MARMARA UNIVERSITY
INSTITUTE FOR GRADUATE STUDIES IN PURE AND APPLIED SCIENCES

SOLVING BIN PACKING PROBLEM WITH PROBLEM SPECIFIC OPERATORS USING AN EVOLUTIONARY ALGORITHM

TUGBA ZEYNEP YILDIZ

MASTER THESIS

Department of Computer Engineering

ADVISOR

Asst. Prof. Dr. Betül Demiröz Boz

ISTANBUL, 2019

MARMARA UNIVERSITY
INSTITUTE FOR GRADUATE STUDIES
IN PURE AND APPLIED SCIENCES

SOLVING BIN PACKING PROBLEM WITH PROBLEM SPECIFIC OPERATORS USING AN EVOLUTIONARY ALGORITHM

TUGBA ZEYNEP YILDIZ

(524116009)

MASTER THESIS

Department of Computer Engineering

ADVISOR

Asst. Prof. Dr. Betül Demiröz Boz

ISTANBUL, 2019

MARMARA UNIVERSITY

INSTITUTE FOR GRADUATE STUDIES IN PURE AND APPLIED SCIENCES

TUĞBA ZEYNEP YILDIZ a Master of Science student of Marmara University Institute for Graduate Studies in Pure and Applied Sciences, defended her thesis entitled "**Solving Bin Packing Problem with Problem Specific Operators Using an Evolutionary Algorithm**", on 2.07.2019 and has been found to be satisfactory by the jury members.

Jury Members

Asst. Prof. Dr. Betül DEMİRÖZ BOZ (Advisor)
Marmara University .....

Asst. Prof. Dr. Fatma CORUT ERGİN (Jury Member)
Marmara University.....

Asst. Prof. Dr. Tevfik AYTEKİN (Jury Member)
Bahçeşehir University .....

APPROVAL

Marmara University Institute for Graduate Studies in Pure and Applied Sciences Executive Committee approves that Tuğba Zeynep YILDIZ be granted the degree of Master of Science in Department of Computer Engineering, Computer Engineering Program on 10.07.2019..... (Resolution no: 2019.114-02..).

Director of the Institute
Prof. Dr. Bülent EKİCİ

TABLE OF CONTENTS

PAGE

TABLE OF CONTENTS i
ÖZET ii
ABSTRACT iii
SYMBOLS iv
ABBREVIATIONS v
1-D BPP : One Dimensional Bin Packing Problem v
LIST OF FIGURES vi
LIST OF TABLES vii
1. INTRODUCTION 1
2. RELATED WORK 3
3. PROPOSED WORK 6
3.1. Population Initialization 7
3.2. Individual Representation 7
3.3. Offspring Creation 8
3.4. Pool Based Crossover Operator 9
3.5. Local Search 17
3.6. Placement of The Offspring In the Population 21
4. EXPERIMENTAL STUDY 22
5. CONCLUSION 28
6. FUTURE WORK 29
7. REFERENCES 30
8. APPENDICES 32

ÖZET

Evrimsel Algoritma Kullanarak Probleme Özgü Operatörler ile Kutulama Probleminin Çözümü

Kutu paketleme problemi, literatürdeki en önemli optimizasyon problemlerinden biridir. Problem çok çeşitli gerçek yaşam kullanımına sahiptir. Kutu paketleme problemi NP-Hard sınıfına ait olduğundan, bu problemi farklı alanlarda kendi avantajlarıyla çözmek için önerilen birçok çözüm vardır, ancak çoğu optimum çözüme ulaşamamıştır.

Kutu paketleme probleminde, kutulara yerleştirilmesi gereken sınırlı sayıda obje vardır. Her objenin kendi ağırlığı vardır ve her kutu sınırlı bir kapasiteye sahiptir. Problemin temel amacı, tüm objeleri kutulara koyarken, kullanılan kutu sayısını en aza indirmektir. Bu çalışmada, tek boyutlu kutu paketleme problemini çözmek için yeni bir havuz temelli evrimsel algoritma öneriyoruz. Algoritma, problemin arama alanını arttırmayı amaçlayan havuz tabanlı bir çaprazlama operatörü ve çaprazlama çözümü sonucunda oluşan bireyde mevcut olan az kullanılmış kutuları dikkate alarak çözümün kalitesini iyileştirmeyi amaçlayan yerel bir arama tekniğini kullanır. Önerilen yöntem, kıyaslama problem kümelerinden alınan orta ve zor örneklere uygulandı ve literatürdeki altı algoritma ile karşılaştırıldı. Deneysel sonuçlarımız, önerilen algoritmanın, sağlanan test vakalarının çoğunda bu algoritmalardan önemli ölçüde daha iyi performans sergilediğini göstermektedir.

Temmuz, 2019 **Tuğba Zeynep YILDIZ**

ABSTRACT

Solving Bin Packing Problem with Problem Specific Operators Using an Evolutionary Algorithm

Bin packing problem is one of the most important optimization problems from the literature. The problem has a wide range of real life usage. Since bin packing problem belongs to NP-Hard class, there are many heuristics proposed to solve this problem with their own advantages in different domains, but most of them could not reach the optimum solution.

In bin packing problem, there are finite number of items which must be placed into bins. Each item has their own weight and each bin has a finite capacity. Main objective of the problem is to place all items into the bins in a manner that minimizes the number of bins used. This work proposes a novel pool-based evolutionary algorithm for the solution of one-dimensional bin packing problem. The algorithm exploits a pool-based crossover operator which increases the problem's search space and a local search method which tries to decrease the bin usage of the solution by considering underutilized bins available in the offspring. The proposed method is applied to medium and hard instances taken from benchmark problem sets and is compared with six algorithms from the literature. Our experimental evolution indicates that the proposed algorithm significantly outperforms these algorithms in most of the test cases provided.

July, 2019 **Tuğba Zeynep YILDIZ**

SYMBOLS

I_i : i^{th} item in the item list

c : fixed capacity for each bin

n : number of items

S : Offspring

B_i : i^{th} bin in the individual

W_i : weight of i^{th} item

ABBREVIATIONS

ACS : Adaptive Cuckoo Search

AS : Ant System Algorithm

BPP : Bin Packing Problem

BFD : Best Fit Decreasing

DPBC : Dynamic Pool Based Crossover

EA : Evolutionary Algorithm

FA : Firefly Algorithm

FCO : Firefly Colony Optimization

ILWOA : Improved Lévy-Based Whale Optimization Algorithm

PBC : Pool Based Crossover

PBEA : Pool Based Evolutionary Algorithm

SA : Simulated Annealing

QICS : Quantum Inspired Cuckoo Search

WOA : Whale Optimization Algorithm

1-D BPP : One Dimensional Bin Packing Problem

LIST OF FIGURES

Figure 2. 1 – Overall steps for EA flow 4
Figure 3. 1 – Overall steps for Pool-Based Evolutionary Algorithm 6
Figure 3. 2 – Main Scheme of Pool-Based Evolutionary Algorithm 6
Figure 3. 3 – Population Initialization 7
Figure 3. 4 – Individual Representation 7
Figure 3. 5 – Offspring Creation 8
Figure 3. 6 – Pool-Based Crossover Operation 10
Figure 3. 7 – Items and weights combinations for schemes shown in Figure 3.8, Figure 3.9, Figure 3.10 11
Figure 3. 8 – Shuffle Crossover Example 12
Figure 3. 9 – Keep as is Crossover Example 15
Figure 3. 10 – Sorted Crossover Example 16
Figure 3. 11 – Local Search Operation 18
Figure 3. 12 – Weights of items used in Local Search 19
Figure 3. 13 – Local Search Applied to the underutilized bins of an offspring 20
Figure 3. 14 – Placement of the Offspring 21
Figure 4. 1 – Experimental results of medium class dataset 22
Figure 4. 2 – Experimental results of hard class dataset 23
Figure 4. 3 – Performance of the operators on hard class dataset 26

LIST OF TABLES

Table 4. 1 – Experimental results of medium class dataset 24
Table 4. 2 – Experimental results of medium class dataset 25
Appendix 1-Table 1 – Whole experimental results of medium class dataset 32

1. INTRODUCTION

Bin packing problem (BPP) is one of the most important optimization problems from the literature. Bin packing problem is an NP-Hard optimization problem. It can be applied to many real-life problems including industrial and logistic applications, multiprocessor scheduling and cloud computing. Some of them are loading trucks with weighted capacity constraints, layouts of computer chips, scheduling tasks, creating file backups in media and technology mapping in Field-programmable gate array semiconductor chip design. This list is very large and can be extended extremely. One of the most applicable approach for solving bin packing problem is using evolutionary algorithms.

In bin packing problem there are objects having different volumes, which must be packed into bins. The aim is to fill the finite number of bins with the given objects in a manner that minimizes the number of bins used. The main purpose of the problem is to make efficient use of the space and time.

Given a set of items with different weights and an unlimited number of bins with fixed bin capacity, the objective of the problem is to pack these items to minimum number of bins such that the total weight of the items assigned to a bin does not exceed the capacity of the bin. There are many variations of this problem such as 1D packing, 2D packing, 3D packing, regular or irregular packing, packing by cost, packing by weight and so on.

Bin-packing is an NP-Complete [4] problem and many algorithms have been proposed for the solution of the problem, the best results are obtained from hybrid and heuristic algorithms.

There are also meta heuristics that have been proposed to solve the problem such as ant colony optimization [7], cuckoo search algorithm [8, 12], firefly algorithm [11] and whale optimization algorithm [1]. In this study we apply an evolutionary algorithm to solve bin packing problem.

Evolutionary Algorithm (EA) is a heuristic based approach to solve problems that can not be solved in polynomial time such as NP-hard problems. The underlying idea behind evolutionary algorithm is that pressure on the population environment

causes the natural selection. It is also called as the survival of the fittest, which keeps the best solution in the population by replacing one of the poor solutions. An EA contains four base steps: initialization, selection, genetic operators and termination. For initialization process, population which consists of individuals are built. It is often created randomly, optionally if some prior-knowledge of the task is known, chosen algorithms which are suitable for the optimization problem, can be used to create initial population. Then in selection phase of EA, crossover between individuals is used to increase quality by carrying best genes from one generation to the next. Mutation and recombination are applied by genetic operators and they increase the diversity of the population by changing the properties of the individuals. Process terminates when satisfaction criteria or predefined number of iterations is reached. It provides the best solutions (closer to optimum) by the advantage of survival of the fittest principle.

In this study, we propose an evolutionary algorithm for the solution of 1-D bin packing problem by using many problem specific crossover operators and a local search method. The main contributions of this study are: pool-based crossover operator which targets to increase the solution's diversity; local search method which targets to decrease bin usage; intelligent packing by rearranging items in underutilized bins, therefore increase performance and decrease bin usage. Our experimental study indicates that it outperforms related studies from the literature for medium and hard class instances.

