T.C.

TED UNIVERSITY

GRADUATE SCHOOL

INTERACTIVE COMPUTING AND INFORMATION SYSTEMS

EFFECTS OF EVOLUTIONARY OPERATORS IN GROUPING GENETIC ALGORITHMS ON DIVERSITY AND RESULT QUALITY

GÖKTUĞ YORGANCILAR

ANKARA, AUGUST 2020

EFFECTS OF EVOULUTIONARY OPERATORS IN GROUPING GENETIC ALGORITHMS ON DIVERSITY AND RESULT QUALITY

A Thesis Submited To
The Graduate School
of
TED University

by

Göktuğ YORGANCILAR

In Partial Fulfillment of The Requirements
For
The Degree of Master Of Science
In
Interactive Computing and Information Systems

Ankara, August 2020

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: Göktuğ Yorgancılar

Signature :

ABSTRACT

EFFECTS OF EVOULUTIONARY OPERATORS IN GROUPING GENETIC ALGORITHMS ON DIVERSITY AND RESULT QUALITY

GÖKTUĞ YORGANCILAR
M.S., Interactive Computing and Information Systems
Supervisor : Asst. Prof. Dr. Tayfun Kucukyilmaz

AUGUST 2020

Genetic algorithms are evolutionary metaheuristic algorithms that are used to solve complex problems. Although many variations exist in the literature, a common theme in all genetic algorithms is the application of a set of evolutionary operators to a set of random solutions in an iterative fashion. These evolutionary operators aim to generate new solutions in a semi-random manner in order to achieve the best possible solution. The effectiveness of evolutionary operators is largely dependent on two competing factors, namely exploration and exploitation of the problem space.

Although various genetic algorithm operators and their modifications are proposed and compared in the literature, how different operators perform together in terms of exploration and exploitation is not analyzed. We reduce exploration and exploitation to diversity and fitness of the solution set and examine the interactions between various components of a genetic algorithm in detail. For that purpose, we select grouping genetic algorithms which are specialized versions of genetic

algorithms for grouping problems such as bin packing and line balancing problems as our domain. Then, by first proposing a new grouping genetic algorithm for U-Shaped Assembly Line Balancing problem (UALBP) we provide a detailed cross-examination of the possible evolutionary operators for bin packing problem (BPP) and UALBP.

Keywords: Evolutionary Operators, Genetic Algorithms, Grouping Genetic Algorithm, U-Shaped Assembly Line Balancing Problem, Exploration vs Exploitation

ÖZ

GRUPLAYICI GENETİK ALGORİTMALARDA EVRİMSEL OPERATÖRLERİN ÇEŞİTLİLİK VE SONUÇ KALİTESİNE ETKİSİ

GÖKTUĞ YORGANCILAR
YÜKSEK LİSANS, İnteraktif Bilişim Sistemleri
Tez Yöneticisi: Dr. Öğr. Üyesi Tayfun Küçükyılmaz

Ağustos 2020

Genetik algoritmalar, karmaşık problemleri çözmek için kullanılan evrimsel meta-sezgisel algoritmalardır. Literatürde birçok varyasyon olmasına rağmen, tüm genetik algoritmalardaki ortak nokta bir dizi evrimsel operatörün tekrarlı bir şekilde bir dizi rastgele çözüme uygulanmasıdır. Bu evrimsel operatörler, mümkün olan en iyi çözümü elde etmek için yarı rastgele bir şekilde yeni çözümler üretmeyi amaçlamaktadır. Evrimsel operatörlerin etkinliği büyük ölçüde iki rakip faktöre, yani problem alanının keşfi ve sömürüsüne bağlıdır.

Her ne kadar çeşitli genetik algoritma operatörleri ve bunların modifikasyonları önerilmiş ve literatürde karşılaştırılmış olsa da, farklı operatörlerin bir arada keşif ve sömürü açısından nasıl bir performans sergilediği analiz edilmemiştir. Bu tezde, keşif ve sömürü çözüm kümesinin çeşitliliğine ve uygunluğuna indirgenip genetik algoritmanın çeşitli bileşenleri ile arasındaki etkileşimleri ayrıntılı olarak incelenmiştir. Bu amaçla, kutulama problemi ve hat dengeleme problemleri gibi problemleri çözmek için genetik algoritmaların özel versiyonları olan gruplayıcı

genetik algoritma kullanılmıştır. Daha sonra, önce U Tipi Montaj Hattı Dengeleme Problemi için yeni bir gruplama genetik algoritması önerilerek, kutu paketleme problemi ve U Tipi Montaj Hattı Dengeleme problemi için olası evrimsel operatörlerin ayrıntılı bir çapraz incelemesi sunulmuştur.

Anahtar Kelimeler: Evrimsel Operatörler, Genetik Algoritmalar, Gruplayıcı Genetik Algoritma, U Tipi Montaj Hattı Dengeleme Problemi, Keşif ve Sömürü

TABLE OF CONTENTS

ABSTRACT vi

ÖZ viii

TABLE OF CONTENTS x

LIST OF TABLES xii

LIST OF FIGURES xiii

LIST OF ABBREVIATIONS xv

1 INTRODUCTION 1

2 GENETIC ALGORITHMS 3

2.1 Genetic Encoding 4

2.2 Initial Population Creation 5

2.3 Fitness Calculation 5

2.4 Darwinian Evolution 5

2.4.1 Selection 6

2.4.2 Crossover 6

2.4.3 Mutation 7

2.4.4 Elimination Strategy 7

3 RELATED WORK 8

3.1 Initial Population Generation 8

3.2 Fitness Calculation 9

3.3 Darwinian Evolution 10

3.3.1 Selection 10

3.3.2 Crossover 12

3.3.3 Mutation 12
4 FRAMEWORK 15
4.1 Bin Packing Problem 15
4.2 Single Line Balancing Problem 17
4.3 U-shaped Assembly Line Balancing Problem 19
4.3.1 Feasibility Control Mechanism 20
4.3.2 Repair Method 22
4.3.3 Fitness Function 23
5 EXPERIMENTS AND RESULTS 27
5.1 Experiments on BPP and the Results 27
5.1.1 Experiments on Elimination Strategies 30
5.1.2 Experiments on Child Check Feature 34
5.1.3 Experiments on Selection Operators 37
5.1.4 Experiments on Crossover Operators 42
5.1.5 Experiments on Mutation Operators 45
5.1.6 Results on Different Combinations 46
5.2 Experiments on UALBP and the Results 48
6 CONCLUSION 51
REFERENCES 53

LIST OF TABLES

Table 1 Genetic Algorithm operators used in experiments 17
Table 2 Results of the cost efficient and the most state of the art GGA on Hard28 dataset 28
Table 3 Experiment results on Hard28 dataset 46
Table 4 Experimentations on UALBP 48
Table 5 Results of GGA with random and high-quality initial population on UALBP 49

LIST OF FIGURES

Figure 2.1: Steps of a genetic algorithm run 4
Figure 4.1 Assembly line precedence network 18
Figure 4.2: Single assembly line 18
Figure 4.3: U-shaped assembly line 19
Figure 4.4 Modified GGA for UALBP 20
Figure 4.5 Algorithm of Feasibility Control Mechanism.....21
Figure 4.6 Algorithm of the Repair Method 23
Figure 4.7 Results of the first fitness function with penalty 25
Figure 4.8 Results of the second fitness function with penalty.....26
Figure 5.1: Average fitness / Generation graph for the cost efficient GGA 29
Figure 5.2: Diversity / Generation graph for the cost efficient GGA 29
Figure 5.3: Average fitness / Generation graph for the cost efficient GGA with half random half worst elimination.....31
Figure 5.4 : Diversity / Generation graph for the cost efficient GGA with half random half worst elimination.....31
Figure 5.5 : Average fitness / Generation graph for the cost efficient GGA with random elimination 32
Figure 5.6 Diversity / Generation graph for the cost efficient GGA random elimination 32
Figure 5.7 Average fitness / Generation graph for the cost efficient GGA with best elimination.33
Figure 5.8 Diversity / Generation graph for the cost efficient GGA with best elimination.....33
Figure 5.9 Average fitness / Generation graph for the cost efficient GGA with random elimination and child check.....35
Figure 5.10 Number of Added Children / Generation graph for the cost efficient GGA with random elimination and child check.....36
Figure 5.11 Average fitness / Generation graph for the cost efficient GGA with worst elimination and child check.....36
Figure 5.12 Added Children / Generation graph for the cost efficient GGA with worst elimination and child check.....37

Figure 5.13 Average Fitness / Generation graph for Proportional Selection 38
Figure 5.14 Diversity / Generation graph for Proportional Selection 38
Figure 5.15 Average Fitness / Generation graph for Rank Selection.....39
Figure 5.16 Diversity / Generation graph for Rank Selection.....40
Figure 5.17 Average Fitness / Generation graph for Tournament Selection.....40
Figure 5.18 Diversity / Generation graph for Tournament Selection 41
Figure 5.19 Average Fitness / Generation graph for GR Selection.....41
Figure 5.20 Diversity / Generation graph for GR Selection 42
Figure 5.21 Falkenauer Crossover - Falkenauer Mutation - Proportional Selection graphs.....43
Figure 5.22 Quiroz Crossover - Falkenauer Mutation - Proportional Selection graphs 43
Figure 5.23 Falkenauer Crossover - Falkenauer Mutation - GR Selection graphs 44
Figure 5.24 Quiroz Crossover - Falkenauer Mutation - GR Selection graphs.....44
Figure 5.25 Falkenauer Crossover - Quiroz Mutation - GR Selection graphs.....45
Figure 5.26 Quiroz Crossover - Quiroz Mutation - GR Selection graphs.....46

LIST OF ABBREVIATIONS

GA	Genetic Algorithm
GGA	Grouping Genetic Algorithm
BPP	Bin Packing Problem
UALBP	U-Shaped Assembly Line Balancing Problem

1 INTRODUCTION

A genetic algorithm is an evolutionary algorithm that solves complex problems by starting from a random solution set using an iterative approach. Genetic algorithms depend on Darwinian operators called mutation, crossover, selection and elimination, applied over a set of possible solutions and it aims to find the best result by performing semi-random operations. Genetic algorithms are based on two concepts: exploration and exploitation.

