGFof and LFGi to pack rectangles into bins. At the beginning of genetic algorithm, each indivudual picks only one of the heuristics. In the next phases of GA, an individual can have two different heuristics. Each heuristic is applied to corresponding part of rectangle list. Rotation mutation and swap-rotation mutation are applied to change orientation of rectangles. Best result of GA gives the solution of problem. MHNO-MOGA is explained in Algorithm 16, Algorithm 17 and Algorithm 18:

Algorithm 16 Multi-Heuristic Non-Oriented Multi-Objective Genetic Algorithm Functions

```
1: function INITALIZE(rectList, n, k)
2:     for i ← 1, n do
3:         copyList ← rectList
4:         for j ← 1, k do
5:             ROTATIONMUTATION(SwapMutation) or SWAPROTATIONMUTATION(SwapMutation)
6:         end for
7:         heuristic ← CHOOSEONE(FNF, FFF, BFDH, LGFof, UTS, LGFi)
8:         heuristic1 ← heuristic
9:         heuristic2 ← heuristic
10:        CREATECHROMOSOME(copyList, heuristic1, heuristic2)
11:        PUTPOPULATION(chromosome)
12:    end for
13:    return population
14: end function
```

Algorithm 17 Multi-Heuristic Non-Oriented Multi-Objective Genetic Algorithm Functions Part2

```
1: function FITNESS(population)
2:     maxBin ← 0
3:     minBin ← MAXINTEGER
4:     maxCG ← 0
5:     minCG ← MAXINTEGER
6:     for eachchromosomeinPopulation do
7:         if heuristic1 = Heuristic2 then
8:             chromosomeBin ← INSERT(heuristic1, rectList)
9:         else
10:            resHeuristic1 ← INSERT(heuristic1, firstHalfRectList)
11:            resHeuristic2 ← INSERT(heuristic2, secpmdHalfRectList)
12:            chromosomeBin ← resHeuristic1 + reHeuristic2
13:        end if
14:        if chromosomeBin < minBin then
15:            minBin ← chromosomeBin
16:        end if
17:        if chromosomeBin > maxBin then
18:            maxBin ← chromosomeBin
19:        end if
20:        chromosomeCG ← SUM(bin of chromosome)
21:    end for
22:    diffBin ← maxBin − minBin
23:    diffCG ← maxCG − minCG
24:    for eachchromosomeinPopulation do
25:        fitnessBin ← (chromosomeBin − minBin)/DiffBin
26:        FitnessCG ← (chromosomeCG − minCG)/DiffCG
27:        fitness ← (FitnessBin ∗ 0.5) + (FitnessCG ∗ 0.5)
28:    end for
29:    SortChromosomesaccordingtotheirfitnessvalue
30:    return population
31: end function
```

Algorithm 18 Multi-Heuristic Non-Oriented Multi-Objective Genetic Algorithm (MHNO-MOGA) algorithm

```
1:  function MHNO-MOGA(rectList, binW, binH, numGen, numPop, numIMut)
2:      population ← INITIALIZE(rectList, numPop, numIMut)
3:      FITNESS(population)
4:      for k ← 1, numGen do
5:          parents ← ELITISMSELECTPARENTS(population)
6:          offsprings ← SINGLEPOINTCROSSOVER(parents)
7:          ROTATIONMUTATION(population) or SWAPROTATIONMUTATION(population)
8:          FITNESS(population)
9:      end for
10:     resultHeuristic ← bestindividualheuristic
11:     resultNumberOfBin ← bestindividualfitness
12:     return resultHeuristic, resultNumberOfBin
13: end function
```

CHAPTER 4

EXPERIMENTAL RESULTS

In this Chapter, we present the results of proposed algorithms. All the algorithms are implemented in C++ language and tests are executed under Windows7 OS. The computer, which is used for tests, has 2.10 GHz Intel Core 2 Duo CPU with usable 3 GB RAM.

4.1 Problem Sets

In this study, we tried to solve 2D offline BPPs. For test cases we used two well known problem instance sets. First one is Berkey-Wang classes [35]. They generated six classes. Their classses structures are as follows:

- Class 1 : w_j and h_j uniformly random in [1,10], W = H = 10;
- Class 2 : w_j and h_j uniformly random in [1,10], W = H = 30;
- Class 3 : w_j and h_j uniformly random in [1,35], W = H = 40;
- Class 4 : w_j and h_j uniformly random in [1,35], W = H = 100;
- Class 5 : w_j and h_j uniformly random in [1,100], W = H = 100;
- Class 6 : w_j and h_j uniformly random in [1,100], W = H = 300.

Second instance set is composed of four classes which are generated by Martello and Vigo [41]. Their proposed type structures is :

- Type 1 : w_j uniformly random in $[\frac{2}{3}W; W]$, h_j uniformly random in $[1; \frac{1}{2}H]$;

- Type 2 : w_j uniformly random in [1; $\frac{1}{2}$W], h_j uniformly random in [$\frac{2}{3}$H; H];
- Type 3 : w_j uniformly random in [$\frac{1}{2}$W; W], h_j uniformly random in [$\frac{1}{2}$H; H];
- Type 4 : w_j uniformly random in [1; $\frac{1}{2}$W], h_j uniformly random in [1; $\frac{1}{2}$H].

Martello-Vigo class structures is using W = 100 and H = 100 and listed below:

- Class 7 : type 1 with probability 70%, type 2, 3, 4 with probability 10% each;
- Class 8 : type 2 with probability 70%, type 1, 3, 4 with probability 10% each;
- Class 9 : type 3 with probability 70%, type 1, 2, 4 with probability 10% each;
- Class 10 : type 4 with probability 70%, type 1, 2, 3 with probability 10% each.

In this study, we have four different problems as listed below:

- Single Objective O-2DBPP
- Single Objective NO-2DBPP
- Multi-Objective O-2DBPP with LB
- Multi-Objective NO-2DBPP with LB

In order to solve these problems, we need to find the population size and generation number. We listed Berkey-Wang and Martello-Vigo instances and picked a random number between 1 and 500. The number is 245 and its properties are listed below :

- a Berkey-Wang instance
- its class number is 5
- has 100 rectangles
- its bin width and bin height are 100 unit

Our experimental setup is consists of UTS, LFGi and GA whose heuristics are FNF, FFF, BFDH and LGFof. First we apply GA to the problem and then best result's rectangle list is given as input to UTS and LGFi. Each set is run twice. Our experimental result is shown in Table 4.1, Table 4.2 and Table 4.3.

Table 4.1: Result of population-generation for instance number 245 (Part 1)

Population	Generation	GA Algo(s)	GA Res(s)	UTS	LGFi
10	10	LGFof	29	30	29
	20	LGFof / FFF	29/5	30.5	29
	40	LGFof / FFF	29.5	30	29
	60	BFDH / LGFof	29	31.5	29
	80	LGFof	28.5	30	29
	100	LGFof / BFDH	28.5	30	29
	200	LGFof / BFDH	28	30.5	29
	300	LGFof	28	30	29
	400	LGFof	28	29.5	29
	500	LGFof	28	30.5	29
20	10	LGFof	29	30	29
	20	LGFof	29	30	29
	40	LGFof / BFDH	29	30.5	29
	60	LGFof	28.5	30	29
	80	LGFof	28.5	29.5	29
	100	LGFof	28	29.5	29
	200	LGFof	28	30.5	29
	300	LGFof	28	29.5	29
	400	LGFof	28	29.5	29
	500	LGFof	28	29.5	29
40	10	LGFof	29	30.5	29
	20	LGFof	29.5	30	29
	40	LGFof	28	30	29
	60	LGFof	28.5	30	29
	80	LGFof	28.5	30	29
	100	LGFof	28	30	29
	200	LGFof	28	30	29
	300	LGFof	28	30	29
	400	LGFof	28	29.5	29
	500	LGFof	28	30	29
60	10	LGFof	29	30	29
	20	LGFof	28.5	29.5	29
	40	LGFof	28	30	29
	60	LGFof	28	30	29
	80	LGFof	28	30	29
	100	LGFof	28	30	29
	200	LGFof	28	30.5	29
	300	LGFof	28	30.5	29
	400	LGFof	28	29.5	29
	500	LGFof	28	29.5	29