2. RELATED WORK

Bin packing is an NP-Hard problem. It searches for a solution which ensures all n items I = {I_1, I_2,…, I_n} are placed into minimum number of bins. In this study we focus on one dimensional bin packing problem (1-D BPP). Items have different weights w_1 , w_2,…., w_n. and all bins have fixed capacity c. It is assumed that all items have lower weight than fixed bin capacity c. The aim of the algorithm is to place all items into the bins in such a way that minimizes the number of bins used. Some applications for 1-D BP are loading of containers like truck, placing data on multiple disks, job scheduling and virtual machine (VM) placement on physical machines (PM).

There are many heuristics to solve BPP. [18], [19], [20]. There is a hybrid genetic algorithm for solving the 1-D BPP which is applied to the virtual machine placement in cloud [13]. Their hybrid genetic algorithm uses Best Fit Decreasing (BFD) to eliminate infeasible solutions which are caused by their bin-used representation. They applied their algorithm to find most suitable host for given virtual machines with hardware and software requirements.

Tabu search algorithms [17] are also used to solve BPP. A two level tabu search algorithm is proposed to solve 3D-BPP [14]. They presented two level tabu search, where the first level reduces the number of bins used, second level aims to optimize packing of bins.

Another different approach is called BISON [2], which combines two algoritms which are tabu search and branch and bound, to solve one dimensional bin packing problem.

The main reference for our algorithm is called Lévy-based whale optimization algorithm (ILWOA) [1]. No-Free-lunch theorem in optimization says that there is no algorithm to solve all optimization problems [15, 16]. But Whale Optimization Algorithm (WOA) gives good results on combinatorial optimization problems. So they adapt WOA to solve BPP. They proposed a new mutation phase to the WOA for improving the convergence speed. Their experimental studies include the HARD dataset of Scholl [2]. Their results obtained by using Hard instances, outperforms all algorithms compared so far.

Since Bin Packing problem belongs to NP-Hard class, many evolutionary algorithms are proposed for solution. EA is a very commonly used algorithm to solve NP-Hard optimization problems.

Evolutionary algorithm is a heuristic based approach which can be applied to solve problems that can not be solved in polynomial time. In other words, there are many problems that exact algorithms are inadequate to solve, for these cases evolutionary algorithms can be able to find optimum solution. They can find optimal starting point faster than exact algorithms. Since bin packing problem is an NP-Hard problem, EA can be used to solve it.

The concept of EA is very similar to natural selection. EA consists of four main stages; initialization, crossover, mutation and selection. Figure 2.1 shows the overall steps of EA.

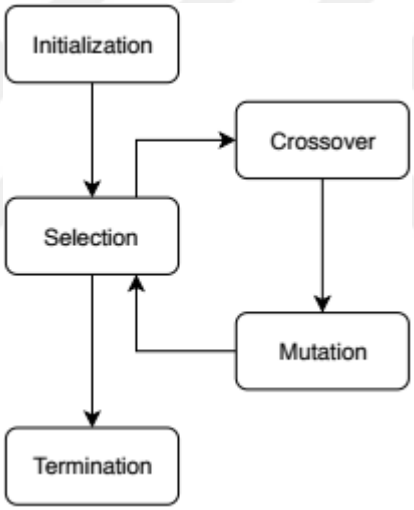

Figure 2. 1 – Overall steps for EA flow

Initialization is the stage where population is created randomly or by using an algorithm. In order to create initial population, a finite number of solutions to the problem are generated. These solutions are called individuals. Individual creation is usually done randomly. But in some cases, we have a prior knowledge about problem specific situations. For these cases, individuals are created by using problem specific information. Each individual created are appended to the initial population. Initial population represents the gene pool. So the more different genes we have, the higher the exploration of the possible solutions.

Selection can be examined in two separate parts. First one is the parent selection for crossover which is done randomly in most of the cases. Two individuals are selected as parent for the crossover operation. Second selection is known as the survival selection. After crossover and mutation, a newly created individual is obtained. We have to decide if the offspring is better than its parents or not and also decide whether offspring will be replaced with one of its parents or not. To select survivor gene, fitness function must be defined. Fitness value measures the quality of solutions in population, defined by considering problem specific parameters and measurements. Evaluating fitness value plays very important role in selection stage in EA. In most of the cases, if offspring has better fitness value, it will be replaced with the parent having the worst fitness value. Otherwise population will not include the newly created offspring, since it provides no improvement.

In EA, we have two genetic operators, crossover and mutation. For crossover after the parents are selected, the characteristics of them will be used to create the offspring. While producing offspring, existing genetic information from both parents are used and protected in one level, on the other hand a different offspring will be created with different genetic sequencing. All genes of the offspring are coming from one of the parents, and the genes coming from different parents provides diversity. There are some classical crossover operators that can be applied to any problem.

Mutation is the phase that changes the solution of the offspring to obtain a different solution. It increases diversity of the solutions and extends the solution space generally. Mutation is applied with some probability in EA, and is not essential to be defined in every EA.

Termination step includes a termination criteria which indicates when to stop the algorithm. It can be predefined as maximum number of iterations, or can be defined as threshold performance, when there is no improvement observed since past predefined steps.

3. PROPOSED WORK

In our approach, the stages we used are; initialization of population, crossover, local search and selection. In our algorithm Pool Based Evolutionary Algorithm, crossover is designed to increase diversity of solutions and extends the search space, local search improves the solution quality by considering the utilization of bins. Figure 3.1. shows the flow of PBEA and Figure 3.2 explains overall flow of PBEA.

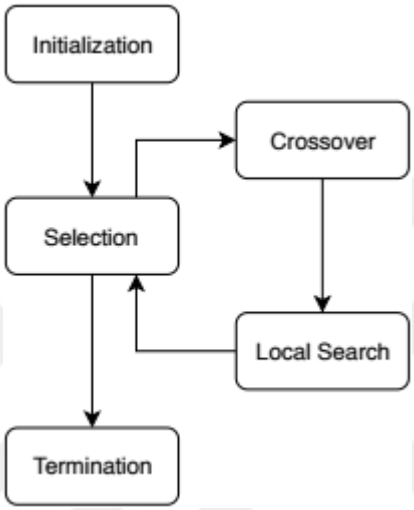

Figure 3. 1 – Overall steps for Pool-Based Evolutionary Algorithm

At the beginning of PBEA, we have items denoted by i, number of iterations represented with *iteration,* population size as *pop_size,* crossover type denoted as *ct* and fixed bin capacity which is *c.* First initial population is created, two parents are selected randomly for predefined number of iterations. According to the specified crossover strategy, crossover operator is applied to generate a new offspring from the selected parents.

Input: Items *i*, number of iteration *iteration*, population size *pop_size*, crossover type *ct,* bin capacity c
Output: Population P
1. P ← initialize_population (*pop_size*, *i, c*)
2. **for** i ← 0 **to** *iteration* **do**
3. Select two parents randomly S_1 and S_2 from P.
4. Find crossover strategy according to ct
5. S_c ← crossover_operation (S_1, S_2)
6. $S_{improved}$ ← local_search (S_c)
7. survival_selection (S_1, S_2, $S_{improved}$)
8. P ← update_pop ()
9. **end for**

Figure 3. 2 – Main Scheme of Pool-Based Evolutionary Algorithm

Generated offspring will be improved by local search. We did not define a probability to apply local search, at each iteration local search will be applied to the offspring. Then depending on the quality of the solution produced by local search, either one of the parents or the offspring is selected to be placed to the population.

3.1. Population Initialization

At the beggining of PBEA, we have to create initial population containing initial solutions which have a number of bins and included items in each bin. List of items are taken as a parameter in the population initialization algorithm and are read from an input file. Fixed capacity for each bin is also taken from the input file. Population size is determined by user input. For most of our test cases, we specify population size denoted by *k* to be 100. Initial population P has predefined number of individuals represented as $P = \{S_1, S_2, ..., S_k\}$.

Figure 3.3 indicates creation process of initial population. There is another function mentioned in Section 3.2 which is called from population initialization, to create each individual in the population.Each individual is a solution for BPP.

Input: Items *i,* size of population k, bin capacity *c*
Output: Initial population *P* which is a list of k number of individuals
Initialization: P ← ⦰
1. **for** t ← 1 **to** *k* **do**
2. Create a new individual S_t: S_t ← create_individual (*i, c*)
3. Put S_t in P: P ← P U S_t
4. **end for**

Figure 3. 3 – Population Initialization

3.2.Individual Representation

Individual can consist of varying number of bins and each bin can include varying number of items as long as they do not exceed the bin's capacity. Individual representation is shown in Figure 3.4.

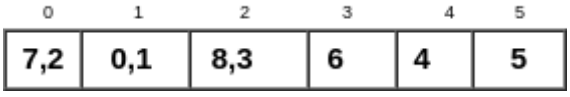

Figure 3. 4 – Individual Representation

Individual is a list of bins and each bin is a list of items. Bins have indices and items have names as numbers in most of the cases.

3.3. Offspring Creation

Offspring creation algorithm is used to create each individual in the population. Figure 3.5 shows the scheme of the algorithm. Every time it is called, the list of items is the same. To increase diversity of individuals in population, items are shuffled first. While iterating over items, algorithm searches backwards to find a suitable bin for the current item. If it is found, the item will be placed into the bin, otherwise a bin will be created and the current item will be placed into newly created bin. At each iteration, algorithm can create at most one bin.

Input: Items *items, Bin capacity c*
Output: Offspring S : S = {B_0 , B_1 ,…, B_n } where B is newly created bin.
1. bins ← ⦰
2.Shuffle items to change the default ordering, and provide randomness to offspring will be created
2. **for** item **in** items **do**
3. bin ← best_fit (*bins, item*) Find bin that is most suitable for current item, if any exists
4. **if** not bin **do**
5. Create new empty bin : bin ← ⦰
6. Put bin in bins: bins ← bin U *bin*
7. **end if**
8. Put *item* in bin: bin ← bin U *item*
9. **end for**
10. Create new individual as S: S ← bins

Figure 3. 5 – Offspring Creation

Offspring creation includes Best Fit (BF), best_fit function, which intends to find the most suitable bin for the current item. Function takes list of bins created in offspring so far and the current item. BF checks whether any space left into the bins which is large enough to put given item. If BF founds more than one bin which satisfy criteria, free space comparison is made between available bins. We iterate over bins and calculate the space will be left when item placed into the each bin. As a result of calculations if we found a bin which item fills all free space of bin and there will be no more free capacity left when item put into, we will select that bin to place item. Else we select the bin which will have minimum area left when item is placed into. This logic based on the idea; we must increase the utilization of bins as much as we can and also we must fill the bins such a way that bin will be fully utilized. If the function couldn't found any bin which current item can be placed into, returns none.