Exploration is about examining as much of the candidate solutions as possible in the search space of the problem. The aim is to widen the search space in order to obtain more individuals by increasing the diversity, so that we can get better candidate solutions that might lead better combinations. More exploration means more diversity however, if exploration rate is too high the effort of trying to find different individuals may take more time than it should, and the algorithm may stop before finding the highest quality result. On the other hand, exploitation aims to use the quality of the individuals in a population to come up with a high-quality result. When exploitation rate is too high, the genetic algorithm might result in converging to local optima. Evolutionary operators in a genetic algorithm such as selection, crossover, mutation and elimination contribute to exploration, exploitation or both. Having a balance between exploration and exploitation means that the genetic algorithm would possibly find better results in the least amount of time. Thus, optimizing a genetic algorithm is all about finding a good balance between exploration and exploitation for a given problem.

In the literature there are various studies on the comparison of genetic algorithm operators in terms of the results quality. Despite it is a well-known fact that exploration and exploitation is the core of a genetic algorithm, there is little work in the literature about how different genetic algorithm operators behave in unison in terms of exploration and exploitation.

The aim of the thesis is twofold: first, we analyze how different genetic algorithm operators in different steps behave together. To this end, we select two problems: bin packing problem and u-shaped assembly balancing problem. We implement various state-of-the-art genetic algorithm

operators and conduct experiments on them. Second, using the knowledge we gained throughout our experiments and by making modifications on the genetic algorithm we propose a new genetic algorithm for solving the U-Shaped Assembly Balancing Problem (UALBP).

The structure of the thesis is as follows. Chapter 2 includes the necessary information to understand genetic algorithms, its brief history, and its steps. Chapter 3 presents the literature on the topic and explains the different genetic algorithm operators that are proposed in the literature. Chapter 4 explains the problems that are used and shows the framework. Chapter 5 presents the experimental results and discussions. Finally, we conclude in Chapter 6.

2 GENETIC ALGORITHMS

The history of the genetic algorithms is based on the desire and the motivation of creating computer programs that have the ability to adapt and learn from the environment. Mitchell (1998) explains that early computer scientists such as Alan Turing, John von Neumann and Norbert Wiener were also interested in psychology and biology. Their vision in biology inspired them to focus on the improvement of the living things in time. In years, their desire resulted in such concepts that we are familiar today called machine learning, neural networks and evolutionary computation.

Holland (1975) proposed genetic algorithm as an evolutionary algorithm. His starting point was not to develop an algorithm that solves a specific problem but rather propose a meta-algorithm that is suitable for any challenging problem where, the computer would improve the starting solution set by using operators such as natural selection and adaptation in biology. He named the individuals in a population as chromosomes and defined the terminology of we are still using today with the concepts like selection, crossover, mutation etc. as they are used in computer science.

In general, a genetic algorithm is an evolutionary, metaheuristic algorithm that solves complex problems by starting from a random solution set using an iterative approach. Genetic algorithms mimic the behavior of the populations in the nature. In theory, in the nature, the individuals that has better genetic code for the circumstances tend to survive longer and create better offsprings whereas others die earlier. This results in having a population that has the majority of individuals with better genetic code in time. Thus, if we have or can generate some solutions for our problem space and can consider those solutions as individuals in the nature, then we can create a population and try to get better generations iteratively.

A run of a genetic algorithm has three main steps: initial population creation, fitness calculation and Darwinian evolution. In each run, after creating an initial population the algorithm iterates between the steps of fitness calculation and Darwinian evolution to create new and better populations. Figure 2.1 show the common execution flow of a genetic algorithm. In the following sub-sections, the steps of the genetic algorithms are explained.

Figure 2.1: Steps of a genetic algorithm run

2.1 Genetic Encoding

Genetic encoding defines how a solution (an individual) is going to be represented in a genetic algorithm. Since it is a design decision, that is made at implementation stage, genetic encoding cannot be considered as a step in the genetic algorithm. How a solution is represented is directly correlated with the performance of a genetic algorithm. In order for a genetic encoding to be considered as accurate, the operators that are used in the algorithm should often produce better solutions and various solutions in the problems search space should be covered (Floreano & Mattiussi, 2008).

2.2 Initial Population Creation

Every genetic algorithm starts with an initial population. Diversity and the size of the initial population has direct effect on the success of a run (Floreano & Mattiussi, 2008). Low diversity and insufficient size on the initial population may result in convergence to local optima whereas a diverse and high-quality initial population ends up with a quick and possibly poor result. The ideal size of an initial population is usually determined by experimentation. However, after an optimal limit, increasing the size is not always the best option. A larger size means performing more operations on more solutions and since the cost and complexity of genetic operators are considerably high, increasing the population size can also increase the cost.

2.3 Fitness Calculation

In order to evaluate how good or bad a solution is, we need an evaluation criterion. Fitness value is the evaluation criterion of a solution that is measured by a predefined fitness function. Fitness function should directly express the aim of the problem and it also needs to be compatible with the selected operators of the genetic algorithm. For instance, some of the selection operators such as proportional selection does not work with negative fitness values. The precision of the fitness value should also be considered to diversify the population.

2.4 Darwinian Evolution

Darwinian evolution encapsulates evolutionary operators in a genetic algorithm. They can be categorized in four groups: selection, crossover, mutation, and elimination operators. Each generation (iteration of a genetic algorithm) starts with the selection of solution couples (parents) that are going to create the new child solutions. Then, crossover operation is applied to the parents to produce new offsprings. After all children are created, mutation operation is performed on them. Mutation constitutes to minor changes in a solution. The importance of the mutation operator in a genetic algorithm is that, it provides a way for the algorithm to avoid local optima. Finally, an elimination strategy is applied to the population in order to keep the number of candidate solutions

in the population constant. In following subsections steps of Darwinian evolution are explained more in detail.

2.4.1 Selection

Selection is the first step of the Darwinian evolution iteration. In order to create a new, conceivably better, solutions we need some parent solutions from our population to be selected. While applying a selection on a population we have two variables to define: the number of parents to be selected and the selection operator that is going to be used. The number of parents to be selected which is also called as *selective pressure*, is important in terms of diversity. Selecting a smaller number of couples could lead to a rapid decrease in diversity and may result in convergence to local optima. However, as we already stated, exploration and exploitation concepts are not only defined by a single operator, so that it may also be the case that a low selection ratio could be a perfect match with other operators that we choose and may produce good results. Second variable is the selection operator that is going to be used. In literature, some of the most popular selection operators are proportional selection, rank selection, and tournament selection. Each selection operator has different effects on exploration and exploitation having its own advantages and disadvantages. More on selection operators will be discussed in Chapter 3 in detail.

2.4.2 Crossover

Crossover is the process of a couple to create new child or children. It is also named as *recombination*. Genetic information encoded in the parents are combined to create a new child in various ways. Created child is expected to have similarities with its parents, thus crossover is important for a genetic algorithm since it ensures inheritance and facilitate exploitation of good genes. Crossover operator is usually selected considering the encoding scheme that is being used. It should also fit the aim of the fitness value represents. Various crossover operators are defined in the literature and explained in more detail in Chapter 3.3.2.

2.4.3 Mutation

Each child that is created at the stage of crossover has a probability to mutate. This probability is called as the mutation probability. If the child solution is lucky enough to have a mutation, some random operations are applied on the child in order to create variation in the population. Mutation is applied to children in order to prevent producing same or similar children, or in other words in order to increase the diversity. Since mutation is usually a set of random changes, it is not easy to predict if the child will become better or worse. What is almost guaranteed is that it will be different so that mutation is an operator that increases the diversity. Thus, it is an operator of exploration.

2.4.4 Elimination Strategy

In genetic algorithms population size is constant. Which means, while creating new child solutions some of the solutions in the population should be eliminated in order to maintain the computation time at each generation. In the literature, there are various strategies such as best solution elimination, worst solution elimination, random solution elimination and half worst-half random solution elimination.

3 RELATED WORK

In this chapter, various proposed operators which are suitable for our problem domain in the literature are explained in detail.