Table 4.2: Result of population-generation for instance number 245 (Part 2)

Population	Generation	GA Algo(s)	GA Res(s)	UTS	LGFi
80	10	LGFof	29	30	29
	20	LGFof	29	30	29
	40	LGFof	28	29.5	29
	60	LGFof	28	30	29
	80	LGFof	28	29	29
	100	LGFof	28	30	29
	200	LGFof	28	29.5	29
	300	LGFof	28	29.5	29
	400	LGFof	28	29.5	29
	500	LGFof	28	29	29
100	10	LGFof	29	30	29
	20	LGFof	28.5	30	29
	40	LGFof	28.5	30	29
	60	LGFof	28	30	29
	80	LGFof	28	29	29
	100	LGFof	28	30	29
	200	LGFof	28	29.5	29
	300	LGFof	28	29.5	29
	400	LGFof	28	29.5	29
	500	LGFof	28	29	29
200	10	LGFof	29	30	29
	20	LGFof	28.5	30.5	29
	40	LGFof	28	30	29
	60	LGFof	28	29.5	29
	80	LGFof	28	30	29
	100	LGFof	28	29	29
	200	LGFof	28	30	29
	300	LGFof	28	29	29
	400	LGFof	28	29.5	29
	500	LGFof	28	29.5	29
300	10	LGFof	29	29.5	29
	20	LGFof	28.5	30	29
	40	LGFof	28	30	29
	60	LGFof	28	29.5	29
	80	LGFof	28	29	29
	100	LGFof	28	29.5	29
	200	LGFof	28	30	29
	300	LGFof	28	29.5	29
	400	LGFof	28	29.5	29
	500	LGFof	28	29.5	29

Table 4.3: Result of population-generation for instance number 245 (Part 3)

Population	Generation	GA Algo(s)	GA Res(s)	UTS	LGFi
400	10	LGFof	29	31	29
	20	LGFof	29	30	29
	40	LGFof	28	29.5	29
	60	LGFof	28	29.5	29
	80	LGFof	28	29.5	29
	100	LGFof	28	29.5	29
	200	LGFof	28	29.5	29
	300	LGFof	28	29	29
	400	LGFof	28	29.5	29
	500	LGFof	28	30	29
500	10	LGFof	29	30.5	29
	20	LGFof	28	29.5	29
	40	LGFof	28	30	29
	60	LGFof	28	29	29
	80	LGFof	28	29.5	29
	100	LGFof	28	29.5	29
	200	LGFof	28	30	29
	300	LGFof	28	29	29
	400	LGFof	28	29.5	29
	500	LGFof	28	29.5	29

As a result of this experiment, we chose our population size as 60 and our generation number as 40.

4.2 A Tool for Visual Analysis of 2DBPPs

In this study, in order to analyze the results of packing heuristics and proposed algorithms we developed an application named as A Tool for Visual Analysis of 2D Bin Packing which is written in Java. We write the results of processed problem into a text file in a predefined format. Then, the result text file is given as input to the application. At the end, the application draws the position and shape of rectangles in the bins and prints them into graphical canvas.

4.3 MHO-SOMA

In order to solve single objective O-2DBPP, we applied MHO-SOMA. Experimental setup is composed of :

- population size is 60
- number of generation is 40
- swap mutation is used
- for each population generation step, swap mutation is applied for ten times
- swap mutation is applied for once in each offspring generation to keep diversity

The general results of experiment are listed in Table 4.4 and Table 4.5. The results are compared with best known solution lower bound of University of Bologna D.E.I.S. Operations Research [7].

Table 4.4: Result of MHO-SOMA for Berkey-Wang instances

Class	Num of Rect	lb	Result
1	20	71	71
	40	134	136
	60	197	202
	80	274	277
	100	317	323
	Av.	198.6	201.8
2	20	10	10
	40	19	20
	60	25	26
	80	31	32
	100	39	40
	Av.	24.8	25.6
3	20	51	54
	40	92	97
	60	136	144
	80	187	198
	100	221	233
	Av.	137.4	145.2
4	20	10	10
	40	19	19
	60	23	27
	80	30	34
	100	37	39
	Av.	23.8	25.8
5	20	65	66
	40	119	124
	60	179	186
	80	241	253
	100	279	294
	Av.	176.6	184.6
6	20	10	10
	40	15	19
	60	21	23
	80	30	30
	100	32	35
	Av.	21.6	23.4

Table 4.5: Result of MHO-SOMA for Martello-Vigo instances

Class	Num of Rect	lb	Pr. Algo
7	20	55	57
	40	109	115
	60	156	161
	80	224	232
	100	269	276
	Av.	162.6	168.2
8	20	58	59
	40	112	115
	60	159	166
	80	223	227
	100	274	284
	Av.	165.2	170.2
9	20	143	143
	40	278	278
	60	437	437
	80	577	577
	100	695	695
	Av.	426	426
10	20	42	43
	40	74	76
	60	98	106
	80	123	134
	100	153	164
	Av.	98	104.6

4.4 MHNO-SOMA

In order to solve single objective NO-2DBPP, we applied MHNO-SOMA. We have two different setups.

First experimental setup is composed of :

- population size is 60
- number of generation is 40
- rotation mutation is used

- for each population generation step, rotation mutation is applied for ten times
- rotation mutation is applied for once in each offspring generation to keep diversity

Second experimental setup is composed of :

- population size is 60
- number of generation is 40
- swap rotate mutation is used
- for each population generation step, swap rotate mutation is applied for ten times
- swap rotate mutation is applied for once in each offspring generation to keep diversity

The general results of two experiments are listed in Table 4.6 and Table 4.7. We used simplest continous lower bound which is described in Equation 4.1.

$$LB = \left\lceil \frac{\sum_{j=1}^{\#Rect} w_j h_j}{WH} \right\rceil \tag{4.1}$$

Table 4.6: Result of MHNO-SOMA for Berkey-Wang instances

Class	Num of Rect	clb	Pr. Algo(r)	Pr. Algo(sr)
1	20	64	66	66
	40	120	131	129
	60	185	196	195
	80	253	270	270
	100	305	314	313
	Av.	185.4	195.4	194.6
2	20	10	10	10
	40	19	20	19
	60	25	25	25
	80	31	31	31
	100	39	39	39
	Av.	24.8	25	24.8
3	20	44	48	48
	40	82	95	95
	60	125	137	137
	80	173	186	187
	100	205	225	225
	Av.	125.8	138.2	138.4
4	20	10	10	10
	40	19	19	19
	60	23	25	25
	80	30	32	33
	100	37	38	38
	Av.	23.8	24.8	25
5	20	54	59	59
	40	101	116	115
	60	157	175	176
	80	215	240	241
	100	259	284	284
	Av.	157.2	174.8	175
6	20	10	10	10
	40	15	18	17
	60	21	22	22
	80	30	30	30
	100	32	34	34
	Av.	21.6	22.8	22.6

Table 4.7: Result of MHNO-SOMA for Martello-Vigo instances

Class	Num of Rect	clb	Pr. Algo(r)	Pr. Algo(sr)
7	20	47	52	52
	40	97	106	107
	60	140	152	153
	80	197	216	217
	100	238	260	259
	Av.	143.8	157.2	157.6
8	20	48	53	53
	40	96	106	105
	60	141	155	154
	80	195	213	214
	100	241	261	262
	Av.	144.2	157.6	157.6
9	20	94	143	143
	40	180	275	275
	60	276	435	435
	80	371	573	573
	100	450	693	693
	Av.	274.2	423.8	423.8
10	20	38	41	41
	40	69	73	73
	60	94	101	101
	80	122	129	130
	100	153	161	162
	Av.	95.2	101	101.4

4.5 MHO-MOGA

In order to solve multiple objective O-2DBPP, we applied MHO-MOGA. Experimental setup is composed of :

- population size is 60
- number of generation is 40
- swap mutation is used
- for each population generation step, swap mutation is applied for ten times

- swap mutation is applied for once in each offspring generation to keep diversity

The general results of experiment are listed in Table 4.8 and Table 4.9. We used LGFof algorithm result in order to compare results.