3.4. Pool Based Crossover Operator

In EA crossover is an reproduction operation similar to the biological crossover. More than one parent are selected, and one or more offspring will be created by using parent chromosomes genetic materials. Most of genetic algorithms apply crossover with a predefined probability. There are many applicable types of crossover operation and also there can be many proposed problem specific crossover logic. For our work we prefer to propose different crossover algorithm rather than using classical crossover types. At each iteration two parent selected randomly, and crossover applied with using these parents.

In this study we apply pool based crossover with considering varying number of bins. Two parents are selected and an empty pool is initiated. In every iteration of crossover, items which belong to selected bins from parents, are moved to pool first. These items will be placed to offspring's bins, if it is possible, otherwise they remain in the pool. We defined a constraint while placing the items in pool, that is, at most one new bin can be created in one iteration of crossover.

Starting from the first item in pool, for each item we try to place item to bins of the offspring. If item can be placed existing bin of offspring, it is placed. If item can not be placed any bin of offspring and no bin created in current iteration of crossover, a bin will be created and item will be placed in this bin. If item can not place any bin of offspring, a bin was also created in this iteration, another bin isn't created and item will remain in pool. In next iteration of PBEA new items from newly selected bins from parents will be merged with remaining items in pool. The operation goes on like this. Using pool provide us to benefit of larger search space while creating the offspring.

Input: Two parent chromosome $P_1 = \{B_0^1, B_1^1, \ldots, B_k^1\}$ and $P_2 = \{B_0^2, B_1^2, \ldots, B_k^2\}$, boolean *sort,* boolean *must_shuffle,* boolean *max_first,* boolean *average,* boolean *discharge_pool*
Output*: Offspring S : S = {B0 , B1 ,..., Bk } where B is newly created bin.*

```
1. pool ← ⦰
2. child_bins ← ⦰
3. while not stop_criteria ( P1 , P2 ) do
4.    bin_per_step ← 0
5.    partition1 ← select_bin (P1 , max_first, max_average)
6.    partition2 ← select_bin (P1 , max_first, max_average)
7.    if discharge_pool or pool do
8.       discharge_pool ( pool, child_bins)
9.    end if
10.   items ← Merge items in partition1 and partition2
11.   for each  item in items do
12.      Put item into pool : pool ← pool U item
13.   end for
14.   if sort do
15.      sort_by_weight ( pool )
16.   end if
17.   if must_shuffle do
18.      shuffle ( pool )
19.   end if
20.   placed ← ⦰
21.   for each  item in pool do
22.      Remove item from P1 :  P1 ←  P1 / item
23.      Remove item from P2 :  P2 ←  P2 / item
24.      bin ← best_fit ( child_bins )
25.      if  not bin and not bin_per_step do
26.         Create anew bin for offspring. Bin ← ⦰
27.         bin_per_step ← bin_per_step + 1
28.      end if
29.      else if not bin and bin_per_step do
30.         continue
31.      end else if
32.      Place item into the bin : bin ← bin  U item
33.      if bin not in child_bins do
34.         child_bins ← child_bins U bin
35.         placed ← placed U item
36.      end if
37.   end for
38.   Remove items that in placed list, from pool
39. end while
40. Create offspring S ← child_bins
41. Return S
```

Figure 3. 6 – Pool-Based Crossover Operation

Figure 3.6 shows pool based crossover algorithm. We proposed different versions of Pool Based Crossover. The difference between versions originate from bin selection in parents and pool ordering.

3.3.1. Shuffle Crossover

Pool Based Crossover operator has a parameter named *must_shuffle*. This parameter indicates whether the pool will shuffle or not. If value is true, in each iteration of crossover, pool is shuffled before placing items to the offspring. Shuffling increases the randomness and it prevents to stack the local optima. In shuffle crossover, item distribution in offspring is very diverse. Diversity in offsprings provides larger search space for local search algorithm which improves the solution quality.

Item	0	1	2	3	4	5	6	7	8
Weight	5	7	3	5	12	11	10	11	9

Figure 3. 7 – Items and weights combinations for schemes shown in Figure 3.8, Figure 3.9, Figure 3.10

At each step, one bin from S_1 and one bin from S_2 are selected randomly. These bins and the items in these bins are assigned as selected and will not be used by the crossover operator again. All the items that are present in the selected bins are combined with the items in the pool and are shuffled for increasing the solution's diversity. The items in the pool are then placed into the bins of the offspring one by one. For each item x, there is a back search operation which visits the bins currently available in the offspring one by one. Starting from the first bin B_0 to B_{i-1} if there is any bin B_j which has free space that is equal to or greater than the weight of the item x, than this item is deleted from the pool and is placed to B_j. If no such bin can be found, item x is placed (if there is enough space) to B_i that is generated in the offspring during the current step. Once an item is placed to one of the bins in the offspring, the free space of the corresponding bin is also updated. If item x can not be placed to any bin, then it is kept in the pool to be placed to one of the bins created in the next steps. Figure 3.8 shows the working principle of the crossover operator which is applied to the selected parents. Item weights used in this example are shown in Figure 3.7. The capacity of the bins are 14.

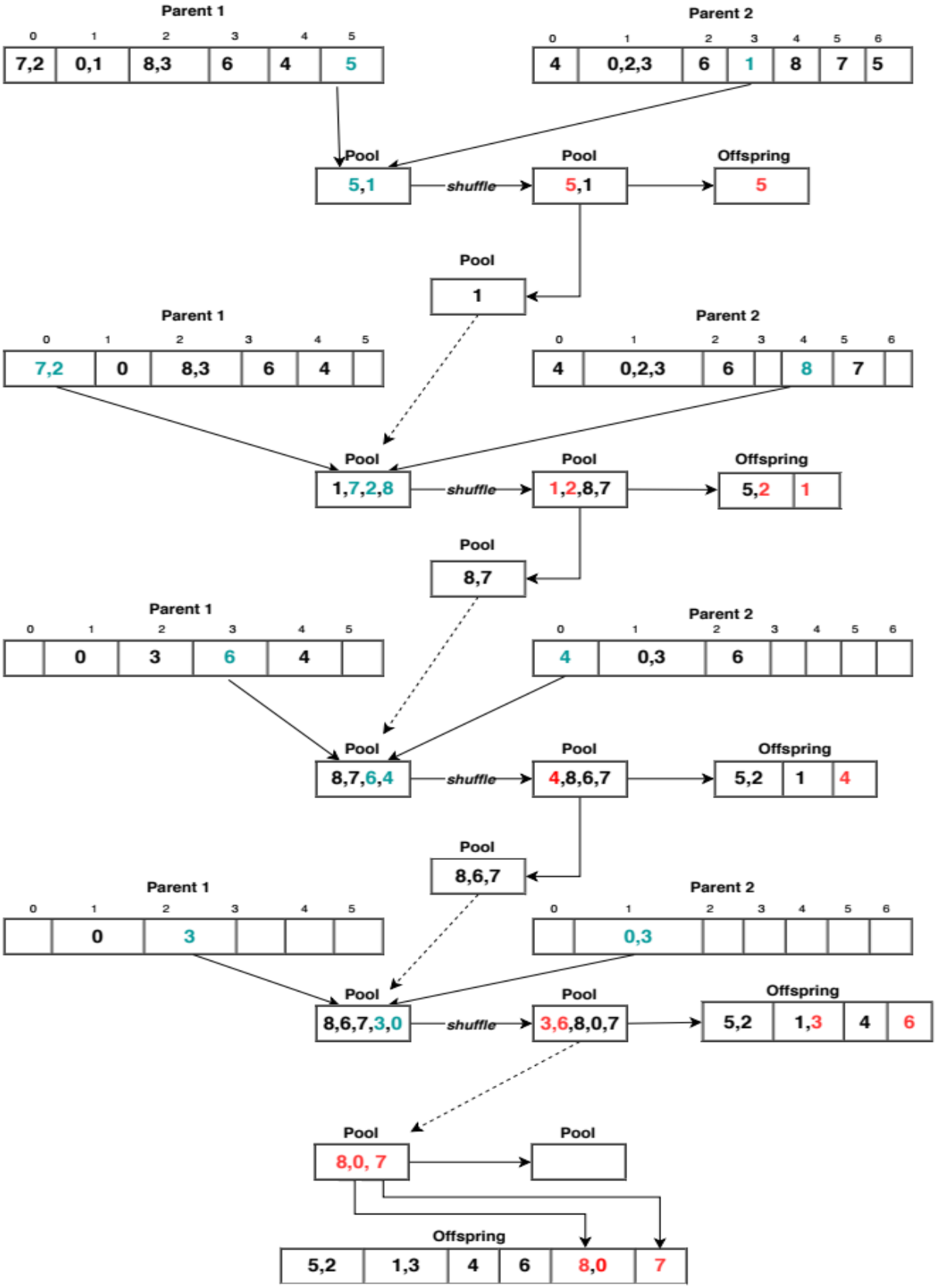

Figure 3. 8 – Shuffle Crossover Example

Two partitions from the parents are selected on a random manner as B_5^1 and B_3^2 and items I_5 and I_1 in these bins are combined in the pool. The items in the pool are shuffled and the first bin having free space of 14 is generated as B_0. Since the weight of the first item I_5 in the pool is less than the free space of B_0, I_5 is deleted from the pool and is placed to B_0 and free space of B_0 is updated to 3. The algorithm tries to put the second item I_1 having a weight of 7 to B_0 , but there is no enough space to store this item in B_0 , so it is kept in the pool. In the second step, B_0^1 and B_4^2 are selected randomly and the items I_2, I_7 and I_8 in these bins are combined with the items in the pool. The pool is shuffled again and B_1 having a capacity of 14 is generated. The first item I_1 is placed into the newly generated bin B_1 of the offspring. The free space of B_1 is updated to 7. The weight of the second item I_2 in the pool is equal to 3, with the back search operation, it is placed to B_0 which makes B_0 fully utilized. The algorithm tries to place items I_7 and I_8 into B_1, but the weights of these items are greater than the free space available in B_1, so they are kept in the pool. In the third step, B_3^1 and B_0^2 are selected randomly and the items I_4 and I_6 in these bins are combined with the items in the pool. After the pool is shuffled, I_4 is placed to the newly generated bin B_2. In the last step, B_2^1 and B_1^2 are selected randomly and all items are thrown into the pool. Again the pool is shuffled and I_3 is placed into B_1 by using the back search operation, whereas I_6 is placed to the newly generated bin B_3. At the end of the this step, all partitions of the parents become empty, so the crossover operator only considers the items in the pool. Since a back search operation has been performed on all items before, a new partition B_4 is generated to place these items. I_0 and I_8 is put into B_4 and for the remaining item I_7 bin B_5 is created. The crossover operator continues until all items in the pool are placed into one of the bins.