3.1 Initial Population Generation

The first step of the genetic algorithm is to create a population to work with. In the literature, the initial population is generated by randomly shuffling the items and then using a packing heuristic to place the items into the bins (Falkenauer, 1996; Stawowy, 2008; Rohlfshagen & Bullinaria, 2010). Shuffling the list allows us to get different item orders in order to pack each solution and it is important to ensure the diversity.

There are various packing heuristics in the literature to make the decision of in which bin the items are going to be placed. In this work first fit (FF), best fit (BF), worst fit (WF) and first fit-next (FFN) packing heuristics are considered.

FF heuristic places the randomly shuffled items to the first bin they fit. Results obtained using FF heuristic can be considered as closest to random. Hence, in theory, it is expected to result in more diverse populations.

BF heuristic places the randomly shuffled items into the bins that has the maximum fullness after inserting the item among the ones that it can fit. In order to minimize the number of bins in the Bin Packing Problem (BPP), the fulness of the bins should be high and close to each other. This approach aims to obtain bins that have higher fullness values by placing the item to the fullest bin. Hence, in theory, it is expected to result in less diverse populations.

WF heuristic is a variation of BF heuristic that has been used in operating systems successfully. It places randomly shuffled items into the bins that has the minimum fullness after inserting the item

among the ones that it can fit. Comparison of FF, BF and WF shows that they perform similarly in most situations and there is not a significant difference in terms of result quality (Johnson, 1974).

FFN heuristic can be considered as the newest packing heuristic in the literature (Quiroz & Castellanos, 2015). When the bin capacity is 2n, placing more than one item whose weights are greater than or equal to n in a bin is impossible so that, traditional heuristics can yield to solutions that has bins with a single item. In order to solve this problem, FFN heuristic first places the items whose weights are greater than or equal to n to the separate bins to obtain well-filled bins. After that, remaining items are placed into the bins using the FF heuristic. It is also stated that FFN heuristic results in around 2% improvement on the number of bins in the initial population.

3.2 Fitness Calculation

As also mentioned earlier, the fundamental aim of the classical BPP is to find the minimum number bins to pack the given items. In order to evaluate how good or bad a solution is, we need to define a function called the fitness function. Since the aim of the BPP is to minimize the number of bins it seems like a good idea to use the number of bins as fitness function at first. However, this approach can yield to losing important information in the search space about the solution. It can be exemplified as rounding all floating-point numbers between 0 and 1 to 0 or 1 and it is obvious that doing so is superficial.

Falkenauer and Delchambre (1992) has one of the earliest works on BPP fitness function. They stated that similar solutions should have similar but not the equal fitness values and using the number of bins can be a wrong generalization since it is not the atomic piece of meaningful information in a solution. Their starting point is concentrating on the aim of the problem. The aim is simply having a smaller number of bins that packs the given items and in order to have lesser bins we need to use the capacity of each bin better. For instance, we prefer one fully packed bin instead of two half-packed bins. This approach leads to the following fitness function where n is the number of bins, $full_i$ is the fulness of the bin i, C is the bin capacity. The proposed fitness function is presented in Equation 1.

$$f_{bpp} = \sum_{i=1}^{N} \frac{(full_i \ / \ C)^2}{N} \tag{1}$$

3.3 Darwinian Evolution

Darwinian evolution is considered in four steps: selection, crossover, mutation and elimination. Here we present the proposed techniques for each evolutionary operator separately.

3.3.1 Selection

Various selection operators are widely used in the literature. Small changes made in state-of-the-art selection operators according to the needs of a problem can make significant difference in terms of exploitation and exploration. This leads to having a significant amount of studies on the selection operators in the literature. Since it is not possible to analyze all selection operators with their modifications, here we will concentrate only on widely used, state of the art, and the best performing selection operators.

One of the widely used selection operator is **proportional selection**. In proportional selection solutions have the chance to be selected proportionally to their fitness values. It can also be considered as a roulette wheel strategy: Each solution is assigned to an area on the roulette wheel proportional to its fitness value. Then, the wheel is spinned randomly to select a solution.

Ranking selection is another popular selection operator. As we have already explained proportional selection, the downside of the proportional selection is when the fitness values of the solutions in a population is too close to each other. In such cases, proportional selection gives almost equal probability to all of the solutions to be selected. The selection result then become similar to that of a random selection. Ranking selection is proposed to fix this downside (Grefenstette & Baker, 1989; Whitley, 1989). Ranking selection assigns selection chances to the solutions using their rankings in the population. The rank of a solution is its position in the population when the solutions are ordered by their fitness values. Finally, a similar selection procedure with proportional selection is used to select a solution. Ranking selection also allows usage of negative fitness values.

Tournament selection is another widely used selection operator since it is easy to implement, efficient, and easy to adapt to different domains (Goldberg & Miller, 1995). Tournament selection runs tournaments using the solutions fitness values to select each solution. To perform a tournament selection, we first need to decide on the tournament size. When the tournament size is high, the solutions with lower fitness values will have less probability to be selected and it can be used when exploitation is more of a concern. On the other hand, using a smaller tournament size increases the probability of selecting solutions with lower fitness values, so it increases the diversity on selection. Thus, using smaller tournaments can be a good idea when exploration is more of a concern. This shows us that tournament selection can be adapted to various domains and needs easily. After deciding on the tournament size (t) we select t random solutions from the population and select the one with the highest fitness value for crossover. This process is repeated until the desired number of solutions are selected for the crossover phase.

Zhang and Kim (2000) theoretically compared aforementioned selection operators by making a series of experimentations on real life industrial problems and concluded that ranking selection and tournament selection perform better than the proportional selection.

Quiroz and Castellanos (2015) proposed a new selection operator called *controlled selection*. Instead of using an approach that just contributes for exploration or exploitation, controlled selection combines both. Two groups of parents are selected, and they are pairwise matched for crossover. Where n is the number of children to be generated, n/2 number of solutions are selected for groups G and R. First, from the best n solutions, n/2 solutions are selected using a uniform probability for the group G. Then, n/2 number of solutions from the rest of the population are selected for group R similarly, using uniform probability. Finally, group G and R are matched pairwise for crossover. Using the best solutions leads to survival of the best genes which contributes to exploitation whereas using random solutions preserve the diversity and contributes to exploration.

3.3.2 Crossover

Falkenauer and Delchambre (1992) proposed one of the earliest grouping crossover operators named BPCX, in the literature. BPCX is a segment level grouping operator. Which means bins of a solution is considered as a string and a segment of this string is transferred between the parents in crossover. Given two parents p_1 and p_2, the bins are first randomly divided into segments. Then, one of the segments from the p_2 is injected into the end of one of the segments of p_1. This can yield to having bins that have repeating items. Thus, eliminating the bins that have repeating items and reinserting them as the non-repeating free items using FFD packing heuristic is the strategy of BPCX.

Quiroz and Castellanos (2015) proposed a grouping crossover operator to improve Falkenauer and Delchambre's work (Falkenauer & Delchambre, 1992). As it is already mentioned in the previous chapters, in classical genetic algorithms, how close two genes are is a meaningful and important information (Brown & Sumichrast, 2003). However, when a grouping genetic algorithm is used distance between the genes are less important. In BPP the important decision is not the distance between the genes but how well-fitted the bins are. Grouping crossover algorithm focuses on this insight. Given two parents p_1 and p_2, grouping crossover operator creates two child solutions from the crossover of p_2 and p_1, and p_2 and p_1. The bins of the parent solutions are first sorted in descending order with respect to the bins' fulnesses. Then the bins of the parents are compared one by one by one and the fullest bin is added to the child solutions bins followed by the other compared bin. After that, we obtain a solution that has repeating bins and items. While iterating the bins if we encounter an item in the bin that already exists in a previous bin, we omit that bin and add its non-repeating items to the free items list. Finally, we reinsert the free items using an FFD packing heuristic.

3.3.3 Mutation

Falkenauer and Delchambre (1992) proposed a simple yet effective mutation operator. Given a solution, the least full bin and a few randomly selected bins are removed from the solution and the items of the removed bins are reinserted to the solution using FF packing heuristic.

Falkenauer and Delchambre's approach is effective but in order to decide how many bins to eliminate in a solution, we need to perform an experimental study. The adaptive mutation operator which is proposed by Quiroz and Castellanos (2015) solves this issue and also improves it by making minor modifications. In adaptive mutation the number of bins to eliminate is calculated by a series of functions presented in Equations 2-4.

$$n_B = [i \,.\, \varepsilon \,.\, p_e] \tag{2}$$

$$\varepsilon = \frac{2 - (i/m)}{i^{1/k}} \tag{3}$$

$$p_\varepsilon = 1 - Uniform(0, \frac{1}{i^{1/k}}) \tag{4}$$

For a solution, number of bins to eliminate n_b is calculated using number of incomplete bins i using an elimination proportion which is defined by Eq. 3 where, k is the rate of change constant and p_e is the elimination probability which is defined by Eq. 4. Observing Eq. 2 and Eq. 3 we see that elimination proportion is inversely proportional to the number of incomplete bins and the percentage of incomplete bins so that, the number of bins to eliminate will decrease when the solutions get larger.