Table 4.8: Result of MHO-MOGA for Berkey-Wang instances

Class	Num of Rect	LGFof b	LGFof cg	Pr. Algo b	Pr. Algo cg
1	20	74	268.1	71	117
	40	137	469.1	136	189.4
	60	202	748.1	202	408.6
	80	278	1114.6	277	545.7
	100	323	1188.5	323	616.5
	Av.	202.8	757.7	201.8	375.4
2	20	10	376.5	10	11.1
	40	20	646.4	20	6.8
	60	27	1024.2	26	150
	80	32	1533.7	32	178.6
	100	40	1854.6	40	152.2
	Av.	25.8	1087.1	25.6	99.7
3	20	56	1079	54	305.6
	40	102	2012.6	98	776.3
	60	146	2987.3	145	1144.5
	80	199	4375	198	1687.2
	100	236	5217.8	233	2353.5
	Av.	147.8	3134.3	145.6	1253.4
4	20	11	1542.3	10	115.2
	40	19	3469.3	19	432.3
	60	27	4758.2	27	277.5
	80	34	6017	33	794.9
	100	41	8005.5	40	741.7
	Av.	26.4	4758.5	25.8	472.3
5	20	66	2699.8	66	1126.2
	40	124	5891.1	123	2850.6
	60	187	7584.2	183	4045.6
	80	254	11502.6	251	5713.5
	100	296	12875	292	7311.6
	Av.	185.4	8110.5	183	4209.5
6	20	10	4790	10	879.1
	40	19	9357.9	19	228.9
	60	23	11447.4	23	612.2
	80	30	16052.8	30	1202.3
	100	36	20841	36	1875.9
	Av.	23.6	12497.8	23.6	959.7

Table 4.9: Result of MHO-MOGA for Martello-Vigo instances

Class	Num of Rect	LGFof b	LGFof cg	Pr. Algo b	Pr. Algo cg
7	20	58	4086.4	57	1200.2
	40	115	9133.5	115	2064.1
	60	161	12805.9	162	3594.5
	80	235	18095.9	234	4984.6
	100	276	22237.1	277	6297.1
	Av.	169	13271.8	169	3628.1
8	20	61	2150.7	60	935.8
	40	119	4482.7	115	2921.7
	60	168	5912.5	166	3827.4
	80	231	7719.8	230	6235.7
	100	286	9266.1	285	6835.6
	Av.	173	5906.4	171.2	4151.2
9	20	144	3718.7	143	2120.7
	40	278	7808.1	278	4280
	60	438	11027.8	437	7597.2
	80	577	15529.9	577	9894.4
	100	696	18976	695	12836.7
	Av.	426.6	11412.1	426	7345.8
10	20	44	3162.7	45	531.2
	40	77	5445.5	77	1721.7
	60	108	9310.4	105	2670.5
	80	134	10527.7	137	3331.5
	100	165	13949.1	167	5850.5
	Av.	105.6	8479.1	106.2	2821.1

4.6 MHNO-MOGA

In order to solve multi-objective NO-2DBPP, we applied MHNO-MOGA. We have two different setups.

First experimental setup is composed of :

- population size is 60
- number of generation is 40
- rotation mutation is used

- for each population generation step, rotation mutation is applied for ten times
- rotation mutation is applied for once in each offspring generation to keep diversity

The general results of first experiment are listed in Table 4.10 and Table 4.11. We used LGFi algorithm result in order to compare results.

Table 4.10: Result of MHNO-MOGA Ro for Berkey-Wang instances

Class	Num of Rect	LGFi b	LGFi cg	Pr. Algo b	Pr. Algo cg
1	20	67	230.9	68	67.5
	40	131	392.6	131	218.7
	60	199	620.5	197	359.6
	80	271	877.5	270	584.6
	100	317	1123.7	320	676.6
	Av.	197	649	197.2	381.4
2	20	10	256.8	10	1.7
	40	20	530.6	20	6.4
	60	25	993.6	25	160.7
	80	32	1107.9	31	134.1
	100	39	1078.7	39	195.7
	Av.	25.2	793.5	25	99.7
3	20	50	918.7	49	241.3
	40	97	1706.4	95	643.1
	60	139	2785.7	139	1073
	80	189	3805.2	192	1709
	100	227	4422.1	229	2211.8
	Av.	140.4	2727.6	140.8	1175.6
4	20	10	1794.8	10	0.8
	40	19	3940.2	19	8
	60	26	4109.7	25	174.8
	80	33	5558.4	33	162.9
	100	39	7418.4	40	360.7
	Av.	25.4	4564.3	25.4	141.4
5	20	62	2097.4	60	555.3
	40	117	5039.1	117	2167.1
	60	180	7031	177	3292
	80	246	9068.5	242	4477
	100	290	11640	286	5573.6
	Av.	179	6975.2	176.4	3213
6	20	10	4788.4	10	8
	40	19	13578.4	19	14.6
	60	23	15069.6	22	202.6
	80	30	20176	30	999.9
	100	35	25989.9	34	1995.2
	Av.	23.4	15920.5	23	644.1

Table 4.11: Result of MHNO-MOGA Ro for Martello-Vigo instances

Class	Num of Rect	LGFi b	LGFi cg	Pr. Algo b	Pr. Algo cg
7	20	55	1648.6	53	1145.2
	40	109	3717.4	107	2968.7
	60	161	6041	155	3869.6
	80	223	6989.4	221	6229.4
	100	271	9330.1	264	7259.2
	Av.	163.8	5545.3	160	4294.4
8	20	56	1858.4	55	873.5
	40	111	3877.2	107	2052.2
	60	163	5440.7	155	3055.2
	80	221	7640.6	214	4818.7
	100	269	8981.9	264	6412.7
	Av.	164	5559.8	159	3442.5
9	20	143	3012.3	143	2060.1
	40	275	6081.2	275	4432.6
	60	435	10005.9	435	7661.3
	80	573	13963.7	573	9919.7
	100	693	16765.3	693	12324
	Av.	423.8	9965.7	423.8	7279.5
10	20	41	3800.9	43	396.2
	40	75	6710.7	75	732.4
	60	104	9447.3	105	1885.1
	80	133	12136.1	134	2937.9
	100	163	15835.2	165	4615.5
	Av.	103.2	9586	104.4	2113.4

Second experimental setup is composed of :

- population size is 60
- number of generation is 40
- swap rotate mutation is used
- for each population generation step, swap rotate mutation is applied for ten times
- swap rotate mutation is applied for once in each offspring generation to keep diversity

The general results of second experiment are listed in Table 4.12 and Table 4.13. We used LGFi algorithm result in order to compare results.

Table 4.12: Result of MHNO-MOGA SwRo for Berkey-Wang instances

Class	Num of Rect	LGFi b	LGFi cg	Pr. Algo b	Pr. Algo cg
1	20	67	230.9	66	76.9
	40	131	392.6	125	202.8
	60	199	620.5	200	359.2
	80	271	877.5	264	630.7
	100	317	1123.7	320	711.7
	Av.	197	649	195	396.3
2	20	10	256.8	10	1
	40	20	530.6	20	8.7
	60	25	993.6	25	133
	80	32	1107.9	31	131.8
	100	39	1078.7	39	245.5
	Av.	25.2	793.5	25	104
3	20	50	918.7	49	260.3
	40	97	1706.4	97	581.1
	60	139	2785.7	141	1165.3
	80	189	3805.2	190	1775.7
	100	227	4422.1	228	2335.5
	Av.	140.4	2727.6	141	1223.6
4	20	10	1794.8	10	0.5
	40	19	3940.2	19	22.6
	60	26	4109.7	25	97.1
	80	33	5558.4	33	346.4
	100	39	7418.4	39	633.6
	Av.	25.4	4564.3	25.2	220
5	20	62	2097.4	60	719.4
	40	117	5039.1	116	2027
	60	180	7031	177	3444.2
	80	246	9068.5	245	5043.8
	100	290	11640	286	5154.2
	Av.	179	6975.2	176.8	3277.7
6	20	10	4788.4	10	4.9
	40	19	13578.4	19	51.3
	60	23	15069.6	22	112.3
	80	30	20176	30	502
	100	35	25989.9	34	1356.8
	Av.	23.4	15920.5	23	405.5