3.3.2. Keep as is Crossover

By this manner, items in pool are not shuffled, and they may be nearly sorted, since items in bin nearly sorted. There are two bins at each iteration, one of them comes from first parent and the other one comes from the second parent. There may be another items in pool which can not placed at previous iterations of crossover. All items in pool is ordered by their addition time. First remaining items from previous iteration, than items of first parent's selected bin, and then items of second parent's selected bin. This

order in pool is not changed in any iteration. Figure 3.9 shows the working principle of keep as is crossover.

3.3.3. Sorted Crossover

Pool Based Crossover Operator has another parameter that named *sort*. It indicates whether the pool will be sorted or not. Items in pool is sorted by descending order according to their weights. The logic behind sorting is the same with the logic of algorithm which used to create initial population. Items with higher weight have less change to add any underutilized bin, since the free space of bin may not large enough to place high weighted item. Sorted Crossover decreases randomness, bu it produce better solutions than the others. Since the best movement is preferred in every step, it can be stack in local optima. By applying sorted crossover, offspring includes consecutive bins which have sorted items by weight in ascending order. Figure 3.10 shows the working principle of sorted crossover.

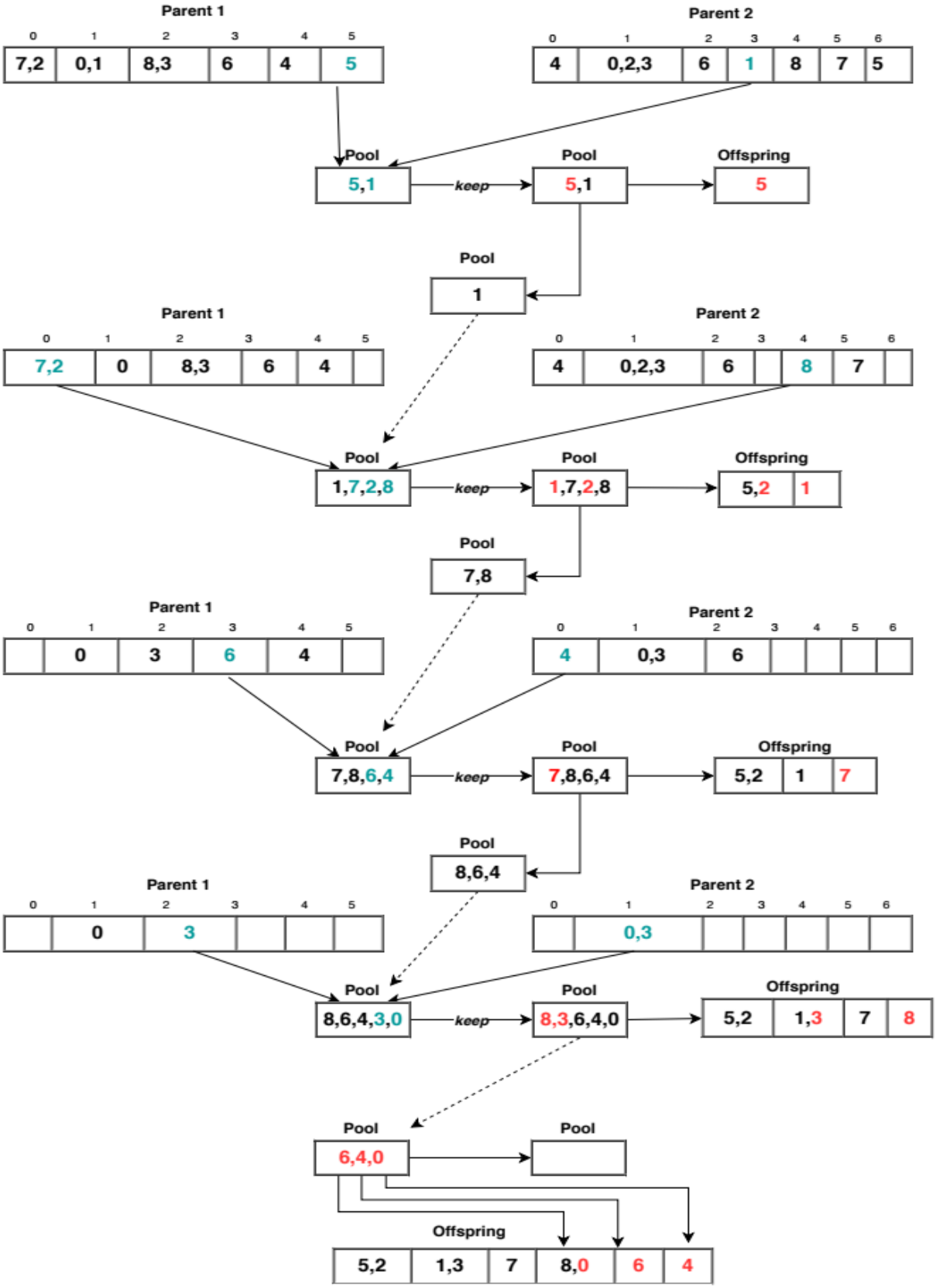

Figure 3. 9 – Keep as is Crossover Example

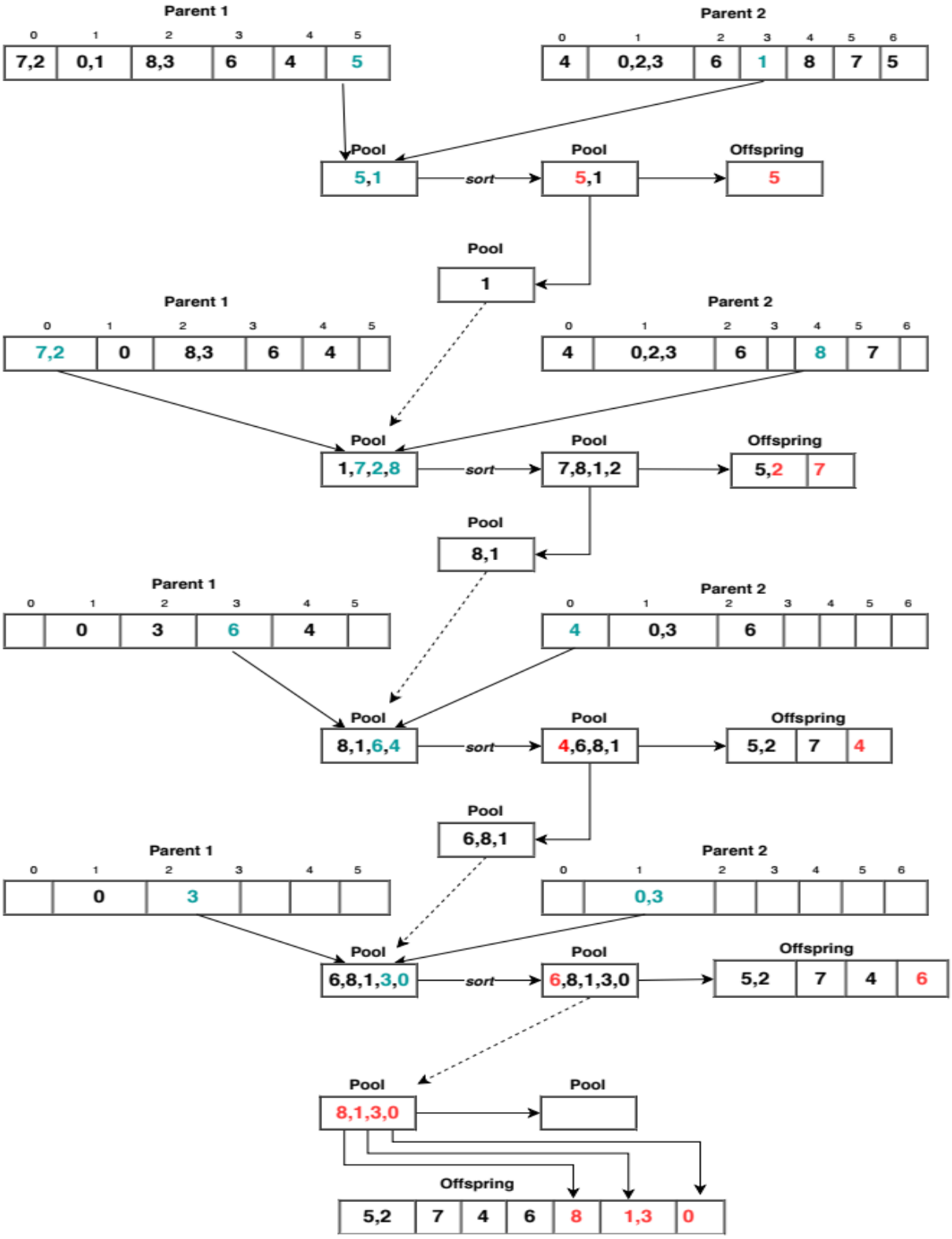

Figure 3. 10 – Sorted Crossover Example

3.3.4. Sorted Max First Crossover

There is another parameter named max_first which relates to bin selection in crossover. If this parameter is true, bin that includes maximum weighted item will be selected from both parents in each iteration of crossover. It is the same logic with the algorithm which used to create initial population. By this method, items that have higher weights, can catch higher change to be placed earlier. SMF Crossover has the same algorithm with Sorted Crossover, but the bin selection priority differs.

3.3.5. Sorted Average Crossover

There is an additional parameter named average which relates to bin selection too. If value is true, bin which have the maximum weights in average will be selected from both parents. For all bins in both parent, average of weighs of items is calculated. Then according to calculated averages, bins will be scored in direct proportion. Highest scored bin will be selected from both parents in each iteration of crossover. SA Crossover is the same algorithm with sorted crossover too, but has an addition consideration in bin selection.

3.3.6. Sorted Discharged Pool Crossover

There is another parameter to discharge the pool. If the pool exceeds or equals the predefined capacity, it will be discharged by creating another bin in offspring, and some items in pool will be placed to the newly created bin in offspring. Discharging in applied on Sorted Crossover.

3.5. Local Search

Instead of searching the whole solution space, searching a sample from solution space is faster way to reach better solution. Solution found may not be the global optima but, it is more practical way to improve existent solutions. Local Serach algorithms based on this idea. Local Search means that the new solution is searched in a neighborhood of the current solution.

In our approach, we proposed a local search algorithm that aims to minimize the number of bins used, by the way it prevents nearly fully utilized bin’s content. First of

all we define a underutilization threshold to improve underutilized bins and prevent utilized ones. This threshold defined parametrically. Accortding to the results of most of test cases, we specified threshold as %10 for HARD [2] dataset instances. This means bins that has lower utilization than predefined threshold is used in local search.