Another advantage of the adaptive mutation operator is it uses a rearrangement heuristic called *rearrangement by pairs.* After some bins are eliminated, before inserting the free items using a standard packing heuristic like FF heuristic, every combination of item pair is first compared with every combination of item pair in the solution to make a swap that makes the solution better. Let (i,j) be an item pair in bin b, (p,k) be an item pair in free items, w_y is the weight of item y. Following steps are followed for all item pairs:

- If $w_p > w_i + w_j$ and swapping p with i and j does not exceed the capacity of b then p is swapped with i and j

- If $w_k > w_i + w_j$ and swapping k with i and j does not exceed the capacity of b then k is swapped with i and j
- If $w_p + w_k > w_i + w_j$ and swapping p and k with i and j does not exceed the capacity of b then p and k are swapped with i and j

Finally, free items are reinserted using FF packing heuristic. This way, it is aimed to obtain better filled bins which results in better solutions and increases the exploitation. On the other hand, rearrangement by pairs feature increases the complexity of the mutation significantly.

4 FRAMEWORK

In this chapter the problems selected for our experiments and GA approaches are explained. Two problems are selected to analyze using genetic algorithms: bin packing problem and u-shaped assembly line balancing problem.

4.1 Bin Packing Problem

Bin packing problem (BPP) is an NP-hard combinatorial optimization problem (Garey & Johnson, 1979). It is an important problem since it has various real-life applications such as logistics, manufacturing, and engineering (Dokeroglu & Cosar, 2014). In literature, BPP is widely researched since it is the core problem of various industrial optimization problems (Camacho, Terashima-Marin, Ochoa, & Conant-Pablos, 2013; Fleszar & Charalambous, 2011; Fleszar, 2013). BPP has many variations in the literature such as 1D, 2D and 3D variations. In this work 1D bin packing problem is considered where, given a set of n items with weight w the aim of the 1D bin packing problem is to pack all items into minimum number of bins with capacity c.

Kucukyilmaz and Kiziloz (2018) states that BPP with small number of inputs can easily be solved by using brute-force algorithms. However, they also state that for a large scale of input it will not be possible considering the structure of the problem. Thus, some metaheuristic methods such as genetic algorithms, particle swarm optimization, tabu search are used to solve the problem (Buljubai & Vasquez, 2016; Cantu-Paz, 2000; Dokeroglu & Cosar, 2014; Fernández, Gil, Baños, & Montoya, 2013; Holland, 1992; Luque & Alba, 2011).

Falkenauer and Delchambre (1992) is one of the earliest works applying genetic algorithms for the bin packing problem. They first defined the concept of grouping problems. A problem is considered as a grouping problem, when the goal is to pack some items in some group, and if the value of a solution (fitness value in this case) increases when the number of the items in a group also increases but decreases when number of groups increases. Using the definition above it can obviously be seen that BPP is a grouping problem. So that, using this idea they enhanced the classical genetic

algorithm and defined a grouping genetic algorithm which is a specialized version of classical genetic algorithm for grouping problems.

Falkenauer (1996) explained the disadvantages of using classical genetic algorithms in grouping problems. First, classical genetic algorithms create redundancy in the search space. As it is stated in the original paper, classical genetic algorithm encoding works as follows. For a solution encoded as ABA, the encoding denotes that the first and third items belong to group A whereas second item belongs to group B. Using that encoding schema let us consider two different solutions S_1 as ABBAC and S_2 as BCCBA. Classical genetic algorithms consider S_1 and S_2 as different solutions. However, they are actually the same solutions having first and fourth items in group one, second and third items in group two and fifth item in group three. Thus, the encoding of standard genetic algorithms creates additional redundant solutions using different gene representations which enlarges the search space and obviously affects the performance.

Second, classical crossover on classical encoding may create child solutions that has nothing similar with its parents. Since the genes that are together does not represent any similar information in classical encoding, swapping some gene groups in between the parents may create child solutions that have nothing in common with its parents and violates the principles of genetic algorithms.

Considering the disadvantages of classical genetic algorithms, grouping genetic algorithms have two main differences. First, the encoding is enhanced to serve the aim of the grouping problems better. Instead of using an encoding schema that focuses on the item order, using an encoding schema that focuses on the bin order is more beneficial. Let us again consider the encoding ABBAC as it is in the previous example. The bins A, B and C has the following items: A = {1,4}, B = {2,3}, C = {5}. So that, the new encoding becomes {1,4}{2,3}{5} in which close genes have similar information in common, and redundancy is prevented. Secondly, genetic algorithm operators that are more suitable with grouping approach are used. The grouping genetic algorithm operators are explained in more detail in Chapter 3.

In this work, 1D BPP is selected as the first problem. The literature already provides several challenging benchmarks for the problem. Hence, the effect of different evolutionary operators and their interactions can be easily observed.

Table 1 Genetic Algorithm operators used in experiments

Packing Heuristic	Selection	Crossover	Mutation	Elimination
First Fit	Proportional	Falkenauer	Falkenauer	Best Elim.
	Rank	Quiroz	Quiroz	Worst Elim.
	Tournament			Random Elim.
	GR			Best and Worst Elim.

In our experiments, the commonly used operators in the literature are shown in Table 1 and are used on 1D BPP in order to analyze how different genetic algorithm operators in different steps behave together in terms of exploration and exploitation. In the table different evolutionary operators at different steps of a genetic algorithm are represented as separate columns.

4.2 Single Line Balancing Problem

Another problem considered to solve using grouping genetic algorithm is U-Shaped Assembly Line Balancing Problem (UALBP). Before explaining UALBP we should first explain the simpler version of it, single line assembly line balancing problem or line balancing problem (LBP) in short. As BPP, UALBP and LBP are also NP-hard combinatorial optimization problems. An assembly line is a production line that is used in factories that consists of multiple workstations. The line stops for a specific time at each station in order for the operators to perform some operations on the product (Figure 4.2). The aim of the line balancing problem is optimally splitting the operations into the workstations (Falkenauer, 2005). Line balancing problem is a more complex form of bin packing problem where items have precedence relationships in between.

Figure 4.1 Assembly line precedence network

Figure 4.1 shows a sample assembly line precedence network. In the figure, each circle represents a task and the arcs between the tasks represent precedence relationships between tasks. In the classical problem, precedences are always considered pairwise. For items i_1 and i_2 the precedence relationship i_1-i_2 denotes that i_2 can be inserted in a workstation if and only if i_1 is already inserted. In i_1-i_2 relationship i_1 is called as the predecessor of i_2 and i_2 is called as the successor of i_1. These constraints make line balancing problem more complex and time consuming than BPP.

Figure 4.2: Single assembly line

4.3 U-shaped Assembly Line Balancing Problem

A u-shaped assembly line problem is remarkably similar to a single assembly line problem except the shape of the assembly line. In a u-shaped assembly line, the workstations are positioned on a U shape and the operators are placed in between the workstations so that they can perform operations on both side (Figure 4.3) whereas in a single assembly line the workstations are positioned on a straight line and operators are only responsible for a single station (Figure 4.2). Hence, while splitting the operations into the workstations we also need to consider the successor and predecessor relationship among the operations, and it makes the UALBP an even more complicated problem.

After analyzing BPP and UALBP we observed that GGA for BPP can also be used to solve UALBP by making some modifications on top of the GGA for BPP that is discussed in chapter 4.1. Thus, we propose a new GGA for UALBP in this study. Figure 4.4 summarizes basic functionality of using GGA for solving UALBP. Note that the genetic algorithm is modified with the addition of two new operators. Modifications made on top of GGA are considered in three steps and are explained in the next sub-sections: feasibility control, repair method, and fitness function.

Figure 4.3: U-shaped assembly line

Figure 4.4 Modified GGA for UALBP

4.3.1 Feasibility Control Mechanism

The first difference between BPP and UALBP is not every child solution we generate using BPP crossover operators are feasible for UALBP. Hence, we need a control mechanism that guarantees a child solution to be feasible. We need to consider the encoding scheme and precedence network to define a feasible solution.

Our approach focuses on the operators rather that the workstations at encoding stage. For instance, O_1 in Figure 4.3 is responsible for the operations in workstations W_1 and W_8, so that as long as we do not exceed the time limit which is similar to the bin capacity in BPP. In the case of UALBP we

can consider W_1 and W_8 as a single bin. This encoding approach leads using precedence network more effectively and it also fits the principles of grouping genetic algorithm we have previously discussed.

Using the encoding scheme and the precedence network, a solution for UALBP is considered to be feasible if we do not encounter an item that has a predecessor or successor that is not yet inserted while iterating the items one by one as they are placed in the bins. In short, while iterating the items, all items we encounter should be freed up on either side of the precedence network in order for the solution to be considered as feasible.

```
1   for each Bin b in bins do
2         for each Item i in b.items do
3               item_list.add(i)
4         for each Item i in b.items do
5               if !item_list.containsAll(i.predecessors) and
6               !item_list.containsAll(i.successors) then
7                     return false
8   return true
```

Figure 4.5 Algorithm of Feasibility Control Mechanism

Figure 4.5 shows the algorithm of the proposed feasibility control mechanism. First, in line 1, each bin in a solution is iterated. In lines 2-3, all items in the current and previous bins are inserted into an auxiliary data structure. Next, in line 4, for each item in the aforementioned bins the predecessors and the successors are checked. If the predecessors or the successors of an item are not already in one of the bins examined so far, the solution is considered as infeasible (Lines 5-7).