Table 4.13: Result of MHNO-MOGA SwRo for Martello-Vigo instances

Class	Num of Rect	LGFi b	LGFi cg	Pr. Algo b	Pr. Algo cg
7	20	55	1648.6	53	1015.6
	40	109	3717.4	108	2612.5
	60	161	6041	156	3880.6
	80	223	6989.4	220	6126.4
	100	271	9330.1	267	6899.4
	Av.	163.8	5545.3	160.8	4106.9
8	20	56	1858.4	53	785.3
	40	111	3877.2	107	2170.4
	60	163	5440.7	155	3044
	80	221	7640.6	214	4521.9
	100	269	8981.9	265	6237.6
	Av.	164	5559.8	158.8	3351.8
9	20	143	3012.3	143	1885.9
	40	275	6081.2	275	4241.6
	60	435	10005.9	435	6762.7
	80	573	13963.7	573	9355.1
	100	693	16765.3	693	11598.4
	Av.	423.8	9965.7	423.8	6768.7
10	20	41	3800.9	42	263.2
	40	75	6710.7	75	971
	60	104	9447.3	102	2616.8
	80	133	12136.1	133	3837.2
	100	163	15835.2	166	4662.4
	Av.	103.2	9586	103.6	2470.1

4.7 Analysis of Algorithms

In order to analyze runtime and efficiency of algorithms, we randomly picked five different item size (20, 40, 60, 80, 100) problems. Each test is runned for five times. Best values of FNF, FFF, BFDH, LGFof and LGFi are used as results. For proposed algorithms and UTS, we used average values of tests.

In each subsection, comparisons of algorithms according to 500 instance test setup are also explained in detail.

4.7.1 Oriented Single Objective Problems

Runtime analysis of FNF, FFF, BFDH, UTS, LGFof and MHO-SOMA for oriented single objective random picked tests are shown in Table 4.14:

Table 4.14: Runtime of algorithms for orientated single objective problems in msec

Rect	FNF	FFF	BFDH	UTS	LGFof	MHO-SOMA
20	4.0	4.5	4.6	81.7	64.0	21748
40	4.4	4.6	4.9	317.4	29.7	47196
60	4.8	6.0	7.7	27761	1529.5	754233
80	7.4	8.3	9.1	3050	334.5	25486
100	10.1	12.1	17.2	3746	291.0	582892
Av.	6.1	7.1	8.7	6991.2	449.7	286311

Result (bin) analysis of FNF, FFF, BFDH, UTS, LGFof and MHO-SOMA for oriented single objective random picked tests are shown in Table 4.15:

Table 4.15: Results (bin) of algorithms for orientated single objective problems

Rect	FNF	FFF	BFDH	UTS	LGFof	MHO-SOMA
20	11	9	9	9	9	8
40	17	13	13	13	13	12
60	53	47	47	47	47	46
80	30	24	23	24	24	23
100	44	31	31	30	30	30
Av.	31	24.8	24.6	24.6	24.6	23.8

Extra bin usage of FNF, FFF, BFDH, UTS, LGFof and MHO-SOMA for oriented single objective 500 problem set to reach results Bologna University DEIS Operations Research are shown in Table 4.16:

Table 4.16: Comparisons of heuristics and MHO-SOMA for orientated single objective 500 problem set

Bin Sol.	FNF	FFF	BFDH	UTS	LGFof	Pr. Algo.
No ex.	16.8%	42.4%	45.4%	46.2%	57.2%	63.4%
1 ex.	20%	36.2%	38.6%	38.6%	34.8%	32%
2 ex.	6%	16%	15%	14.2%	7.6%	4.4%
3 ex.	9.2%	5.2%	1%	1%	0.6%	0.2%
3+ ex.	48%	0.2%	0%	0%	0%	0%

4.7.2 Non-Oriented Single Objective Problems

Runtime analysis of FNF, FFF, BFDH, UTS, LGFof, LGFi and MHNO-SOMA for non-oriented single objective random picked tests are shown in Table 4.17:

Table 4.17: Runtime of algorithms for non-orientated single objective problems in msec

Rect	FNF	FFF	BFDH	UTS	LGFof	LGFi	MHNO-SOMA
20	4.0	4.5	4.6	81.7	64.0	65.0	101612
40	4.4	4.6	4.9	317.4	29.7	30.0	46640
60	4.8	6.0	7.7	27761	1529.5	2416.4	873217
80	7.4	8.3	9.1	3050	334.5	257.2	411683
100	10.1	12.1	17.2	3746	291.0	324.1	645343
Av.	6.1	7.1	8.7	6991.2	449.7	618.5	415699

Result (bin) analysis of FNF, FFF, BFDH, UTS, LGFof, LGFi and MHNO-SOMA for non-oriented single objective random picked tests are shown in Table 4.18:

Table 4.18: Results (bin) of algorithms for non-orientated single objective problems

Rect	FNF	FFF	BFDH	UTS	LGFof	LGFi	MHNO-SOMA
20	11	9	9	9	9	8	8
40	17	13	13	13	13	12	11
60	53	47	47	47	47	46	46
80	30	24	23	24	24	22	21
100	44	31	31	30	30	29	28
Av.	31	24.8	24.6	24.6	24.6	23.4	22.8

Extra bin usage of FNF, FFF, BFDH, UTS, LGFof, LGFi and MHNO-SOMA (r) and MHNO-SOMA (sr) for oriented single objective 500 problem set ro reach results of continuous lower bound are shown in Table 4.19 and Table 4.20:

Table 4.19: Comparisons of heuristics and MHNO-SOMA (r) for non-orientated single objective 500 problem set

Bin	FNF	FFF	BFDH	UTS	LGFof	LGFi	Pr. Algo.(r)
No ex.	16%	24.6%	25.6%	21.4%	27.4%	31.8%	41.4%
1 ex.	15.8%	19.2%	20.2%	23.6%	23%	27%	29%
2 ex.	5%	17%	17.4%	19.6%	17.8%	15.4%	14.4%
3 ex.	4.6%	12.2%	12.8%	13.6%	11.8%	12.2%	3.8%
3+ ex.	58.6%	27%	24%	21.6%	20.4%	13.6%	11.4%

Table 4.20: Comparisons of heuristics and MHNO-SOMA (sr) for non-orientated single objective 500 problem set

Bin	FNF	FFF	BFDH	UTS	LGFof	LGFi	Pr. Algo.(sr)
No ex.	16%	24.6%	25.6%	21.4%	27.4%	31.8%	41.8%
1 ex.	15.8%	19.2%	20.2%	23.6%	23%	27%	28.8%
2 ex.	5%	17%	17.4%	19.6%	17.8%	15.4%	14%
3 ex.	4.6%	12.2%	12.8%	13.6%	11.8%	12.2%	4.2%
3+ ex.	58.6%	27%	24%	21.6%	20.4%	13.6%	11.2%

4.7.3 Oriented Multi-Objective Problems

Runtime analysis of FNF, FFF, BFDH, UTS, LGFof and MHO-MOGA for oriented multi-objective random picked tests are shown in Table 4.21:

Table 4.21: Runtime of algorithms for orientated multi-objective problems in msec

Rect	FNF	FFF	BFDH	UTS	LGFof	MHO-MOGA
20	4.1	4.6	4.7	81.7	64.0	17515
40	4.7	4.8	5.0	317.8	29.9	88024
60	5.3	6.4	7.8	27761.4	1529.8	3034228
80	7.6	8.4	9.2	3050.4	334.6	4664450
100	10.6	12.4	17.4	3746.9	291.1	677750
Av.	6.5	7.3	8.8	6991.6	449.9	1696393.4

Result (bin/cg) analysis of FNF, FFF, BFDH, UTS, LGFof and MHO-MOGA for oriented multi-objective random picked tests are shown in Table 4.22:

Table 4.22: Results (bin/cg) of algorithms for orientated multi-objective problems

Rect	FNF	FFF	BFDH	UTS	LGFof	MHO-MOGA
20	11	9	9	9	9	9
	310.4	167.3	151.3	491.1	301.4	116.6
40	17	13	13	13	12	12
	62.4	59.8	50	64.5	33.2	19.6
60	53	47	47	47	47	46
	1207.5	1024.7	1008.5	1103.9	1021.1	813.7
80	30	24	23	24	24	24
	1704.5	1850.1	1440.7	900.9	1728.7	421
100	44	31	31	30	30	29
	1623.9	1391.2	1098.2	1088.9	1331.2	769.9
Av.	31	24.8	24.6	24.6	24.6	24
	981.5	898.6	749.7	729.9	883.1	428.2