At the end of crossover operation, some bins in the offspring are fully utilized such as B_0 and B_4 as given in Figure 3.8 so there is no need to touch these bins. Local search tries to decrease the number of bins used in the offspring by exchanging information stored in underutilized bins. In order to decrease the number of bins, the total number of items placed into a bin should be increased, so the items that should be considered in the local search phase should have as low weight as possible. Local search phase selects underutilized bin pairs in the offspring and combines the items in these bins to form new bins. When the items from two bins are combined, a more intelligent method than shuffling is used which places the items with the highest weight to the new bin. The items with the lowest weight are thrown to the pool to increase the chance of placing more number of items into the same bin. When all underutilized bins are recombined and the bins in the offspring has been changed, the local search phase ends and the offspring is successfully generated. Local search technique does not always guarantee to decrease the number of bins used in the offspring but it guarantees to increase the utilization of the underutilized bins which increases the chance to find a better solution in the next iterations. Figure 3.11 shows the overall local search operation flow.

Input: An offspring $S_0 = \{B_1\ B_2, .., B_k\}$
Output: An offspring $S_0 = \{B_1\ B_2, .., B_k\}$
Initialization: An empty pool *Pool* ← ⦸
1. *improved* ← S_0
2. Find underutilized bins of offspring S_0 *underutilizeds* ← find_underutilizeds (*improved*)
3. *terminate* ← False
4. **while** not *terminate* **do**
5. Merge 2 underutilized bins to create new partition which is more utilized and add remaining elements to the pool : partition, pool ← partition_reduction (*underutilizeds*)
6. if not underutilizeds do:
7. *terminate* ← True
8. Merge pools to create new partitions partitions, pool ← pool_reduction (*underutilizeds*)
9. **return** *improved*

Figure 3. 11 – Local Search Operation

Figure 3.13 shows an example local search scenario applied to the underutilized

bins of an offspring. In this example, actual data taken from HARD0 instance of hard data sets [2] is used. The weights of the items are divided by 1000 and rounded to increase readability as shown in Figure 3.12. This simple example also shows that the local search technique is able to decrease the total number of bins used in the offspring by 1. The capacity of each bin is 100 and the bins which have utilization less than 90% are considered in this phase.

Item	5	7	19	40	41	47	54	55	57	70	93	96	111	121	124	129	132	134	148	158	159	160	186	190
Weight	35	34	33	32	31	31	30	30	30	29	28	27	26	26	26	25	25	25	24	23	23	23	21	21

Figure 3. 12 – Weights of items used in Local Search

The first underutilized bin pair deleted from the offspring is B_{11} and B_{15}. The items that have the highest weight in these bins are I_5, I_{47} and I_{50} so they are used to generate the first partition P_1. The remaining items I_{124} , I_{129} and I_{186} having lower weights are thrown to the pool. Likewise B_{20} and B_{21} are combined, P_2 is filled with items I_7, I_{41} and I_{96} having highest weight. The items with lower weights are thrown to the pool. P_3 and P_4 are generated by using bin pairs B_{25} and B_{32}, B_{47} and B_{51}. All the underutilized bins have 3 items, so does the newly generated partitions P_1-P_4 . Also the utilization of the partitions are higher as compared to the selected bins. All remaining items that have not been placed to a partition are available in the pool and we know that the weights of these items are lower, so the pool increases the chance to generate partitions that hold more items. Our algorithm initially selects the items with the highest weight from the pool. I_{111}, I_{121} and I_{124} are placed to P_5 and there is still 22 free space in P_5. The next item with the highest weight is I_{129} but it has a weight of 25, so it can not be placed to this partition. Our algorithm applies best fit in this case to see if there is an item in the pool that has a weight equal to the free space available in the partition. If so, it selects that item, removes it from the pool and puts it to the partition. If there is no such item having an equal weight, than it selects the item which has a lower but closer weight. I_{186}'s weight is 21, so it is placed to P_5. Partitions P_6 and P_7 are generated by using the same procedure.

The newly generated partitions P_5 and P_6 is nearly fully utilized and all newly generated partitions have 4 number of items, which decreases the total number of bins used in the offspring by 1. At the end of local search phase, the newly generated partitions are replaced to the underutilized bins in the offspring.

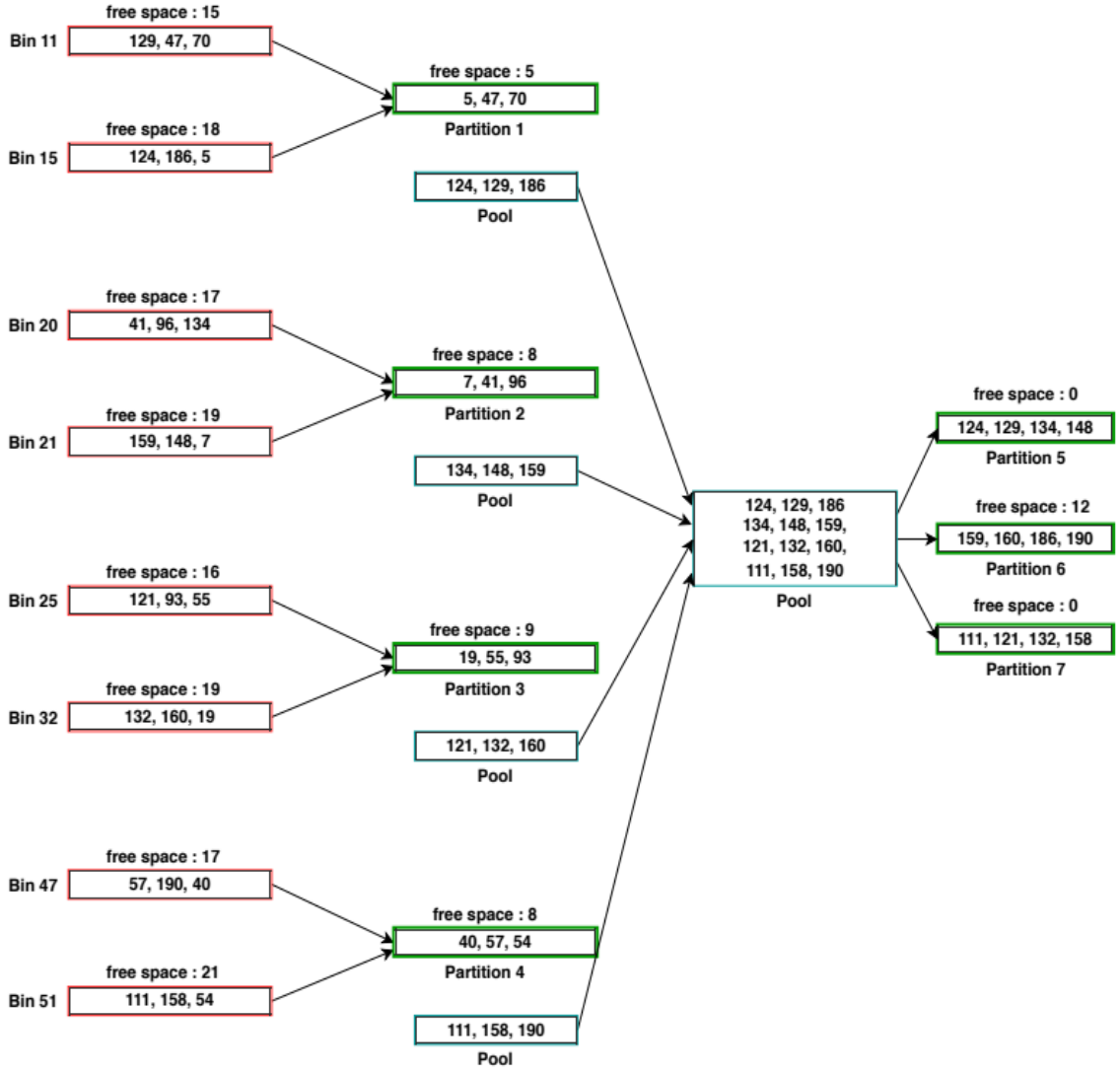

Figure 3. 13 – Local Search Applied to the underutilized bins of an offspring

3.6. Placement of The Offspring In the Population

At the end of each iteration, we decide whether offspring must join population or not. This decision made by comparing fitness value of both parents and offspring. The worst one excluded from population. If the offspring provides improvement relative to its parents, it will be replaced with worse parent. The replacement process is detailed in Figure 3.13.

Input: 1st parent S_1, 2nd parent S_2, the offspring S_0
Output: Update population *Pop*
1. Calculate the fitness values of S_0, S_1 and S_2 according to the function in Section 2.7
2. **if** *fitness*(S_0) > *fitness*(S_1) **then**
3. Replace S_0 with S_1: *Pop* ← *Pop* U S_0 / S_1
4. **else**
5. Replace S_0 with S_2: *Pop* ← *Pop* U S_0 / S_2
6. **end if**

Figure 3. 14 – Placement of the Offspring

4. EXPERIMENTAL STUDY

In this section we provide and run test cases to measure performance of our algorithm. In the first set of tests, the performance of our algorithm compared with Adaptive Cuckoo Search Algorithm (ACS) [12], firefly colony optimization algorithm (FCO) [7], quantum inspired cuckoo search algorithm (QICS) [8], firefly algorithm (FA) [11], ant system algorithm (AS) [6] and improved Lévy-based whale optimization algorithm (ILWOA). In the second set of tests we measured the performance of our algorithm without comparing any other algorithm for all medium dataset taken from benchmark [2], since we have not found any algorithm that run tests all medium class dataset, so far. The last set of tests, we showed the performance of our algorithm using two variations of the pool-based crossover only and crossover combined with local search on hard class instances. For all of the experiments, the results given for each instance is the best result obtained in 10 runs.

Instance	N	Cap	Best	QICS	ACS	FCO	FA	AS	ILWOA	PBEA
N1W1B2R1	50	1000	17	17	17	17	17	18	17	17
N1W1B1R9	50	1000	17	18	17	17	17	17	17	17
N1W1B1R2	50	1000	19	20	19	19	20	20	19	19
N1W1B2R0	50	1000	17	18	17	18	18	18	17	17
N1W1B2R3	50	1000	16	17	17	17	17	17	17	17
N2W1B1R0	100	1000	34	36	34	35	35	37	34	34
N2W1B1R3	100	1000	34	37	35	36	36	36	35	35
N2W1B1R1	100	1000	34	37	35	36	36	36	35	35
N2W1B1R4	100	1000	34	37	34	35	35	35	34	34
N2W3B3R7	100	1000	13	13	13	13	13	13	13	13
N2W4B1R0	100	1000	12	12	12	12	12	12	12	12
N4W2B1R0	500	1000	101	109	105	106	106	107	105	**104**
N4W2B1R3	500	1000	100	108	104	105	105	106	104	**103**
N4W3B3R7	500	1000	74	74	74	74	74	74	74	74
N4W4B1R0	500	1000	56	58	57	57	57	58	57	57
N4W4B1R1	500	1000	56	58	57	58	58	58	57	57

Figure 4. 1 – Experimental results of medium class dataset

The medium class instances contain 50 to 500 items whose weights are between 10-500.