Kim et al. (1996) proposed a similar feasibility control mechanism for single line balancing problem. However, in that work string based classical genetic algorithm is used on single line balancing problem. Therefore, our approach has major differences:

- In this study, a more suitable grouping genetic algorithm approach for the problem is used.
- In this study, the more challenging version of line balancing problems, UALBP, is solved.

- New problem specific operators are proposed for UALBP in order to avoid redundancy.

4.3.2 Repair Method

After defining a feasible solution and the method to identify the infeasible solutions, a repair method should be considered to fix the infeasible solutions. A repair method is also proposed by Kim et al. (1996) for single line balancing problem using classical genetic algorithm. As it is stated in Chapter 4.1.1, Kim's feasibility approach only has predecessor relationship since successors are not important in single line balancing problem. However, by simply adding the successor relationship, we can obtain a repair method that will work suitably for UALBP.

As long as we can decide which items are suitable to insert for the corresponding bin using the precedence network, an infeasible solution can easily be fixed. Let us consider an infeasible solution S_1 with the precedence network in Figure 4.1 and bins as {1,4,7}{2,5,8}{3,6} to explain the proposed repair method. First an empty solution is created to generate the fixed solution. Note that, considering an empty solution items 1,5 and 8 can be inserted in the first bin since they neither have a successor or a predecessor. We call those items "available items". Since the item 1 comes first in the original solution's item order among the available items, item 1 is picked first and inserted to the new solution's bin. Then item 1 is removed from the precedence network and the available items are updated. When a bin exceeds its capacity a new empty bin is created. The insertion process is repeated until all elements are inserted into the bins.

Figure 4.6 shows the algorithm of the proposed repair method. First, in line 7, all items that are free on either side of the precedence network are added into an available item set denoted as *available_set*. In line 12, items of a solution are iterated in bin placement order. If an item is in the *available_set*, then it is added to a list of feasibly placed items denoted as *repaired_list* (Line 14) and the *available_set* is updated by removing the feasibly placed items (Lines 15-19). Additionally, both predecessors and successors of such items are examined and added to the *available_set.* Finally, using the new repaired item order in *repaired_list*, a feasible solution is constructed (Line 21).

```
item_list_from_bins : contains items of the solution in the order they are
placed in the solution
repaired_list : contains the items of the repaired solution (empty)
available_set : contains items that are available to insert using the
precedence network (empty)

for each Item i in item_list_from_bins do
      if repaired_list.containtsAll(i.predecessors) or
      repaired_list.containtsAll(i.successors) then
            available_set.add(i)
while !available_set.isEmpty()
      for each Item i in item_list_from_bins do
            if available_set.contains(i) then
                  repaired_list.add(i)
                  available_set.remove(i)
                  for each Item i in item_list_from_bins do
                        if repaired_list.containtsAll(i.predecessors) or
                        repaired_list.containtsAll(i.successors) then
                              available_set.add(i)
                  break
initializeSolutionUsingNewItemOrder(repaired_list)
```

Figure 4.6 Algorithm of the Repair Method

4.3.3 Fitness Function

In the literature, research on line balancing problem mainly focuses on optimizing three different objectives: minimizing the number of workstations (Type 1), minimizing the cycle time (Type 2) and minimizing the workload variance of the workstations (Type 3) (Hazır, Agi, & Guerin, 2019). In this study, we will focus on minimizing the workload variance of the workstations since in the factories there are usually fixed number of workstations and leaving some of the workstations idle is not beneficial.

At this point, we need to consider another difference between BPP and UALBP. In BPP, our objective is to split the items into minimum number of bins whereas in Type 3 UALBP our objective is to smoothly split the items into predefined number of bins. It might first seem that these two problems are significantly different. However, they are not. In both problems until a predefined number of bins is reached, the objective in both problems is similar, which is to decrease the number of bins. Difference occurs when the desired number of bins is reached. After that, decreasing the number of bins lead to creating infeasible solutions for Type 3 UALBP. Instead, we need to improve the smoothness of the item distribution on bins, which relate to a better fitness for the solution, without changing the number of bins. The first solution comes to mind can be simply ignoring the solutions that has a smaller number of bins than desired. However, ignoring some solutions that we generate could decrease our exploration rate and as discussed in previous chapters, even bad solutions might have valuable genes and losing them by ignoring some solutions can degrade the result quality. For that purpose, we applied two separate fitness functions by making modifications on top of the base fitness function we selected and proposed a secondary fitness function to be used when comparing the value of different solutions.

Let us first explain the base fitness function. As it is mentioned, in Type 3 UALBP the result quality, i.e. the smoothness, increases by minimizing the workload variance of the workstations. This is acquired by minimizing the sum of the squared differences of workstation workloads where C is the cycle time, K is the number of workstations and w_k is the workstation workload as shown in Eq. 5 (Hazır, Agi, & Guerin, 2019). However, this base function when used with a genetic algorithm lacks the ability to focus on the desired number of workstations. Hence, our approaches focus on penalizing the solutions that have a smaller number of workstations.

$$Min \sum_{k=1}^{K} (\mathrm{C} - \mathrm{w_k})^2 \tag{5}$$

As a fitness function we first aimed to penalize the solutions that has less number of bins than desired by multiplying their fitness values with a constant. The motivation here is to penalize

solutions that has less number of bins than the target number of workstations. The penalty constant is defined as D/O where D is the desired number of workstations and O is the number of the workstations in the solution. The first fitness function with penalty is tested on a sample UALBP dataset for 500 generations where desired workstation number is 15. Although theoretically feasible, the technique did not work very well in practice. To understand the reasons, the histogram of the workstation numbers and the average fitness value for each group is logged and presented in Figure 4.7. Results show that, even when multiplied with a penalty constant, the solutions with 14 workstations still have better fitness values. A solution might be increasing the penalty multiplier (15/14 in this case) with respect to the difference between the average fitness of solutions with 14 workstations and solutions with 15 workstations. However, by doing so our fitness function becomes dataset dependent which is problematic in terms of automatization principles.

```
{14=97, 15=3}
14-303.3008
15-1297.0
```

Figure 4.7 Results of the first fitness function with penalty

$$F = \sum_{k=1}^{K} (\mathrm{C} - \mathrm{w_k})^2 + \sum_{n=1}^{N} (\mathrm{C})^2 \tag{6}$$

Our second approach focuses on the empty workstations rather than the full ones. Instead of evaluating the solutions by considering the workstations they actually inhabit, we can evaluate them by considering the workstations they should inhabit. In other words, even if we find solutions with 14 workstations where the desired number of workstations is 15, this solution should still be considered feasible when a workstation is counted as empty. Using this insight, for solutions that have less number of workstations than desired, we calculate the fitness function for non-empty bins as in Eq. 5. Next, for each empty workstation we also add a penalty (P) where P = $(C - w_k)^2$ to the fitness value where $w_k = 0$. In fact, this approach merely factors in empty workstations during fitness calculation. Eq. 6 shows the proposed fitness function where K is the number of

workstations in the solution, N is the number of empty workstations, C is the cycle time, and w_k is the workstation workload.

By doing so we obtained satisfying results as shown in Figure 4.8 for the same dataset. We observe that the fitness value of solutions with 15 workstations are now lower than the ones with 14 workstations. We also see that the majority of the population has 15 workstations which denotes our fitness function to be successful.

```
{16=1, 14=3, 15=96}
16-2295.0
14-1285.0
15-1275.0416666666667
```

Figure 4.8 Results of the second fitness function with penalty

5 EXPERIMENTS AND RESULTS

Various experiments are conducted using the GGA operators that are explained in previous sections on BPP and UALBP. In this section the details of the experiments and the results are analyzed. All aforementioned genetic algorithm operators are implemented in Java 8 and experiments are conducted on a computer having Ryzen 5 3600 CPU and 16gb of ram.

5.1 Experiments on BPP and the Results

In previous chapters various GGA operators are explained and discussed. In this part a top down approach is followed by first sharing the results of the cost efficient (or best performing in terms of code complexity) and state of the art (or the costliest) GGA that can be created using the discussed GGA operators. Then, the reasons of the differences are deeply analyzed in order to get better results. In the following set of experiments Hard28 dataset (Belov, Scheithauer, &Mukhacheva, 2008) is used since it is considered to be the most challenging dataset in the literature.

The cost-efficient approach consists of the crossover and mutation operator that is proposed early by Falkenauer. Proportional, Rank and Tournament selections are all widely used and have similar run complexity, thus proportional selection is randomly selected among all as a starting point. Similarly elimination strategy is again selected randomly. Child check feature that will be discussed later in this chapter is turned off. On the other hand, the state of the art and costly approach that is selected has differences in terms of selection, crossover, and mutation operators. The starting experiments are both conducted with an exceedingly small population size of 100 and for a fairly low number of 100 generations.