Results (bin/cg) of FNF, FFF, BFDH, UTS, LGFof and MHO-MOGA for oriented single objective 500 problem set are shown in Table 4.23:

Table 4.23: Results (bin/cg) of heuristics and MHO-MOGA for orientated multi-objective 500 problem set

Total	FNF	FFF	BFDH	UTS	LGFof	Pr. Alg.
Bin	9489	7591	7514	7521	7430	7389
CG	340954	333247	319187	297483	347076	126581

Superiority of MHO-MOGA to FNF, FFF, BFDH, UTS and LGFof for oriented multi-objective 500 problem set are shown in Table 4.24:

Table 4.24: MHO-MOGA vs. heuristics for orientated multi-objective 500 problem set

	FNF	FFF	BFDH	UTS	LGFof
MHO-MOGA	100%	100%	100%	100%	97.2%

4.7.4 Non-Oriented Multi-Objective Problems

Runtime analysis of FNF, FFF, BFDH, UTS, LGFof, LGFi and MHNO-MOGA for non-oriented multi-objective random picked tests are shown in Table 4.25:

Table 4.25: Runtime of algorithms for non-orientated multi-objective problems in msec

Rect	FNF	FFF	BFDH	UTS	LGFof	LGFi	MHNO-MOGA
20	4.1	4.6	4.7	81.7	64.0	65.0	87615
40	4.7	4.8	5.0	317.8	29.9	30.1	39027
60	5.3	6.4	7.8	27761.4	1529.8	2416.4	1545971
80	7.6	8.4	9.2	3050.4	334.6	257.4	645442
100	10.6	12.4	17.4	3746.9	291.1	324.3	257878
Av.	6.5	7.3	8.8	6991.6	449.9	618.6	515186.6

Result (bin/cg) analysis of FNF, FFF, BFDH, UTS, LGFof, LGFi and MHNO-MOGA for non-oriented multi-objective random picked tests are shown in Table 4.26:

Table 4.26: Results (bin/cg) of algorithms for non-orientated multi-objective problems

Rect	FNF	FFF	BFDH	UTS	LGFof	LGFi	MHNO-MOGA
20	11 310.4	9 167.3	9 151.3	9 491.1	9 301.4	8 150.9	8 60
40	17 62.4	13 59.8	13 50	13 64.5	12 33.2	12 51.9	12 15
60	53 1207.5	47 1024.7	47 1008.5	47 1103.9	47 1021.1	46 901.4	46 700.2
80	30 1704.5	24 1850.1	23 1440.7	24 900.9	24 1728.7	22 887.4	21 821.3
100	44 1623.9	31 1391.2	31 1098.2	30 1088.9	30 1331.2	29 1131.2	28 575
Av.	31 981.5	24.8 898.6	24.6 749.7	24.6 729.9	24.6 883.1	23.4 624.6	23 434.3

Results (bin/cg) of FNF, FFF, BFDH, UTS, LGFof, LGFi and MHNO-MOGA (r) for non-oriented multi-objective 500 problem set (according to continuous lower bound) are shown in Table 4.27:

Table 4.27: Results (bin/cg) of heuristics and MHNO-MOGA (r) for non-orientated multi-objective 500 problem set

Total	FNF	FFF	BFDH	UTS	LGFof	LGFi	Pr. Alg.
Bin	9489	7591	7514	7521	7430	7226	7175
CG	340954	333247	319187	297483	347076	311434	113925

Superiority of MHO-MOGA (r) vs. FNF, FFF, BFDH, UTS, LGFof and LGFi for non-oriented multi-objective 500 problem set are shown in Table 4.28:

Table 4.28: MHO-MOGA(r) vs. heuristics for non-orientated multi-objective 500 problem set

	FNF	FFF	BFDH	UTS	LGFof	LGFi
MHNO-MOGA(r)	100%	100%	100%	100%	100%	95%

Results (bin/cg) of FNF, FFF, BFDH, UTS, LGFof, LGFi and MHNO-MOGA (sr) for non-oriented multi-objective 500 problem set (according to continuous lower bound) are shown in Table 4.27:

Table 4.29: Results (bin/cg) of heuristics and MHNO-MOGA(sr) for non-orientated multi-objective 500 problem set

Total	FNF	FFF	BFDH	UTS	LGFof	LGFi	Pr. Alg.
Bin	9489	7591	7514	7521	7430	7226	7165
CG	340954	333247	319187	297483	347076	311434	111623

Superiority of MHNO-MOGA (sr) vs. FNF, FFF, BFDH, UTS, LGFof and LGFi for non-oriented multi-objective 500 problem set are shown in Table 4.30:

Table 4.30: MHO-MOGA (sr) vs. heuristics for non-orientated multi-objective 500 problem set

	FNF	FFF	BFDH	UTS	LGFof	LGFi
MHNO-MOGA(sr)	100%	100%	100%	100%	100%	96%

4.8 Visual Analysis of Algorithms

In order to visually analyze proposed algorithms, we randomly picked a problem in our problem set. Our problem (instance) number is 108. It is a Berkey-Wang class 3 problem with item size as 20. Bin width and bin height of the problem is 40. Rectangle list of the problem is listed in Table 4.31:

Table 4.31: Rectangle list of instance number 108 with bin width = 40 and bin height = 40

Rectangle	1	2	3	4	5	6	7	8	9	10
Width	30	13	7	29	7	19	13	21	16	17
Height	15	3	27	12	2	14	12	2	4	14
Rectangle	11	12	13	14	15	16	17	18	19	20
Width	19	32	5	2	33	12	5	19	4	27
Height	4	30	17	8	23	23	15	19	31	23

First, we considered the problem as an oriented single objective bin packing problem so rotation of rectangles was not allowed and our objective is minimizing the number of bins. Then, we tested it with FNF, FFF, UTS and MHO-SOMA. Results of test are listed in Table 4.32:

Table 4.32: Results of instance number 108 as single objective BPP

	FNF	FFF	UTS	MHO-SOMA
Num of Bin	7	5	5	4

All results of the test is drawn by the use of our application which is described in Section 4.2.

Visual analysis of FNF for the test is shown in Figure 4.1 and Figure 4.1:

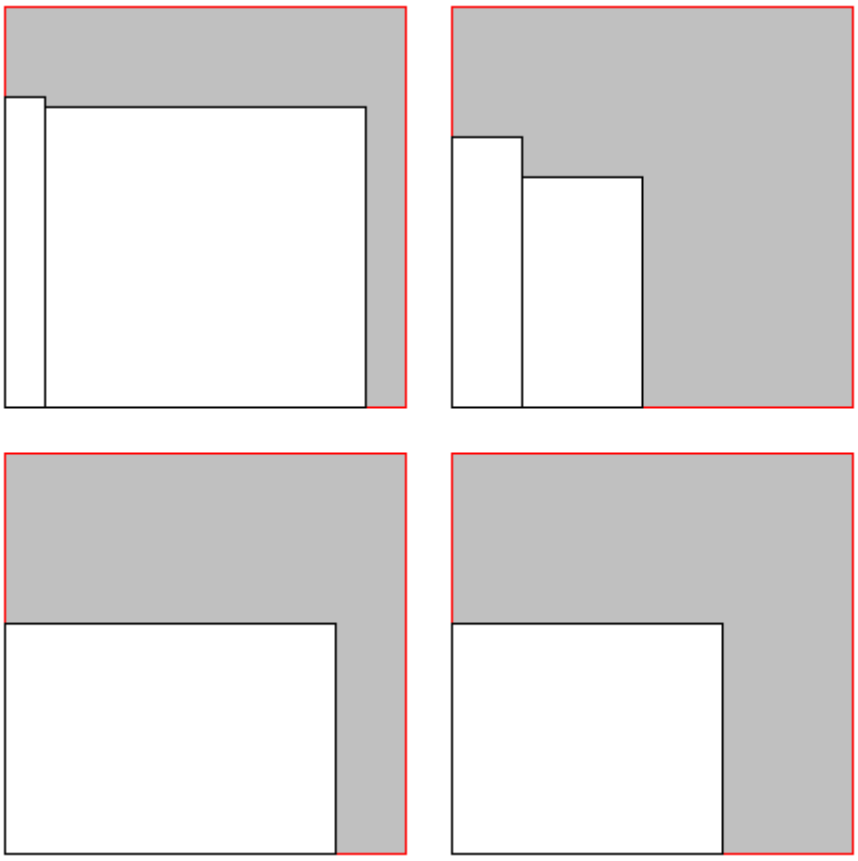

Figure 4.1: Visual analysis of FNF (part 1)