These items should be assigned to the minimum number of bins each having a capacity of 1000. Figure 4.1 shows the performance of the proposed work and the algorithms that are selected from the literature for 16 medium class instances. Our algorithm outperforms all of the given algorithms in 2 instances, and obtains the same performance with ACS and ILWOA in 14 instances. In 9 instances, our algorithm obtained the best known bin value.

Instance	N	Cap	Best	QICS	ACS	FCO	FA	AS	ILWOA	PBEA
HARD0	200	100,000	56	59	58	59	60	59	58	**57**
HARD1	200	100,000	57	60	59	59	60	60	59	**58**
HARD2	200	100,000	56	60	59	59	61	60	59	**58**
HARD3	200	100,000	55	59	58	59	60	59	58	**57**
HARD4	200	100,000	57	60	59	60	61	60	59	59
HARD5	200	100,000	56	59	58	59	60	59	58	58
HARD6	200	100,000	57	59	59	59	61	60	59	**58**
HARD7	200	100,000	55	59	58	58	59	59	57	57
HARD8	200	100,000	57	59	59	59	61	60	59	**58**
HARD9	200	100,000	56	59	59	59	60	59	59	**58**

Figure 4. 2 – Experimental results of hard class dataset

There are 10 large class instances that contain 200 items. The weights of these items are between 20,000 and 35,0000 and the bin capacity is 100,000. Figure 4.2 shows the performance of the proposed work and the algorithms that are selected from the literature for 10 hard class instances. Our algorithm outperforms all the given algorithms in 7 instances and gives the same performance with ACS and ILWOA in 3 instances. Although we are very close to the best known bin value for hard instances, unfortunately our algorithm was not able to produce the best known result.

Table 4. 1 – Experimental results of medium class dataset

DATASET	OPTIMUM	PBEA
N1W1B1R0	18	19
N1W1B1R1	18	19
N1W1B1R2	19	19
N1W1B1R3	18	18
N1W1B1R4	17	17
N1W1B1R5	17	17
N1W1B1R6	17	18
N1W1B1R7	17	18
N1W1B1R8	18	18
N1W1B1R9	17	17
N1W1B2R0	17	18
N1W1B2R1	17	17
N1W1B2R2	17	17
N1W1B2R3	16	17
N1W1B2R4	17	18
N1W1B2R5	17	17
N1W1B2R6	17	17
N1W1B2R7	18	18
N1W1B2R8	16	17
N1W1B2R9	18	18
N1W1B3R0	17	17
N1W1B3R1	17	18
N1W1B3R2	15	15
N1W1B3R3	16	17
N1W1B3R4	19	19
N1W1B3R5	16	16
N1W1B3R6	16	16
N1W1B3R7	18	19
N1W1B3R8	16	16
N1W1B3R9	17	17
N1W2B1R0	11	11
N1W2B1R1	11	11
N1W2B1R2	11	11
N1W2B1R3	11	11
N1W2B1R4	11	11
N1W2B1R5	10	10
N1W2B1R6	11	11
N1W2B1R7	11	11
N1W2B1R8	10	11
N1W2B1R9	11	11
N1W2B2R0	10	10
N1W2B2R1	11	11
N1W2B2R2	10	10
N1W2B2R3	11	11

Table 4. 2 – Experimental results of medium class dataset

DATASET	OPTIMUM	PBEA
N1W2B2R4	10	10
N1W2B2R5	10	11
N1W2B2R6	10	10
N1W2B2R7	10	10
N1W2B2R8	11	11
N1W2B2R9	11	11
N1W2B3R0	10	10
N1W2B3R1	11	11
N1W2B3R2	10	10
N1W2B3R3	11	11
N1W2B3R4	10	10
N1W2B3R5	10	10
N1W2B3R6	11	11
N1W2B3R7	10	10
N1W2B3R8	11	11
N1W2B3R9	10	10
N1W3B1R0	7	8
N1W3B1R1	8	8
N1W3B1R2	7	8
N1W3B1R3	8	8
N1W3B1R4	8	8
N1W3B1R5	8	8
N1W3B1R6	8	8
N1W3B1R7	8	8
N1W3B1R8	7	8
N1W3B1R9	8	8
N1W3B2R0	8	8
N1W3B2R1	8	8
N1W3B2R2	8	8
N1W3B2R3	7	7
N1W3B2R4	7	7
N1W3B2R5	7	7
N1W3B2R6	8	8
N1W3B2R7	8	8
N1W3B2R8	8	8
N1W3B2R9	8	8
N1W3B3R0	7	7
N1W3B3R1	8	8
N1W3B3R2	7	7
N1W3B3R3	8	8
N1W3B3R4	8	8
N1W3B3R5	7	7
N1W3B3R6	7	8
N1W3B3R7	8	8

In Table 4.1, Table 4.2 results obtained from medium class dataset [2] with running our PBEA and optimum values can be reached are shown. There are 480 instances in medium class, and according to our test results we can reach optimum solution in 261 cases. For the rest 219 cases, we are not able to find optimum solutions, even though we are very close. Whole experiment results presented in Appendix 1-Table 1.

	Crossover Only		Crossover + LS	
Dataset	Sorted	Shuffle	Sorted	Shuffle
HARD0	59	59	58	57
HARD1	60	60	59	58
HARD2	60	60	59	58
HARD3	59	59	58	57
HARD4	60	60	60	59
HARD5	59	59	59	58
HARD6	59	60	59	58
HARD7	58	58	58	57
HARD8	60	60	59	58
HARD9	59	60	59	58

Figure 4. 3 – Performance of the operators on hard class dataset

In the last set of tests, the performance of the operators proposed in our algorithm is evaluated. We have conducted tests on 10 hard class instances as given in Figure 4.3. There are two versions of the pool-based crossover operator which vary in the selection of the items in the pool to be placed to the bins. First crossover operator sorts all items in the pool from highest to lowest weight, whereas second operator shuffles the items in the pool as described in Section 3.3. The first crossover type uses a sorted list, so it first tries to place highest weight items to the bins and then tries the other items. Generally, it outperforms the second crossover type because placing heavier items to the bins first is a smarter decision since there will always be much space for lighter items as compared to heavier items. This sorting operation decreases the search space so when combined

with local search, it shows worse performance as compared to shuffling. The pool-based crossover operator using shuffle may not be the best choice when considered alone, but with a combination of the local search operator, it gives the best results.

5. CONCLUSION

In this study, we propose a novel pool-based evolutionary algorithm that solves the one-dimensional bin packing problem. We have designed the pool-based crossover operator to increase the diversity and local search method which decreases the bin usage of the solution.

We also proposed six different versions of pool based crossover operator to make performance comparison between them. If we only consider about crossovers without local search, experimental results shows the best pool based crossover type is sorted crossover. But we clearly notice that, the solutions generated by sorted crossover converges at some point and, due to the sort operation, items are listed consecutively in bins. It is unwanted situation for the further improvement of offspring. Since local search applied only underutilized bins and to get the best performance we need to increase diversity of items included by bins. So we test the performance of all crossover types with using local search too. The results certainly shows local search with shuffle crossover operator has the best performance since shuffle crossover produces more varied solutions, and provides a larger search space for the local search. Then Local search can make the expected improvement on the offspring. At the other hand sorted croosover types generate less varied more ordered solutions, as a result of this, local search owns narrower search space. We clearly observed the best performance with using shuffle crossover with local search. So the rest of experimental study is complete by using shuffle crossover and local search pair.

The experimental study shows that the proposed work has better perfromance than six algorithms from the literature when the total bin number in use is considered.

6. FUTURE WORK

Our algorithm produce solutions so close to the optimum. We planned to propose different local search methodology, to obtain optimum solutions, especially on Hard dataset [2]. Newly designed local search algorithm will focus on grouping items according to their weights while replacing them. There will be two weight class, heavy and light. Items that has middle weight according to the average weight of item population can included by any class, heavy or light, it will not matter for us. By considering the weight classes we generate a pool and try to increase utilization of light class, by replacing items with heavy class.

We think that there is no need left to generate another crossover type to improve our solution. Instead of this, we prefer to improve our local search algorithm or propose another local search which will be applied according to some probablity. When we evaluate offsprings generated by local search, it is shown that there are still minor improvements to reach the optimum solution, by doing little movements.

7. REFERENCES

[1] Mohamed Abdel-Basset, Gunasekaran Manogaran, Laila Abdel-Fatah, and Seyedali Mirjalili. 2018. An improved nature inspired meta-heuristic algorithm for 1-D bin packing problems. Personal and Ubiquitous Computing 22, 5-6 (2018), 1117–1132.

[2] Robert Klein Armin Scholl and Christian Jürgens. 1997. BISON: a fast hybrid procedure for exactly solving the one-dimensional bin packing problem. Computers and Operations Research 24, 7 (1997), 627–645.

[3] Emanuel Falkenauer. 1996. A hybrid grouping genetic algorithm for bin packing. Journal of Heuristics 2, 1 (1996), 5–30.

[4] Micheal Garey and David Johnson. 1990. Computers and Intractability; A Guide to the Theory of NP-Completeness. W.H. Freeman Co., New York, NY.

[5] Mitsuo Gen and Runwei Cheng. 2000. Genetic Algorithms and Engineering Optimization. John Wiley Sons, New York, NY.

[6] Scott Kirkpatrick, Daniel Gelatt, and Mario P. Vecchi. 1983. Optimization by Simulated Annealing. Science 220, 4598 (1983), 671–680.

[7] Abdesslem Layeb and Zeyneb Benayad. 2014. A novel firefly algorithm based ant colony optimization for solving combinatorial optimization problems. International Journal of Computer Science and Applications 11, 2 (2014), 19–37.

[8] Abdesslem Layeb and Seriel Rayene Boussalia. 2012. A Novel Quantum Inspired Cuckoo Search Algorithm for Bin Packing Problem. Information Technology and Computer Science 5 (2012), 58–67.

[9] Marcela Quiroz-Castellanos, Laura Cruz-Reyes, Jose Torres-Jimenez, Claudia Gómez S., Héctor J. Fraire Huacuja, and Adriana C.F. Alvim. 2015. A grouping genetic algorithm with controlled gene transmission for the bin packing problem. Computers and Operations Research 55 (2015), 52–64.

[10] Gizem Sungu and Betul Boz. 2015. An evolutionary algorithm for weighted graph coloring problem. In Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation. ACM, 1233–1236.

[11] Xin-She Yang. 2009. Firefly Algorithm, Lévy Flights and Global Optimization.

Research and Development in Intelligent Systems 26 (2009), 209–218.