Table 2 Results of the cost efficient and the most state of the art GGA on Hard28 dataset

Time (sec)	Selection	Crossover	Mutation	Elimination	Child Check	Pop. Size	Generation	Hard28 Result
3.2	Proportional	Falkenauer	Falkenauer	Worst	Off	100	100	5/28
123.3	GR	Quiroz	Quiroz	Worst	Off	100	100	5/28

Table 2 shows the result of the experiments on the cost efficient and state of the art GGA. As a quick overview, the cost efficient and the state of the art GGA seems to have same results while they have a huge difference in time cost. However, just observing the results would not be enough to discuss more on the algorithms to make them better. It is already discussed that a GGA is a matter of the balance of exploration and the exploitation concepts. Simply, on the exploitation side, the population is expected to get better in time and the child solutions that are generated are explained to be at least as good as their parents. So that, the average population quality and the number of added child solutions are observed to get more insight on exploitation. On the other hand, on the exploration side the diversity of our population shows our search space and it is important in order to generate better solutions. In this work we define the diversity as the number of solutions with different fitness values. In the following experiments, experiments in Table 2 are used as baseline.

Figure 5.1: Average fitness / Generation graph for the cost efficient GGA

Figure 5.2: Diversity / Generation graph for the cost efficient GGA

Let us first concentrate on the cost efficient GGA and observe the flow under the hood. For that purpose, one of the datasets (Hard13) that a result could not be found is selected among Hard28 datasets. Figure 5.1 shows how average fitness changes in generations. We observe a drastic increase in average fitness which means our starting population is rapidly improved by our crossover operator. Then after a point around generation 20 average fitness stops increasing. In an

ideal case, if we could have found a solution for the dataset, the graph would be perfectly acceptable. However, since we could not find a solution, we expect our algorithm to keep improving or searching until it finds a solution. In order to find out the reasons why it stops improving early, we need to observe how diversity changes. Figure 5.2 shows the diversity changes in generations. We observe that around generation 20 the diversity also decreases drastically which means our search space gets smaller and the probability of getting different and better solutions decreases. Thus, in order to make our algorithm to find the result we cannot simply increase the generation limit because as long as we have low diversity, the algorithm will generate similar solutions and will not improve. At this point, the GGA operators that may result in decreasing the diversity could be analyzed.

5.1.1 Experiments on Elimination Strategies

Elimination strategies could be a good starting point since it makes a major change on the population at each iteration. In the initial experiment, the worst elimination operator is used. Eliminating the worst solutions aims to only keep the best solutions in the population at each generation assuming the best solutions will generate better solutions at crossover stage. This insight is only valid as long as the diversity is preserved however when the diversity is low the population will have **same** good solutions, and same good solutions will generate similar solutions decreasing the diversity.

Figure 5.3: Average fitness / Generation graph for the cost efficient GGA with half random half worst elimination

Figure 5.4 : Diversity / Generation graph for the cost efficient GGA with half random half worst elimination

Using half random half worst elimination strategy, we do not observe a change on the average fitness on generations graph (Figure 5.3). On the diversity side, Figure 5.4 shows that instead of a direct drastic decrease, around 20th and 40th generations diversity intends to be preserved. Although the diversity is still not preserved at all and decreased in the end, it shows that randomizing the elimination strategy results in preserving the diversity.

Figure 5.5 : Average fitness / Generation graph for the cost efficient GGA with random elimination

Figure 5.6 Diversity / Generation graph for the cost efficient GGA random elimination

Figure 5.5 and Figure 5.6 show how random elimination performs. Since we do not keep the best solutions at each iteration and eliminate randomly, we observe that the average fitness increases slowly in time with respect to the worst and half random half worst eliminations. On the other hand, we observe a significant difference in the diversity. The diversity increases way more slowly and its lower boundary is also higher. Thus, using random elimination makes the population get better

slowly in time however since it preserves the diversity better, the algorithm can keep searching the search space for more generations. That is why, random elimination is an important elimination strategy which contributes to exploration by preserving the diversity.

Figure 5.7 Average fitness / Generation graph for the cost efficient GGA with best elimination

Figure 5.8 Diversity / Generation graph for the cost efficient GGA with best elimination

By observing the half and worst elimination and the random elimination we observed that the more we randomized the elimination the more diversity we get. For that reason, we used an experimental elimination strategy which is not widely used in the literature: best elimination. Figure 5.7 and Figure 5.8 show how best elimination performs. Figure 5.7 shows that the average fitness increases significantly slower than the other operators which shows that the best elimination strategy is not the best option in practice. On the other hand, Figure 5.8 shows that the diversity is not decreasing. This information gives us an insight about why the diversity decreases while using other elimination strategies. The possible reasons are selection method usually selects the same parents and same parents result in similar children and the crossover operator results in similar child solutions. Thus, we also need to experiment on the crossover and selection operators.

Note that, although we are eliminating the best solutions in the population, the population still gets better at each generation. This shows the importance of the diversity. Even two solutions with lower fitness values can always create a good solution benefiting the exploration.

As on overview, various experiments are conducted using worst elimination, half random half worst elimination, random elimination, and best elimination. The aim of the worst elimination is to improve the exploitation. As long as the initial dataset is poor on fitness or the population is not rapidly increasing it could be preferred. On the other hand, results show that random elimination is the best option in order to improve the exploration. Although the average population gets good enough if the solution still could not be found, in order to keep the diversity high to search effectively random elimination could be preferred.

5.1.2 Experiments on Child Check Feature

Another feature that can be used to keep the diversity high is child check feature. As it is previously explained, the child check feature is partially used in some of the elimination operators in the literature however it is not directly used as a standalone feature. In this study, we examine the child check feature separately from the elimination operator for the first time in the literature. Our expectation is that, the child check feature would preserve the diversity and may result in fruitful analysis.

As it is also discussed in the previous subsection, adding similar child solutions to the population decreases the diversity in time. For that reason, the search space gets smaller and it gets harder to find new and better results. Another approach could be simply not adding the child solutions that have the same fitness to the population. In other words, if the child check feature is turned on, and the fitness value of a child solution already exists in the population it is not inserted to the population. In this way, instead of decreasing the diversity, the number of added children at each iteration decreases which results in keeping the probability of selection and crossover operators creating different children higher.

Figure 5.9 Average fitness / Generation graph for the cost efficient GGA with random elimination and child check

Figure 5.10 Number of Added Children / Generation graph for the cost efficient GGA with random elimination and child check

Figure 5.9 and Figure 5.10 shows the results of the cost efficient GGA with random elimination and child check. We observe that average fitness is almost same with the one without child check feature. On the other hand, with child check feature we expect to preserve the diversity at maximum whereas the added child at each iteration is decreased.

Figure 5.11 Average fitness / Generation graph for the cost efficient GGA with worst elimination and child check

Figure 5.12 Added Children / Generation graph for the cost efficient GGA with worst elimination and child check

Since child check preserves the diversity increasing the exploration, using worst elimination strategy which focuses on the exploration could be a better option. Figure 5.11 shows that by using worst elimination with child check feature we obtain better average fitness values with respect to the random elimination strategy. Figure 5.12 also shows that the number of added children at each generation is also similar. So that, children check performs better with worst elimination having a better balance on exploration and exploitation and could be useful on hard problems by preserving the diversity and increasing the search space.

5.1.3 Experiments on Selection Operators

The experiments conducted so far shows that random elimination results in a more diverse population and child check preserves the diversity while decreasing the number of children added to the population at each turn. In the experiments on the selection operator, on top of the cost efficient GGA defined in Table 2, the random elimination strategy is used considering its superior performance on diversity. Child check feature is also expected to perform better with respect to experiments without child check but since it maximizes the diversity, it may make harder to observe the differences in selection operators, thus, child check feature is ignored in these experiments.

Figure 5.13 Average Fitness / Generation graph for Proportional Selection

Figure 5.14 Diversity / Generation graph for Proportional Selection

Proportional selection has a major difference from the other selection operators. Since the probability of being selected is proportional to the solutions fitness values, after the average fitness of the population gets better (or in other words solutions' fitness values gets similar to each other), the probability of being selected for each solution gets closer to each other. Which means, until the population gets good enough, the proportional selection tends to select good solutions but after the population gets better the chances of the solutions to be selected gets more even and it works like

a random selection increasing the diversity. The diversity graph of the proportional selection (Figure 5.14) also supports this idea. Among all selection operators, proportional selection has the highest diversity ratio. However, when we look at the average fitness graph (Figure 5.13) it can easily be seen that other selection operators perform better since their average fitness values increases earlier and a bit higher. Actually, this is a result of the elimination strategy that is selected for this part of experiments. We already stated that random elimination strategy improves the exploration rather than the exploitation. Since the proportional selection also improves the exploration rate, in total the algorithm lacks exploration using this combination. When the earlier experiment we conducted with worst elimination and proportional selection is checked (Figure 5.1 and Figure 5.2), it can easily be seen that average fitness gets better quickly, and the diversity is also better.