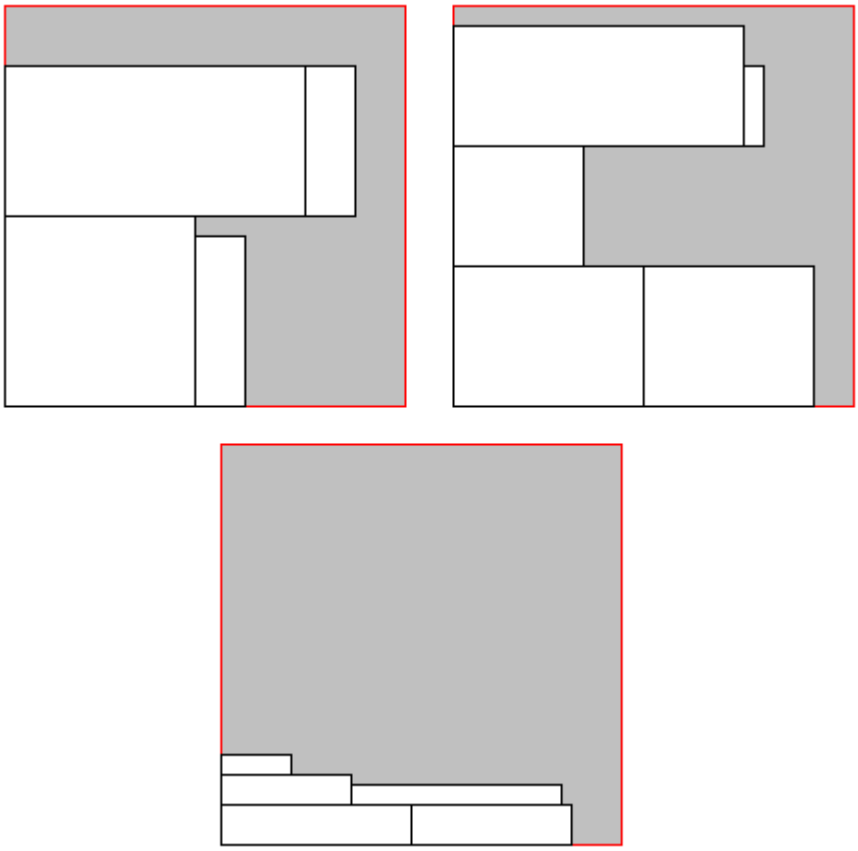

Figure 4.2: Visual analysis of FNF (part 2)

Visual analysis of FNF for the test is shown in Figure 4.1 and Figure 4.1:

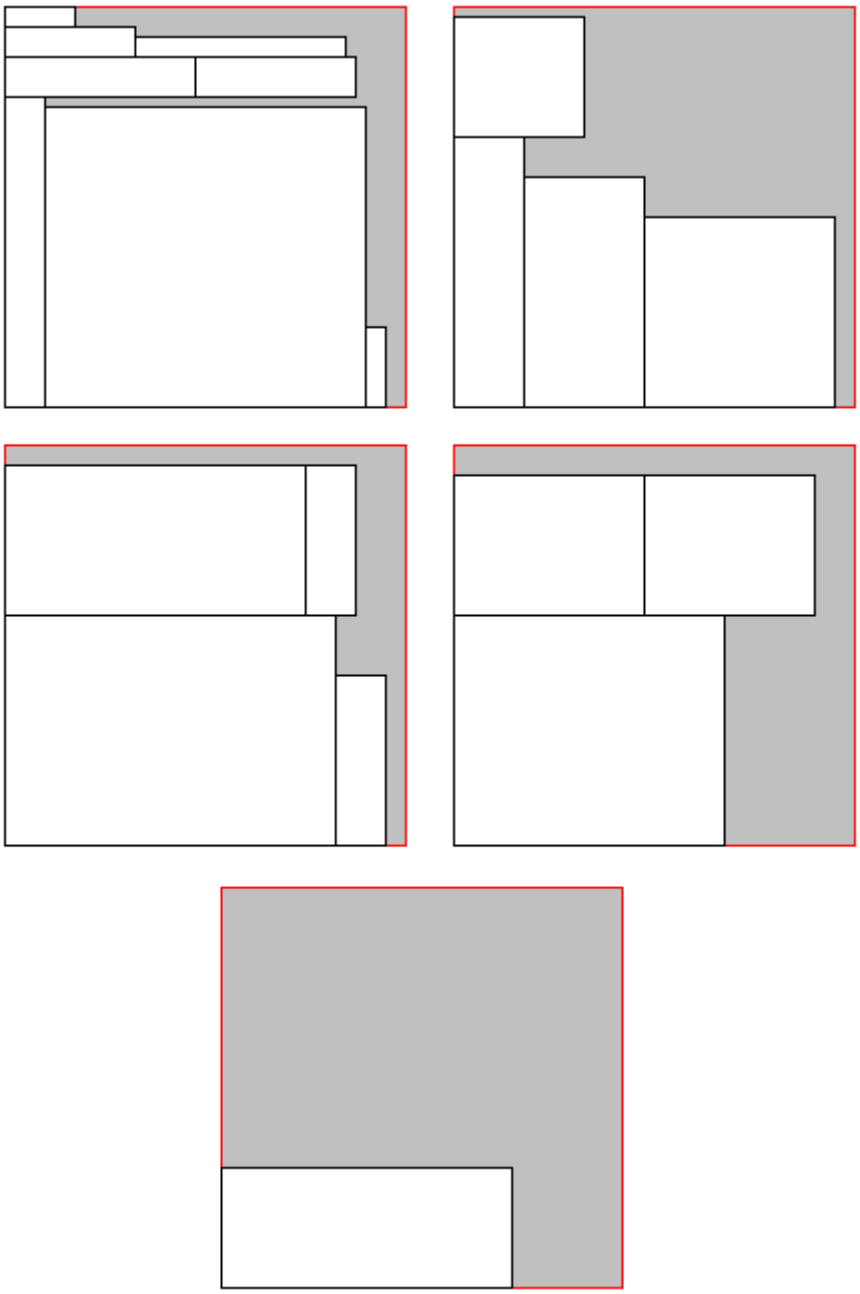

Figure 4.3: Visual analysis of FFF

Visual analysis of UTS for the test is shown in Figure 4.4:

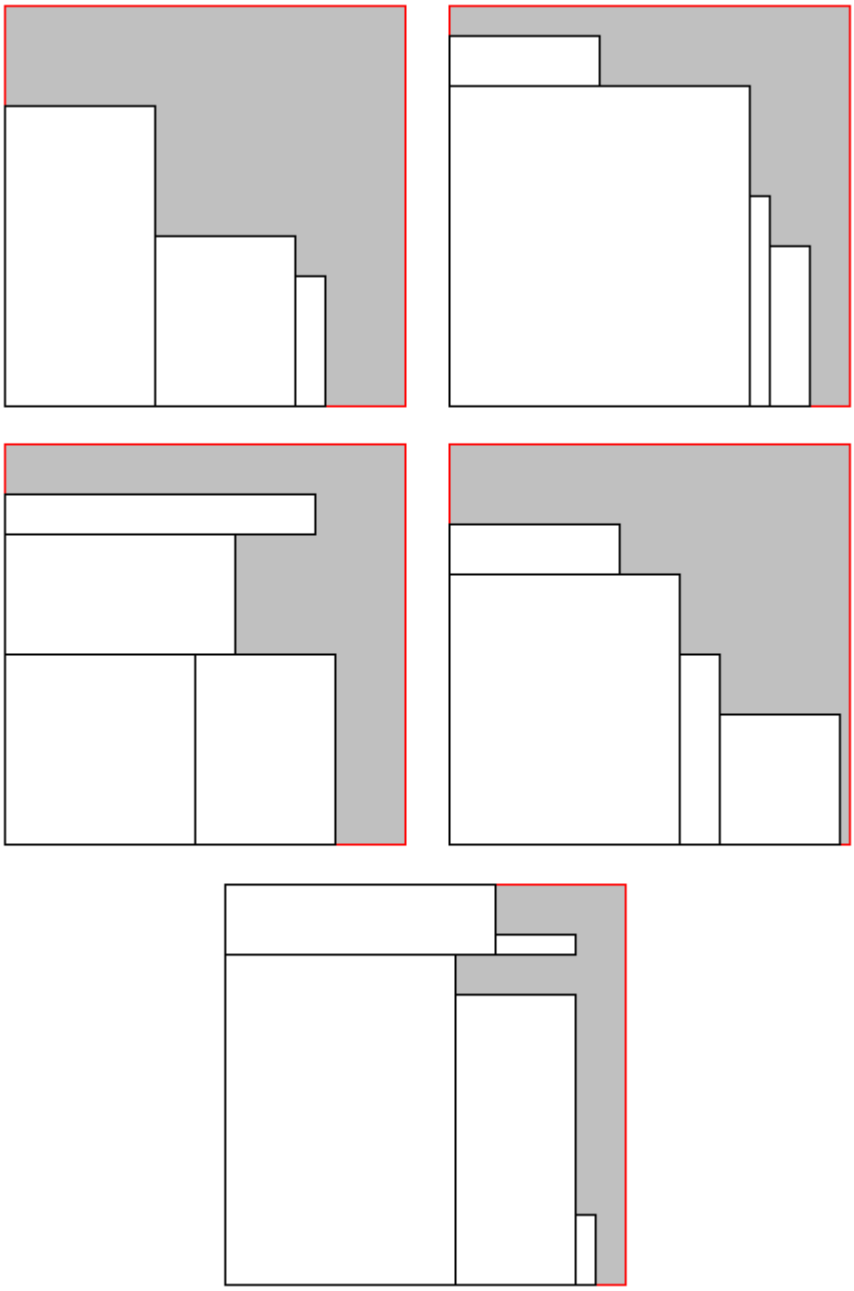

Figure 4.4: Visual analysis of UTS

Visual analysis of our proposed algorithm MHO-SOMA for the test is shown in Figure 4.5:

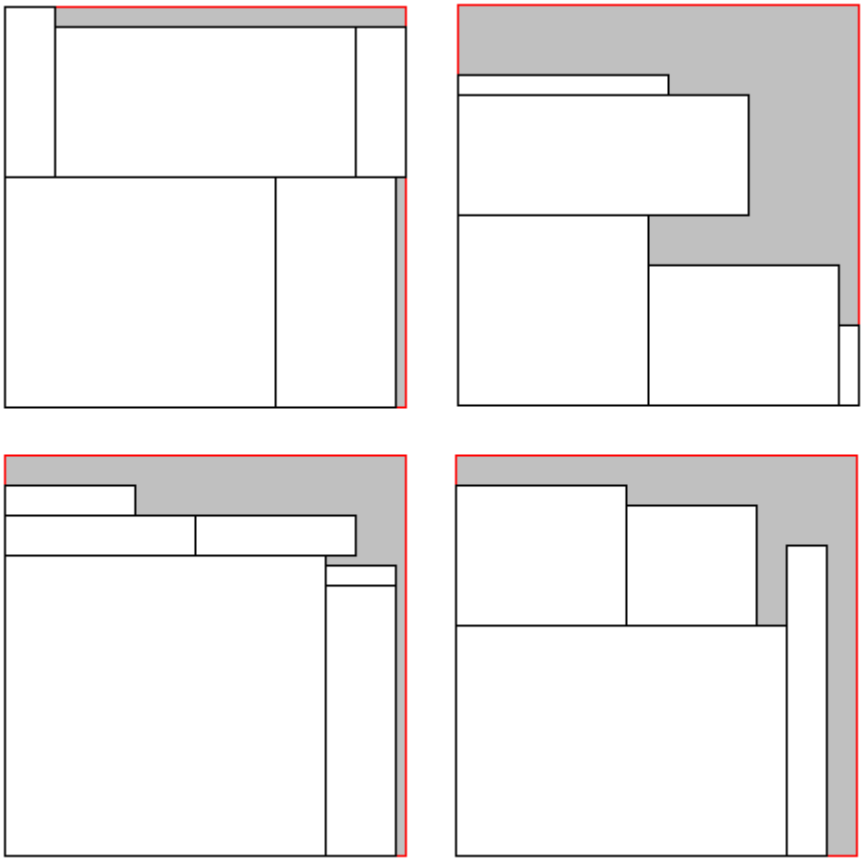

Figure 4.5: Visual analysis of MHO-SOMA

CHAPTER 5

CONCLUSION AND FUTURE WORK

Bin packing problems are NP-hard combinatorial optimization problems. Engineering, manufacturing, transportation and logistics are dealing with these problems. In this study, we developed four algorithms for two variant (single objective and multiobjective) of 2D offline bin packing problems and tested 500 benchmark problems for each developed algorithm.

First, we tried to minimize number of bins to be packed. Two algorithms are proposed to solve this single objective 2DBPP. Our first algorithm is MHO-SOMA. It tries to minimize the number of bins for orientated 2D offline BPP. MHO-SOMA achieves optimum solutions (according to Bologna University DEIS Operations Research) for 63.4% of the problems with no extra bin, 32.0% of the problems with extra one bin and 4.4% of the problems with extra two bins.

Second single objective algorithm is MHNO-SOMA which solves non-oriented single objective 2D offline BPP. MHNO-SOMA is using two different mutation operation. First mutation operation is rotation and second mutation operation is swap-mutation. Each mutation operation is applied for 500 benchmark problems. MHNO-SOMA with rotation mutation achieves optimum solutions (according to continuous lower bound) for 41.4% of the problems with no extra bin, 29.0% of the problems with extra one bin and 14.4% of the problems with extra two bins. MHNO-SOMA with swap-rotation mutation achieves optimum solutions (according to continuous lower bound) for 41.8% of the problems with no extra bin, 28.8% of the problems with extra one bin and 14% of the problems with extra two bins. Experimental results show that the combination of global search of well known heuristics with individual

learning (local search of UTS) can achieve optimal results with a reasonable population and number of generations.

For real world applications, load balancing is as important as minimizing number of bins, so we proposed two different algorithms in order to solve multi-objective (bin size-load balancing) 2D offline bin packing problems. First one is MHO-MOGA and second one is MHNO-MOGA. Computational results show that well known heuristics sometimes produce better results about number of bins but not about load balancing. Our proposed MHO-MOGA and MHNO-MOGA give better results about not only number of bins but also load balancing of bins. Analysis of algorithms also shows that proposed algorithms produce better results than heuristics but the runtime of heuristics are shorter than the proposed algorithms. MHO-MOGA achieves better solutions for 97.2% of the problems than LGFof heuristic. MHNO-MOGA is also using two different mutation operation which are rotation and swap-rotation. MHNO-MOGA with rotation mutation achieves better solutions for 95.0% of the problems than LGFi heuristic and MHNO-MOGA with swap-rotation mutation achieves better solutions for 96.0% of the problems than LGFi heuristic.

With this study, we have proposed a set of novel metaheuristic algorithms for NP-hard combinatorial optimization problem, 2DBPP. Future works of this study will have two categories. First category is integrating other well known heuristics such as Touching Perimeter and Floor-Ceiling into the proposed algorithms. Second one is achieving truly parallelization of MH-SOMA and MH-MOGA.

REFERENCES

[1] Karp R. M., Reducibility among combinatorial problems, In *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art*, pages 219–241, Springer, 2010.

[2] Johnson D. S., *Near-optimal bin-packing algorithms*, PhD thesis, 1973, PHD.

[3] Blum C. and Schmid V., Solving the 2d bin packing problem by means of a hybrid evolutionary algorithm, *Procedia Computer Science*, 18(0):899 – 908, 2013, 2013 International Conference on Computational Science.