[12] Zakaria Zendaoui and Abdesslem Layeb. 2016. Adaptive Cuckoo Search Algorithm for the Bin Packing Problem. Modelling and Implementation of Complex Systems (2016), 107–120.

[13] Mohamed Amine Kaaouache and Sadok Bouamama 2015. Solving Bin Packing Problem with a Hybrid Genetic Algorithm for VM Placement in Cloud. 19th International Conferance on Knowledge Based and Intelligent Information and Engineering Systems (2015), 1061-1069

[14] Teodor Gabriel Crainic, Guido Perboli b, Roberto Tadei 2007. TS 2 PACK: A two-level tabu search for the three-dimensional bin packing problem. European Journal of Operational Research 195 (2009) 744–760

[15] Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. IEEE Trans Evol Comput 1:67–82

[16] Ho YC, Pepyne DL (2002) Simple explanation of the no free lunch theorem of optimization. Cybern Syst Anal 38 (2):292–298

[17] Andrea Lodi, Silvano Martello, Daniele Vigo (2002) Heuristic Algorithms for three-dimensional bin packing problem. European Journal of Operational Research (2002) 410-420

[18] Teodor Gabriel Crainic, Guido Perboli, and Roberto Tadei 2008. Extreme point-based heuristics for three-
dimensional bin packing. Informs Journal on computing 20, 3 (2008), 368–384.

[19] Kevin Sim, Emma Hart 2013. Generating Single and Multiple Cooperative Heuristics for the One Dimensional Bin Packing Problem Using a Single Node Genetic Programming Island Model. In Proceedings of the Genetic and Evolutionary Computation Conference, GECCO'13, 978-1-4503-1963-8/13/07, 2013

[20] P.Ross, S.Schulenburg, J.G. Marín-Blázquez, and E. Hart. Hyper-heuristics: Learning to combine simple heuristics in bin-packing problems. In Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '02, pages 942–948, 2002

8. APPENDICES

Appendix 1-Table 1 – Whole experimental results of medium class dataset

DATASET	OPTIMUM	PBEA
N1W1B1R0	18	19
N1W1B1R1	18	19
N1W1B1R2	19	19
N1W1B1R3	18	18
N1W1B1R4	17	17
N1W1B1R5	17	17
N1W1B1R6	17	18
N1W1B1R7	17	18
N1W1B1R8	18	18
N1W1B1R9	17	17
N1W1B2R0	17	18
N1W1B2R1	17	17
N1W1B2R2	17	17
N1W1B2R3	16	17
N1W1B2R4	17	18
N1W1B2R5	17	17
N1W1B2R6	17	17
N1W1B2R7	18	18
N1W1B2R8	16	17
N1W1B2R9	18	18
N1W1B3R0	17	17
N1W1B3R1	17	18
N1W1B3R2	15	15
N1W1B3R3	16	17
N1W1B3R4	19	19
N1W1B3R5	16	16
N1W1B3R6	16	16
N1W1B3R7	18	19
N1W1B3R8	16	16
N1W1B3R9	17	17
N1W2B1R0	11	11
N1W2B1R1	11	11
N1W2B1R2	11	11
N1W2B1R3	11	11
N1W2B1R4	11	11
N1W2B1R5	10	10
N1W2B1R6	11	11
N1W2B1R7	11	11
N1W2B1R8	10	11

N1W2B1R9	11	11
N1W2B2R0	10	10
N1W2B2R1	11	11
N1W2B2R2	10	10
N1W2B2R3	11	11
N1W2B2R4	10	10
N1W2B2R5	10	11
N1W2B2R6	10	10
N1W2B2R7	10	10
N1W2B2R8	11	11
N1W2B2R9	11	11
N1W2B3R0	10	10
N1W2B3R1	11	11
N1W2B3R2	10	10
N1W2B3R3	11	11
N1W2B3R4	10	10
N1W2B3R5	10	10
N1W2B3R6	11	11
N1W2B3R7	10	10
N1W2B3R8	11	11
N1W2B3R9	10	10
N1W3B1R0	7	8
N1W3B1R1	8	8
N1W3B1R2	7	8
N1W3B1R3	8	8
N1W3B1R4	8	8
N1W3B1R5	8	8
N1W3B1R6	8	8
N1W3B1R7	8	8
N1W3B1R8	7	8
N1W3B1R9	8	8
N1W3B2R0	8	8
N1W3B2R1	8	8
N1W3B2R2	8	8
N1W3B2R3	7	7
N1W3B2R4	7	7
N1W3B2R5	7	7
N1W3B2R6	8	8
N1W3B2R7	8	8
N1W3B2R8	8	8
N1W3B2R9	8	8
N1W3B3R0	7	7
N1W3B3R1	8	8
N1W3B3R2	7	7
N1W3B3R3	8	8
N1W3B3R4	8	8

N1W3B3R5	7	7
N1W3B3R6	7	8
N1W3B3R7	8	8
N1W3B3R8	7	7
N1W3B3R9	7	7
N1W4B1R0	6	6
N1W4B1R1	6	6
N1W4B1R2	6	6
N1W4B1R3	6	6
N1W4B1R4	6	6
N1W4B1R5	6	6
N1W4B1R6	6	6
N1W4B1R7	6	6
N1W4B1R8	6	6
N1W4B1R9	6	6
N1W4B2R0	6	6
N1W4B2R1	6	6
N1W4B2R2	6	6
N1W4B2R3	6	6
N1W4B2R4	6	6
N1W4B2R5	6	6
N1W4B2R6	6	6
N1W4B2R7	6	6
N1W4B2R8	6	6
N1W4B2R9	6	6
N1W4B3R0	6	6
N1W4B3R1	6	6
N1W4B3R2	7	7
N1W4B3R3	6	6
N1W4B3R4	6	6
N1W4B3R5	7	7
N1W4B3R6	6	6
N1W4B3R7	6	6
N1W4B3R8	6	6
N1W4B3R9	6	6
N2W1B1R0	34	34
N2W1B1R1	34	35
N2W1B1R2	34	35
N2W1B1R3	34	36
N2W1B1R4	34	35
N2W1B1R5	34	34
N2W1B1R6	34	36
N2W1B1R7	34	35
N2W1B1R8	34	35
N2W1B1R9	34	35
N2W1B2R0	36	37

N2W1B2R1	33	34
N2W1B2R2	35	36
N2W1B2R3	35	36
N2W1B2R4	33	34
N2W1B2R5	34	36
N2W1B2R6	35	36
N2W1B2R7	33	34
N2W1B2R8	34	36
N2W1B2R9	33	34
N2W1B3R0	35	36
N2W1B3R1	32	33
N2W1B3R2	35	36
N2W1B3R3	33	33
N2W1B3R4	34	34
N2W1B3R5	34	34
N2W1B3R6	31	32
N2W1B3R7	30	31
N2W1B3R8	34	34
N2W1B3R9	29	30
N2W2B1R0	20	21
N2W2B1R1	20	21
N2W2B1R2	21	21
N2W2B1R3	21	21
N2W2B1R4	21	21
N2W2B1R5	21	22
N2W2B1R6	21	21
N2W2B1R7	21	21
N2W2B1R8	21	21
N2W2B1R9	20	21
N2W2B2R0	21	22
N2W2B2R1	21	21
N2W2B2R2	21	21
N2W2B2R3	22	22
N2W2B2R4	22	23
N2W2B2R5	20	21
N2W2B2R6	21	21
N2W2B2R7	20	20
N2W2B2R8	21	21
N2W2B2R9	20	21
N2W2B3R0	21	21
N2W2B3R1	20	20
N2W2B3R2	21	21
N2W2B3R3	19	19
N2W2B3R4	21	21
N2W2B3R5	21	21
N2W2B3R6	20	20

N2W2B3R7	21	22
N2W2B3R8	20	20
N2W2B3R9	20	20
N2W3B1R0	15	15
N2W3B1R1	15	15
N2W3B1R2	15	15
N2W3B1R3	14	15
N2W3B1R4	15	15
N2W3B1R5	15	15
N2W3B1R6	15	15
N2W3B1R7	14	15
N2W3B1R8	15	15
N2W3B1R9	15	15
N2W3B2R0	15	15
N2W3B2R1	14	15
N2W3B2R2	15	15
N2W3B2R3	15	15
N2W3B2R4	15	15
N2W3B2R5	14	14
N2W3B2R6	14	14
N2W3B2R7	15	15
N2W3B2R8	14	14
N2W3B2R9	15	15
N2W3B3R0	15	15
N2W3B3R1	13	13
N2W3B3R2	14	14
N2W3B3R3	14	14
N2W3B3R4	15	15
N2W3B3R5	15	15
N2W3B3R6	15	15
N2W3B3R7	13	13
N2W3B3R8	15	15
N2W3B3R9	14	14
N2W4B1R0	12	12
N2W4B1R1	12	12
N2W4B1R2	12	12
N2W4B1R3	12	12
N2W4B1R4	12	12
N2W4B1R5	12	12
N2W4B1R6	11	12
N2W4B1R7	12	12
N2W4B1R8	11	12
N2W4B1R9	11	12
N2W4B2R0	11	11
N2W4B2R1	11	11
N2W4B2R2	11	12

N2W4B2R3	11	11
N2W4B2R4	12	12
N2W4B2R5	12	12
N2W4B2R6	12	12
N2W4B2R7	11	11
N2W4B2R8	12	12
N2W4B2R9	11	11
N2W4B3R0	12	12
N2W4B3R1	12	12
N2W4B3R2	12	12
N2W4B3R3	11	12
N2W4B3R4	12	12
N2W4B3R5	10	10
N2W4B3R6	11	11
N2W4B3R7	11	11
N2W4B3R8	11	11
N2W4B3R9	12	12
N3W1B1R0	67	70
N3W1B1R1	67	71
N3W1B1R2	67	69
N3W1B1R3	67	70
N3W1B1R4	67	70
N3W1B1R5	67	70
N3W1B1R6	68	71
N3W1B1R7	67	68
N3W1B1R8	67	70
N3W1B1R9	67	69
N3W1B2R0	67	69
N3W1B2R1	68	70
N3W1B2R2	65	67
N3W1B2R3	65	67
N3W1B2R4	68	71
N3W1B2R5	65	68
N3W1B2R6	66	69
N3W1B2R7	66	69
N3W1B2R8	66	69
N3W1B2R9	66	69
N3W1B3R0	68	70
N3W1B3R1	66	68
N3W1B3R2	67	68
N3W1B3R3	66	67
N3W1B3R4	68	69
N3W1B3R5	65	66
N3W1B3R6	66	67
N3W1B3R7	61	62
N3W1B3R8	65	66