Figure 5.15 Average Fitness / Generation graph for Rank Selection

Figure 5.16 Diversity / Generation graph for Rank Selection

Figure 5.17 Average Fitness / Generation graph for Tournament Selection

Figure 5.18 Diversity / Generation graph for Tournament Selection

Figure 5.19 Average Fitness / Generation graph for GR Selection

Figure 5.20 Diversity / Generation graph for GR Selection

Using the diversity and the average fitness graph it is not easy to deduce much about the rest of the selection operators since their results are similar to each other. It should also be considered that; selection operator is not an operator that directly affects the quality or the diversity of the population. Thus, its effects on exploration and exploitation can be better understood when it is analyzed with other operators. In following sections more about the selection operators will be discussed with combination of different GGA operators.

5.1.4 Experiments on Crossover Operators

As it is already mentioned in Chapter 3, Falkenauer crossover simply exchanges a segment in between the parents whereas in Quiroz crossover bins of the two parents are combined and repeating items are removed and reinserted using a heuristic. So far, the experiments are conducted using Falkenauer crossover operator. In this section Quiroz and Falkenauer crossover operators are compared. Considering the experiment results of section 5.1.3 proportional and gr selection are used.

Figure 5.21 Falkenauer Crossover - Falkenauer Mutation - Proportional Selection graphs

Figure 5.22 Quiroz Crossover - Falkenauer Mutation - Proportional Selection graphs

Figure 5.21 and Figure 5.22 show the results of the experiments using Falkenauer and Quiroz crossover operators with proportional selection, respectively. The results show that Quiroz crossover performs better in terms of the diversity, however its average fitness value decreases after some certain point. In order to understand the reason of this decrease, next, the experiments are conducted using gr selection which increases the exploration.

Figure 5.23 Falkenauer Crossover - Falkenauer Mutation - GR Selection graphs

Figure 5.24 Quiroz Crossover - Falkenauer Mutation - GR Selection graphs

Figure 5.23 and Figure 5.24 shows the results of the experiments using Falkenauer and Quiroz crossover operators with gr selection, respectively. It can easily be seen that, as expected, increasing the exploration by using the gr selection increased the average fitness value on Falkenauer crossover whereas it is diversity is not much affected. On the other hand, the average fitness values on Quiroz crossover also increased and the diversity stayed above the half for more generations. It is also observed that the decreasing behavior of the average fitness (Figure 5.22) is solved by increasing the exploration.

In Falkenauer crossover resulting population reaches a higher average fitness value than Quiroz crossover in a short period of time. However, the diversity of the Falkenauer crossover reaches to minimum around 200th generation whereas the diversity of the Quiroz crossover reaches to minimum around 800th generations. After the diversity of the population gets significantly low and

does not increase anymore, it means that the algorithm halts. After the algorithm halts, major change in the result quality is not expected. Moreover, results also show that gr selection significantly improves the performance of Quiroz crossover. Overall, results show that Falkenauer crossover has a high exploitation rate whereas Quiroz crossover has a higher exploration rate. Thus, we cannot say one better than the other but since the complexity of the Falkenauer crossover is lower, it could be a better starting point.

5.1.5 Experiments on Mutation Operators

Final experimentation criteria is the mutation operator. Figure 5.25 and Figure 5.26 shows the results of the experiments using Quiroz mutation on Falkenauer and Quiroz crossover operators. When compared to Falkenauer mutation (Figure 5.23 and Figure 5.24) the results show that Quiroz mutation has similar diversity but better average fitness. Thus, Quiroz mutation performs significantly better than the Falkenauer mutation having similar exploration but better exploitation rate. However, one should also note that there is a huge complexity difference between Falkenauer and Quiroz mutations where the complexity of Quiroz mutation is higher. Our experiments show that, using the proposed datasets no significant observation on the mutation operators can be made.

Figure 5.25 Falkenauer Crossover - Quiroz Mutation - GR Selection graphs

Figure 5.26 Quiroz Crossover - Quiroz Mutation - GR Selection graphs

5.1.6 Results on Different Combinations

Using the knowledge obtained from the experimentations on previous sections various GGA combinations are created and tried on the Hard28 dataset. Experiments were conducted with combinations that were thought to make a difference considering the results of the previous experimentations. All experiments are conducted with population size of 1000 and generation limit as 2000. Table shows the results of the experiments.

Table 3 Experiment results on Hard28 dataset

Experiment	Selection	Crossover	Mutation	Elimination	Child Check	Hard28
1	prop	falk	falk	worst	off	5
2	prop	falk	falk	random	on	5
3	gr	falk	falk	worst	off	5
4	gr	falk	falk	random	on	5
5	gr	quiroz	quiroz	worst	off	8
6	gr	quiroz	quiroz	worst	on	9
7	gr	quiroz	quiroz	random	off	5
8	gr	quiroz	quiroz	random	on	6
9	gr	falk	quiroz	worst	off	14

10	gr	quiroz	falk	worst	off	5
11	gr	falk	quiroz	random	off	15
12	gr	falk	quiroz	random	on	18
13	prop	falk	quiroz	random	off	7
14	gr	falk	quiroz	worst	on	20

Experiments 1, 2, 3 and 4 are conducted using Falkenauer mutation and Falkenauer crossover operators. Different combinations of selection and elimination strategies are tried, however all experiments resulted in 5/28 in Hard28 dataset. Note that the cost efficient GGA in table 2 with population size of 100 and generation limit of 100 also resulted in 5/28. Therefore, 5/28 could be the limit of Falkenauer mutation and Falkenauer crossover combination.

In Table 2 it is stated that the state of the art (the costliest) GGA which uses Quiroz mutation and Quiroz crossover operators, performed 5/28 on Hard28 datasets with population size of 100 and generation limit of 100. In the experiment 5 the population size and the generation limit are increased and unlike the cost efficient GGA results improved with 8/28. Then in experiments 6, 7 and 8 the effect of child check feature and elimination strategies is observed. Results support our deductions in sections 5.1.1 and 5.1.2: Worst eliminations seem to work better when Quiroz mutation and Quiroz crossover operators are used together and when child check feature is turned on results always get better.

The rest of the experiments (9,10,11,12,13,14) are conducted using Falkenauer crossover and Quiroz mutation operators which performed superior than other crossover and mutation combinations. Our first observation is when child check feature is turned on results always gets better and child check feature works better with worst elimination. This assumption is made in section 5.1.2 and it is proved in all experiments we made in this section. Second observation is gr selection works better with Falkenauer crossover and Quiroz mutation combination. In section 5.1.3 it is stated that proportional selection and gr selection would perform similar however, they have differences in terms of exploration and exploitation. The experiments in Table 3 showed that gr selection performs better and should be the first choice. Finally, when used together Falkenauer crossover and Quiroz mutation operators gave the best results among all experiments.

5.2 Experiments on UALBP and the Results

Using the prior knowledge gained in section 5.1, in this section the results of GGA for UALBP is discussed. In the following set of experiments 25 randomly selected datasets provided by Otto et al. (2013) is used.

Table 4 Experimentations on UALBP

Experiment	Crossover	Elimination	Child Check	Pop Size	Generation	Result (over 25)
1	quiroz	random	on	200	1000	20
2	quiroz	random	off	200	1000	19
3	quiroz	worst	on	200	1000	16
4	quiroz	worst	off	200	1000	15
5	falk	random	on	200	1000	9
6	falk	random	off	200	1000	6
7	falk	worst	on	200	1000	7
8	falk	worst	off	200	1000	4

Unlike BPP in UALBP the aim is not only to find a feasible solution with a specific number of bins (or workstation) but also to find better solutions having the specific number of bins. Therefore, for UALBP finding a feasible solution is not enough. However, in order to be able to compare the operators as it is applied for BPP, in the first set of experiments, the datasets are considered to be solved whenever the desired number of bins are reached in a solution.

UALBP Table 4 shows the results of the experimentations. Results show that unlike the BPP dataset we discussed in section 5.1 GGA performs better using Quiroz crossover operator on UALBP. The reason could be as it is previously mentioned UALBP is a complex form of BPP where items has constraints and precedence relationships. Since UALBP is a more complex problem, total possible number of solutions in the search space is expected to be less than the BPP however, those solutions are also expected to be harder to find. Therefore, using Falkenauer crossover, being simpler in nature than the Quiroz crossover cannot produce diverse children, solving a smaller number of datasets successfully. Results also support some of our deductions we made in previous chapters. When child check feature is turned on, i.e. the diversity is preserved through generations, the results always get better.

In the second part of experiments the best performing experiment (Experiment 1) is selected from Table 4 and ran with a larger population size of 1000 and for 10000 generations. In this part the aim is not only to find solutions with desired number of bins but also to find the best solutions among those. In order to evaluate how good a feasible solution is or in other words in order to understand how close a solution is to the optimum, a smoothness value called "lower bound" is used as the optimum value. Lower bound is calculated as follows where N is the number of items, w_i is the weight of the item i and C is the number of desired workstations.

$$LB = \frac{\left(\sum_{i=0}^{N} w_i\right)^2}{C} \tag{7}$$

Table 5 shows the results of the proposed GGA for UALBP on selected benchmarks. The obtained results are evaluated using the optimum value: lower bound. In Table 5, K is the number workstations, LB is the lower bound, Z is the fitness of the best solution, and gap shows the proximity of the solution to the best-case result.