[4] Fernández A., Gil C., Baños R., and Montoya M. G., A parallel multi-objective algorithm for two-dimensional bin packing with rotations and load balancing, *Expert Systems with Applications*, 40(13):5169 – 5180, 2013.

[5] Dokeroglu T. and Cosar A., Optimization of one-dimensional bin packing problem with island parallel grouping genetic algorithms, *Computers & Industrial Engineering*, 75(0):176 – 186, 2014.

[6] Tosun U., Dokeroglu T., and Cosar A., A robust island parallel genetic algorithm for the quadratic assignment problem, *International Journal of Production Research*, 51(14):4117 – 4133, 2013.

[7] University of bologna deis operations research, `http://www.or.deis.unibo.it/research_pages/ORinstances/soluz_2dbp.htm`, last modified in 21 February 2006, last visited on 17 December 2014.

[8] Bennell J. A., Lee L.S., and Potts C. N., A genetic algorithm for two-dimensional bin packing with due dates, *International Journal of Production Economics*, 145(2):547 – 560, 2013.

[9] Bennell J. A., Soon Lee L. S., and Potts C. N., A genetic algorithm for two-dimensional bin packing with due dates, *International Journal of Production Economics*, 145(2):547 – 560, 2013.

[10] Lim A., Ma H., Qiu C., and Zhu W., The single container loading problem with axle weight constraints, *International Journal of Production Economics*, 144(1):358 – 369, 2013.

[11] Lodi A., Martello S., and Vigo D., Heuristic and Metaheuristic Approaches

for a Class of Two-Dimensional Bin Packing Problems, *Informs Journal on Computing*, 11:345–357, 1999.

[12] Lodi A., Martello S., and Vigo D., TSpack: A Unified Tabu Search Code for MultiDimensional Bin Packing Problems, *Annals of Operations Research*, 131:203–213, 2004.

[13] Kröger B., Schwenderling P., and Vornberger O., Parallel genetic packing of rectangles, In *Proceedings of the 1st Workshop on Parallel Problem Solving from Nature*, PPSN I, pages 160–164, London, UK, UK, 1991. Springer-Verlag.

[14] Smith D., Bin Packing with Adaptive Search, In *International Conference on Genetic Algorithms*, pages 202–207, 1985.

[15] Whitley D., Gordon V. S., and Mathias K., Lamarckian evolution, the baldwin effect and function optimization, *Parallel Problem Solving from Nature PPSN III, Lecture Notes in Computer Science*, 866(0):5–15, 1994.

[16] Goldberg D. E., *Genetic Algorithms in Search Optimization and Machine Learning*, Addison-Wesley, 1989.

[17] Hopper E. and Turton B., A genetic algorithm for a 2D industrial packing problem, *Computers & Industrial Engineering*, 37:375–378, 1999.

[18] López-Camacho E., Terashima-Marín H., Ochoa G., and Conant-Pablos S. E., Understanding the structure of bin packing problems through principal component analysis, *International Journal of Production Economics*, 145(2):488 – 499, 2013.

[19] Glover F., Future paths for integer programming and links to artificial intelligence, *Computers & Operations Research*, 13:533–549, 1986.

[20] Glover F., Tabu Search - Part I, *Informs Journal on Computing*, 1:190–206, 1989.

[21] Glover F., Tabu Search - Part II, *Informs Journal on Computing*, 2:4–32, 1990.

[22] Gonçalves J. F. and Resende M. G. C., A biased random key genetic algorithm for 2d and 3d bin packing problems, *International Journal of Production Economics*, 145(2):500 – 510, 2013.

[23] Coffman E. G. and Shor P. W., Average-case analysis of cutting and packing in two dimensions, *European Journal of Operational Research*, 44:134–144, 1990.

[24] Kendall G., Soubeiga E., and Cowling P., Choice function and random hyper-

heuristics, In *Proceedings of the fourth Asia-Pacific Conference on Simulated Evolution And Learning, SEAL*, pages 667–671. Springer, 2002.

[25] Holland J. H., *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI, 1975, second edition, 1992.

[26] Bremermann H. J., The evolution of intelligence. the nervous system as a model of its environment, Technical Report 1, Department of Mathematics, Washington University, Washington, Seattle, July 1958.

[27] Smith J.E., Coevolving memetic algorithms: A review and progress report, *Systems, Man, and Cybernetics, Part B: Cybernetics*, 37(1):6–17, 2007.

[28] Coffman E. G. Jr., Garey M. R., Johnson D. S., and Tarjan R. E., Performance Bounds for Level-Oriented Two-Dimensional Packing Algorithms, *Siam Journal on Computing*, 9:808–826, 1980.

[29] Burke E. K., Kendall G., and Whitwell G., A New Placement Heuristic for the Orthogonal Stock-Cutting Problem, *Operations Research*, 52:655–671, 2004.

[30] Wong L. and Lee L. S., Heuristic Placement Routines for Two-Dimensional Bin Packing Problem, *Journal of Mathematics and Statistics*, 5:334–341, 2009.

[31] Lee L.S., A genetic algorithms for two dimensional bin packing problem, *MathDigest. Res. Bull. Inst. Math. Res*, 2:34 – 39, 2008.

[32] A. Meir and L. Moser., On packing of squares and cubes, *Journal of Combinatorial Theory*, 5(2):126 – 134, 1968.

[33] Krasnogor N., Hybrid evolutionary approaches to terminal assignment in communications networks, *Graduate Student Workshop*, 371, 1999.

[34] Krasnogor N. and Gustafson S., Toward truly "memetic" memetic algorithms: discussion and proofs of concept, In *Advances in Nature-Inspired Computation: The PPSN VII Workshops. PEDAL (Parallel, Emergent and Distributed Architectures Lab). University of Reading*, pages 16–52, 2002.

[35] Berkey J. O. and Wang P. Y., Two-Dimensional Finite Bin-Packing Algorithms, *Journal of The Operational Research Society*, 38:423–429, 1987.

[36] Merz P. and Freisleben B., A Genetic Local Search Approach to the Quadratic Assignment Problem, In *International Conference on Genetic Algorithms*, pages 465–472, 1997.

[37] Moscato P., On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts - Towards Memetic Algorithms, 1989.

[38] Moscato P. and Cotta C., A gentle introduction to memetic algorithms, In Glover F. and Kochenberger G. A., editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 105–144. Springer US, 2003.

[39] Thapatsuwan P.and Pongcharoen P., Hicks C., and Chainate W., Development of a stochastic optimisation tool for solving the multiple container packing problems, *International Journal of Production Economics*, 140(2):737 – 748, 2012.

[40] Johnson D. S., Demers A. J., Ullman J. D., Garey M. R., and Graham R. L., Worst-Case Performance Bounds for Simple One-Dimensional Packing Algorithms, *Siam Journal on Computing*, 3:299–325, 1974.

[41] Martello S. and Vigo D., Exact Solution of the Two-Dimensional Finite Bin Packing Problem, *Management Science*, 1998.

[42] Ong Y. S. and Keane A. J., Meta-lamarckian learning in memetic algorithms, *Evolutionary Computation*, 8(2):99–110, 2004.

[43] Wolf S. and Merz P., A hybrid method for solving large-scale supply chain problems, In Cotta C. and van Hemert J., editors, *Evolutionary Computation in Combinatorial Optimization*, volume 4446 of *Lecture Notes in Computer Science*, pages 219–228. Springer Berlin Heidelberg, 2007.

[44] Fischer T. and Merz P., A memetic algorithm for the optimum communication spanning tree problem, In Bartz-Beielstein T., Blesa Aguilera M. J., Blum C., Naujoks B., Roli A., Rudolph G., and Sampels M., editors, *Hybrid Metaheuristics*, volume 4771 of *Lecture Notes in Computer Science*, pages 170–184. Springer Berlin Heidelberg, 2007.

[45] Yao X., Wang F., Padmanabhan K., and S. Salcedo-Sanz S., Hybrid evolutionary approaches to terminal assignment in communications networks, *Recent Advances in Memetic Algorithms, Studies in Fuzziness and Soft Computing*, 166(0):129–159, 2005.

[46] Bansal N., and Khan A., Improved approximation algorithm for two-dimensional bin packing, In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA 2014, pages 13–25, SIAM, 2014.