N3W1B3R9	70	71
N3W2B1R0	41	42
N3W2B1R1	41	42
N3W2B1R2	41	42
N3W2B1R3	41	42
N3W2B1R4	40	41
N3W2B1R5	41	42
N3W2B1R6	41	42
N3W2B1R7	41	42
N3W2B1R8	40	42
N3W2B1R9	41	42
N3W2B2R0	40	41
N3W2B2R1	40	41
N3W2B2R2	39	40
N3W2B2R3	39	40
N3W2B2R4	41	42
N3W2B2R5	40	41
N3W2B2R6	40	41
N3W2B2R7	40	41
N3W2B2R8	42	43
N3W2B2R9	40	41
N3W2B3R0	42	43
N3W2B3R1	40	40
N3W2B3R2	39	39
N3W2B3R3	39	40
N3W2B3R4	42	43
N3W2B3R5	40	41
N3W2B3R6	40	41
N3W2B3R7	42	42
N3W2B3R8	39	39
N3W2B3R9	39	39
N3W3B1R0	28	29
N3W3B1R1	28	29
N3W3B1R2	29	29
N3W3B1R3	29	29
N3W3B1R4	29	29
N3W3B1R5	29	29
N3W3B1R6	29	29
N3W3B1R7	29	30
N3W3B1R8	29	29
N3W3B1R9	29	29
N3W3B2R0	29	29
N3W3B2R1	29	29
N3W3B2R2	28	29
N3W3B2R3	29	29
N3W3B2R4	30	30

N3W3B2R5	29	29
N3W3B2R6	29	29
N3W3B2R7	28	29
N3W3B2R8	28	28
N3W3B2R9	29	29
N3W3B3R0	27	27
N3W3B3R1	27	27
N3W3B3R2	29	30
N3W3B3R3	29	29
N3W3B3R4	29	29
N3W3B3R5	28	28
N3W3B3R6	29	29
N3W3B3R7	28	29
N3W3B3R8	29	30
N3W3B3R9	29	29
N3W4B1R0	23	23
N3W4B1R1	23	23
N3W4B1R2	23	23
N3W4B1R3	23	23
N3W4B1R4	23	23
N3W4B1R5	23	23
N3W4B1R6	23	23
N3W4B1R7	23	23
N3W4B1R8	23	23
N3W4B1R9	23	23
N3W4B2R0	23	23
N3W4B2R1	22	23
N3W4B2R2	22	22
N3W4B2R3	22	22
N3W4B2R4	22	23
N3W4B2R5	23	24
N3W4B2R6	22	23
N3W4B2R7	22	22
N3W4B2R8	23	23
N3W4B2R9	22	22
N3W4B3R0	24	24
N3W4B3R1	22	22
N3W4B3R2	23	23
N3W4B3R3	24	24
N3W4B3R4	23	23
N3W4B3R5	23	23
N3W4B3R6	23	23
N3W4B3R7	21	21
N3W4B3R8	23	23
N3W4B3R9	23	23
N4W1B1R0	167	173

N4W1B1R1	167	173
N4W1B1R2	167	175
N4W1B1R3	167	175
N4W1B1R4	167	174
N4W1B1R5	167	173
N4W1B1R6	167	177
N4W1B1R7	167	174
N4W1B1R8	167	174
N4W1B1R9	168	178
N4W1B2R0	164	171
N4W1B2R1	170	177
N4W1B2R2	164	171
N4W1B2R3	166	173
N4W1B2R4	165	172
N4W1B2R5	161	168
N4W1B2R6	168	176
N4W1B2R7	168	175
N4W1B2R8	167	174
N4W1B2R9	169	176
N4W1B3R0	166	168
N4W1B3R1	171	174
N4W1B3R2	172	175
N4W1B3R3	170	172
N4W1B3R4	158	160
N4W1B3R5	162	164
N4W1B3R6	169	172
N4W1B3R7	163	165
N4W1B3R8	170	173
N4W1B3R9	162	164
N4W2B1R0	101	105
N4W2B1R1	101	104
N4W2B1R2	101	104
N4W2B1R3	100	104
N4W2B1R4	101	104
N4W2B1R5	103	106
N4W2B1R6	102	105
N4W2B1R7	101	104
N4W2B1R8	101	104
N4W2B1R9	101	104
N4W2B2R0	101	104
N4W2B2R1	100	103
N4W2B2R2	102	106
N4W2B2R3	102	105
N4W2B2R4	100	103
N4W2B2R5	102	105
N4W2B2R6	103	106

N4W2B2R7	101	104
N4W2B2R8	100	103
N4W2B2R9	102	105
N4W2B3R0	101	102
N4W2B3R1	101	102
N4W2B3R2	100	100
N4W2B3R3	100	101
N4W2B3R4	100	101
N4W2B3R5	101	102
N4W2B3R6	99	99
N4W2B3R7	101	101
N4W2B3R8	99	99
N4W2B3R9	102	103
N4W3B1R0	71	73
N4W3B1R1	71	73
N4W3B1R2	71	73
N4W3B1R3	71	72
N4W3B1R4	71	73
N4W3B1R5	71	73
N4W3B1R6	71	73
N4W3B1R7	71	72
N4W3B1R8	71	72
N4W3B1R9	71	73
N4W3B2R0	71	73
N4W3B2R1	71	72
N4W3B2R2	71	72
N4W3B2R3	71	72
N4W3B2R4	73	74
N4W3B2R5	72	73
N4W3B2R6	71	72
N4W3B2R7	70	72
N4W3B2R8	72	73
N4W3B2R9	70	71
N4W3B3R0	72	73
N4W3B3R1	71	71
N4W3B3R2	72	72
N4W3B3R3	71	71
N4W3B3R4	73	73
N4W3B3R5	73	74
N4W3B3R6	73	73
N4W3B3R7	74	74
N4W3B3R8	72	73
N4W3B3R9	71	71
N4W4B1R0	56	57
N4W4B1R1	56	57
N4W4B1R2	56	58

N4W4B1R3	56	57
N4W4B1R4	56	57
N4W4B1R5	56	58
N4W4B1R6	56	58
N4W4B1R7	56	57
N4W4B1R8	56	57
N4W4B1R9	55	57
N4W4B2R0	55	56
N4W4B2R1	57	57
N4W4B2R2	57	57
N4W4B2R3	57	57
N4W4B2R4	56	57
N4W4B2R5	55	56
N4W4B2R6	56	56
N4W4B2R7	57	57
N4W4B2R8	55	56
N4W4B2R9	56	56
N4W4B3R0	55	55
N4W4B3R1	54	54
N4W4B3R2	57	57
N4W4B3R3	56	56
N4W4B3R4	55	55
N4W4B3R5	55	55
N4W4B3R6	58	58
N4W4B3R7	57	57
N4W4B3R8	57	58
N4W4B3R9	56	56

RESUME

PERSONAL INFORMATIONS

Tuğba Zeynep Yıldız
tugba.zeynep.yildiz@gmail.com
04.09.1993
0531 851 60 94
Beykoz/İstanbul

EDUCATION

M.S.: Marmara University Faculty of Engineering Göztepe/İstanbul - Computer Science Engineering(English) 2016-2019 (Still continues) 3.27

B.S.: Marmara University Faculty of Engineering Göztepe/İstanbul - Computer Science Engineering(English) 2011-2016 June 3.00

SKILLS

Proggramming Languages / Frameworks	**Storage/Analysing/Data Processing**	**CI / Deployment Tools**	**Agile Tools**
Python - Django Rest - Celery	SQL – CQL -Postgres – Cassandra	Jenkins	Attlassian Jira – Confluence
Java – Spring – Spring Boot – Maven	Storm	Docker	Bitbucket (Git)
	Elasticsearch		
	Fluentd		
	Rabbitmq – Kafka		
	Redis - Hazelcast		

EXPERIENCE

SEKOM Yazılım (1 January 2016, still continues as Project Lead & Senior R&D Software Engineer) :

→ **Lott (Data Science) :** Forecasting, anomaly detection and leakage detection for Lott Water Application. Machine Learning algorithms applied for short-term predictions and regression algorithms applied for feature engineering, implemented in python with using sckitlearn, pandas frameworks. Paper (Short term water demand forecasting using regional data) submitted and accepted by SIU conference, will be published into IEEE Xplore Digital Library.

→ **Lott (IoT) :** Big data Processing and Analysing on IoT technologies. Polling meter-device data from sensors, data are received via api endpoint of different network servers. Process data and extract meaningfull fields like usage, trend analysis of results and reporting.

→ **IPAM :** IP and planning distribution management system for TT, Turkcell and Vodafone.

→ **Peta Scale Search :** Searching terabytes of network data, analyzing and reporting included.

→ **DPI Console / Manager :** Managing components for DPI devices for multiple vendors , reporting and analyzing whole DPI networks and also analyzing logs.

→ **Sky Frameworks :** Reusable java and python modules.

EGEMSOFT (7 September 2015, 31 December 2015, Software Specialist, Support) :

→ **Sebastiyan :** Bot implemented in python and Django for in-house usage. Implemented bot can respond to messages and catch standup reports submitted via Hipchat Server. Attlassian Jıra integrated bot to do worklog and effort estimation analysis and reporting.

→ **Dpı Console/ Manager :** As referenced before

→ **Esefpy Frameworks :** Reusable java and python modules.

Intern, Alcatel Lucent Teletaş (August 2015-September 2015, 1 month): Worked with Osos Team of R&D Department. Developed web service application in java in order to read and evaluate datas comes from meters, provided crud operations on them with a user interface icefaces and JSF.Java EE facilities used on whole project.

Intern, Egem Bilgi İletişim (EGEMSOFT) (June 2015-July 2015, 1.5 months):Blog site developed with using Django rest framework for functionallİty of backend side, AngullarJs used to implement front end side as an user interface.Recent helpfully technologies grunt and bower is applied.

LECTURES – COURSES & PROJECTS RELATED

Marmara University:

Genetic Algorithms (Thesis Subject): "A Novel Pool-based Evolutionary Algorithm for the Bin Packing" submitted GECCO 2019 (Prague). If it is accepted it will be published ACM.

Combinatorial Optimization, Data Mining Social Networks, Natural Language Processing, Software Engineering, Project Management

LANGUAGES

Turkish: Native English: Advance

CERTIFICATES

ICON-CS (International Conference on Computer Science and Syber Security) (October 2018)

- Data Mining
- IoT Applications
- Machine Learning
- Big Data

Genç Liderler Akademi Programı (Aralık 2013) ,KALDER

The program includes these topics:

- Leadership
- Process Management
- Strategic Management
- Sales, marketing and customer satisfaction

Alcatel-Lucent Telecommunication Networks Knowledge Junior Certificate(August 2015), Alcatel Lucent Teletaş

Consists of tranings below:

- IP/MPLS Protocols
- Cloud Computing
- Software Development on Telecommunication Sector
- Dsl fixed Acces and the Internet
- **Exam** to get certificate