Table 5 Results of GGA with random and high-quality initial population on UALBP

			GGA with random population		**GGA with high quality population**	
file	K	LB	Z	gap (%)	Z	gap (%)
38	14	13400558	13402483	0,014%	13401003	0,003%
61	54	47002675	47537990	1,139%	47537990	1,139%
77	20	19135549	19139945	0,023%	19136777	0,006%
107	14	12777902	12781607	0,029%	12779607	0,013%
108	14	13890217	13890531	0,002%	13890531	0,002%
114	13	12662224	12663104	0,007%	12662712	0,004%
209	56	45358201	46052647	1,531%	46046587	1,518%
218	56	43761536	44641678	2,011%	44634522	1,995%
244	21	19692996	19702278	0,047%	19695650	0,013%
252	14	13445600	13447570	0,015%	13446164	0,004%
325	25	24591681	24593451	0,007%	24593451	0,007%
332	14	13207544	13208906	0,010%	13208236	0,005%
348	14	12474017	12480643	0,053%	12475063	0,008%
349	13	12030772	12033714	0,024%	12031056	0,002%
361	52	45280358	45712786	0,955%	45712786	0,955%
389	23	21070270	21091976	0,103%	21078568	0,039%
399	23	21538003	21555621	0,082%	21544077	0,028%

400	24	22816951	22826719	0,043%	22821429	0,020%
412	14	13128018	13130031	0,015%	13128837	0,006%
438	57	44124962	45076495	2,156%	45022001	2,033%
452	22	20537161	20561732	0,120%	20543102	0,029%
464	25	22682359	22719553	0,164%	22699367	0,075%
481	15	14228193	14232841	0,033%	14229659	0,010%
496	14	13077712	13081623	0,030%	13079147	0,011%
514	60	41209594	42438555	2,982%	42211075	2,430%

In Table 5 the results of two algorithms are provided: results of proposed GGA with a random initial population and results of proposed GGA with a high-quality initial population. Table 5 shows that by starting with a random initial population, all of the datasets are solved and the proximity of the best solutions to the best-case results are less than 3%.

Moreover, in previous experimentations on BPP we showed the importance of the balance of exploration and exploitation. For this purpose, as a second experiment we started the proposed GGA with a high-quality initial population. Hazır, Agi, and Guerin (2019) proposed a heuristic approach to solve single line balancing problem. Their approach consists of three steps: creating priority-based task lists to generate solutions, applying a probabilistic decision-making procedure to place the items into the solutions, repairing infeasible solutions by placing the missing items. In order to generate a high-quality population, approach of Hazır, Agi, and Guerin (2019) is adapted to UALBP by also adding successor relationship on top of the predecessor relationship. In the final experimentation which can be seen on the last column of Table 5, high quality initial population is used in proposed GGA. In this way, as it can be seen in Table 5 the results of all datasets are also improved significantly by decreasing the proximity of the best solutions to the best-case results to less than 2.5%.

6 CONCLUSION

In this study, the main focus is how different genetic algorithm operators in different steps behave together by reducing the two competing factors of the genetic algorithms, exploration, and exploitation, to diversity and fitness. As the problem domain, two problems are selected: Bin Packing Problem (BPP) and U-Shaped Assembly Line Balancing Problem (UALBP). Since the problem domain consists of grouping problems, grouping genetic algorithm which is a specialized version of genetic algorithms is used.

The genetic algorithm is divided into three main steps: initial population creation, fitness calculation and Darwinian Evolution. Darwinian Evolution consists of selection, crossover, and mutation operators. First, various state of the art evolutionary operators in the literature that are used in grouping genetic algorithms are implemented to create a framework for the experiments. In order to solve UALBP, UALBP is considered as a complex form of BPP in which items have precedence relationships. For this reason, three modifications are made on top of GGA for BPP: feasibility control mechanism is added, a repair method is proposed, and the fitness function is modified. This way, a new grouping genetic algorithm is proposed to solve UALBP.

In the first experimentation phase, starting by the experiment of GGA using the cheapest combination of the operators, the low diversity is claimed to be the reason why the algorithm does not result in success. Therefore, various operators at elimination, selection, crossover, and mutation phases are compared and analyzed using BPP in terms of diversity and fitness. In order to preserve the diversity, a feature called *child check feature,* which is not used in the literature before as a standalone feature is proposed. Then, using the knowledge gained throughout the experimentations, different combinations of the operators in GGA are used to solve Hard28 dataset and the results are discussed.

Second set of experiments are conducted on UALBP. First, best performing operator combination is found by making experimentations on different operator combinations. At this point, the

knowledge gained in the first set of experiments are also used to decide on which operators to use in experiments. Final set of experiments are conducted to compare the results of the proposed GGA for UALBP with the best-case results. Results shows that, proposed GGA solves all 25 datasets with less than 3% proximity to the best-case results. Moreover, instead of a random population when a high-quality initial population is used, results of the proposed GGA also improves significantly.

REFERENCES

Belov, G., Scheithauer, G., & Mukhacheva, E. A. (2008). One-dimensional heuristics adapted for two-dimensional rectangular strip packing. Journal of the Operational Research Society, 59(6), 823-832.

Brown, E. C., & Sumichrast, R. T. (2003). Impact of the replacement heuristic in a grouping genetic algorithm. Computers & Operations Research, 30(11), 1575-1593.

Buljubai, M., & Vasquez, M. (2016). Consistent neighborhood search for one-dimensional bin packing and two-dimensional vector packing. Computers and Operations Research, 76, 12–21.

Cantu-Paz, E. (2000). Efficient and accurate parallel genetic algorithms (Vol. 1). Springer Science & Business Media.

Dokeroglu, T., & Cosar, A. (2014). Optimization of one-dimensional bin packing problem with island parallel grouping genetic algorithms. Computers & Industrial Engineering, 75, 176-186.

Falkenauer, E. (1996). A hybrid grouping genetic algorithm for bin packing. Journal of heuristics, 2(1), 5-30.

Falkenauer, E. (2005, July). Line balancing in the real world. In Proceedings of the international conference on product lifecycle management PLM (Vol. 5, pp. 360-370).

Falkenauer, E., & Delchambre, A. (1992, May). A genetic algorithm for bin packing and line balancing. In ICRA (pp. 1186-1192).

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

Fleszar, K. (2013). Three insertion heuristics and a justification improvement heuristic for two-dimensional bin packing with guillotine cuts. Computers & Operations Research, 40(1), 463-474.

Fleszar, K., & Charalambous, C. (2011). Average-weight-controlled bin-oriented heuristics for the one-dimensional bin-packing problem. European Journal of Operational Research, 210(2), 176-184.

Floreano, D., & Mattiussi, C. (2008). Bio-inspired artificial intelligence: theories, methods, and technologies. MIT press.

Garey, M. R., & Johnson, D. S. (1979). Computers and intractability (Vol. 174). San Francisco: W.H. Freeman.

Grefenstette, J. J., & Baker, J. E. (1989). How genetic algorithms work: a critical look at implicit paralleling. The proceedings of the third. international conference on genetic algorithms.

Hazır, Ö., Agi, M., & Guérin, J. (2019). A Problem Specific Heuristic for Smoothing the Workload in Simple Assembly Lines. IFAC-PapersOnLine, 52(13), 2562-2565.

Holland, J. H. (1992). Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. MIT press.

Johnson, D. S., Demers, A., Ullman, J. D., Garey, M. R., & Graham, R. L. (1974). Worst-case performance bounds for simple one-dimensional packing algorithms. SIAM Journal on computing, 3(4), 299-325.

Kim, Y. K., Kim, Y. J., & Kim, Y. (1996). Genetic algorithms for assembly line balancing with various objectives. Computers & Industrial Engineering, 30(3), 397-409.

Kucukyilmaz, T., & Kiziloz, H. E. (2018). Cooperative parallel grouping genetic algorithm for the one-dimensional bin packing problem. Computers & Industrial Engineering, 125, 157-170.

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

Luque, G., & Alba, E. (2011). Parallel genetic algorithms: theory and real world applications (Vol. 367). Springer.

Miller, B. L., & Goldberg, D. E. (1995). Genetic algorithms, tournament selection, and the effects of noise. Complex systems, 9(3), 193-212.

Mitchell, M. (1998). An introduction to genetic algorithms. MIT press.

Otto, A., Otto, C., & Scholl, A. (2013). Systematic data generation and test design for solution algorithms on the example of SALBPGen for assembly line balancing. European Journal of Operational Research, 228(1), 33-45.

Quiroz-Castellanos, M., Cruz-Reyes, L., Torres-Jimenez, J., Gómez, C., Huacuja, H. J. F., & Alvim, A. C. (2015). A grouping genetic algorithm with controlled gene transmission for the bin packing problem. Computers & Operations Research, 55, 52-64.

Rohlfshagen, P., & Bullinaria, J. A. (2010). Nature inspired genetic algorithms for hard packing problems. Annals of Operations Research, 179(1), 393-419.

Stawowy, A. (2008). Evolutionary based heuristic for bin packing problem. Computers & Industrial Engineering, 55(2), 465-474.

Whitley, L. D. (1989, June). The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trials is best. In Icga (Vol. 89, pp. 116-123).

Zhang, B. T., & Kim, J. J. (2000). Comparison of selection methods for evolutionary optimization. Evolutionary Optimization, 2(1), 55-70.