A Hybrid Genetic Algorithm for Packing in 3D with Deepest Bottom Left with Fill Method

Korhan Karabulut and Mustafa Murat İnceoğlu

Ege University, Department of Computer Engineering,
Bornova, Izmir, Turkey
{korhan,inceoglu}@bornova.ege.edu.tr

Abstract. Three dimensional bin packing problems arise in industrial applications like container ship loading, pallet loading, plane cargo management and warehouse management, etc. In this paper, a hybrid genetic algorithm (GA) is used for regular 3D strip packing. The Genetic Algorithm is hybridized with the presented Deepest Bottom Left with Fill (DBLF) method. Several heuristic methods have also been used for comparison with the hybrid GA.

1 Introduction

The bin packing problem can be defined as finding an arrangement for n objects, each with some weight, into a minimum number of larger containing objects (or bins), such that the total weight of the item(s) in the bin does not exceed the capacity of the bin. The bin packing problems arise in forms of one, two and three dimensions. Though a lot of research has gone into 1D and 2D, 3D bin packing is relatively less researched [1]. The bin packing problem is NP-hard and is based on the partition problem [2].

In classical three dimensional bin packing problem, there is a set of n rectangular shaped boxes, each with integer values of width w_i, height h_i and depth d_i. The problem is to pack boxes into a minimum number of bin(s) with width W, height H and depth D. The main objective in 3D bin packing is to maximize the number of items placed in the bin(s), hence, minimize the unused or wasted volume. One of the versions of this problem is the knapsack loading problem where the boxes have an associated profit, and the problem is to maximize the profit of boxes packed into a single bin [3]. Another version of the bin packing problem is strip packing, where the objective is to pack all boxes into a single bin with fixed width and height but with a variable depth; the problem is to minimize this depth.

An algorithm designed for 3D bin packing must consider the constraints and automate the packing process so that the algorithm could be applied to different versions of the problem easily. The algorithm must ensure that there is no overlap between boxes. Intelligent methods can be applied to find good packing patterns. Genetic algorithms are such customizable and adaptive general-purpose intelligent methods that allow exploration of a large search space when compared to specialized placement rules.

Heuristics for 3D bin packing problem can be divided into construction and local search heuristics. Construction heuristics add one box at a time to an existing partial packing until all boxes are placed. The boxes are pre-sorted by one of their dimensions and added using a particular strategy, e.g., variants of first fit or best fit methods [3]. The designed hybrid genetic algorithm maintains a population of solutions and iteratively tries to find better solutions.

Our work presented in this paper is about three dimensional strip packing problem that arise in industrial applications like container ship loading, pallet loading, plane cargo management and warehouse management, etc.

Several researchers have studied on different versions of the three-dimensional bin packing problem. Corcoran and Wainwright [4] have proposed a hybrid genetic algorithm based on a 2D method called level technique that is extended to 3D for the classical bin packing problem. The bin is divided into slices along the height and levels along the length. Next fit and first fit heuristics are used to place the items. They reported a maximum of %76.9 bin utilization. Ilkka Ikonen et al. [5] present a genetic algorithm for packing of three-dimensional objects with complex shapes.

Silvano Martello, et al. [6] presents an exact branch-and-bound algorithm that is able to solve instances with up to 90 items optimally within reasonable time. Günther R. Raidl [7] presents a weight-coded genetic algorithm for the multiple container packing problem with similarities to the bin packing problem. He uses two decoding heuristics for filling multiple bins. Oluf Faroe, et al. [3] uses guided local search method.

In this paper, a 3D packing method called Deepest Bottom Left with Fill (DBLF) is presented. DBLF is used within a hybrid genetic algorithm and several heuristics are also used for comparison.

2 Genetic Algorithms

Genetic Algorithms (GAs) are adaptive methods that may be used to solve search and optimization problems [8]. The principles of the evolution of biological organisms are the main idea behind genetic algorithms. Genetic algorithms are first introduced by Holland in 1975 [9] and are studied by many researchers.

Genetic algorithms are based on the simulation of the natural evolution. The building blocks or elements in genetic algorithms are "individuals" which are candidate solutions for a problem. Each of the individuals is called either a "genotype" [9] or a "chromosome" [10]. They are usually coded as binary bit strings.

A genetic algorithm is an iterative process and is composed of several steps. Generally, a random initialization step is required. Then the iterative process begins with assigning a "fitness score" to each individual according to how good it fits as a solution to the problem. A "fitter" individual is given a higher score. Then, each individual is given a chance to "reproduce". Fitter individuals are given higher chances to reproduce but less fitting elements still have a chance. Several individuals are selected from the new population to mate through a process called "crossover".

Crossover produces new children (or offsprings) with some features from each parent. Since less fitting members have smaller chances to reproduce, they fade away from the population during iterations. After recombination, a mutation operator can be applied. Each bit in the population can be mutated (flipped) with a low probability generally smaller than 1%. It may be a fixed value or can be a function of population size.

A good GA will converge to an optimal solution after several iterations. This basic implementation is called a Simple Genetic Algorithm (SGA) [11]. Genetic algorithms are not guaranteed to find the global optimum solution to a problem but they are generally good at finding "acceptably good" solutions to problems "acceptably quickly" [8].

Typical applications of genetic algorithms are numerical function optimization, scheduling, game playing, adaptive control, transportation problems, traveling salesman problems, bin packing and time tabling, etc.

3 The DBLF Method

Genetic algorithms are robust and can deal with a wide range of problem areas that can be difficult to solve with other methods. This is yet a weakness of the genetic algorithms. Since GAs are general methods, they do not contain valuable domain information; so hybridization with domain information can dramatically increase the quality of the solution and the performance of the genetic algorithm by "smarter" exploration the search space.

In the literature, researchers generally use some form of layering technique for 3D bin packing, in which a bin is packed layer by layer. However, layering algorithms have one drawback: their performance degrades if the input forces fragmentation of the loading surface in the early stages of the algorithm [12]. Our method packs the bin box by box. This method also has the disadvantage that decisions about where to place a box can only be made by local criteria so that algorithm may end up with a poor packing. To overcome this problem, genetic algorithms have been used to maintain a population of different packings.

Bottom Left (BL) and Bottom Left with Fill (BLF) methods are used in 2D bin packing. In BL, introduced by Jakobs [16], each item is moved as far as possible to the bottom of the object and then as far as possible to the left. BL is relatively a fast algorithm with complexity $O(n^2)$. The major disadvantage of the method is; empty areas in the layout are generated. Hopper [1], to overcome this disadvantage, develops the BLF algorithm. This algorithm allocates each object to the lowest possible region of the bin thus fills the empty areas in the layout. The major disadvantage of this algorithm is its $O(n^3)$ complexity.

Deepest BLF is an extension of the BLF method to cover 3D bin packing problems. An object is moved to the deepest available position (smallest z value) in the layout, and then as far as possible to the bottom (smallest y value) and then as far as possible left (smallest x value).

Since the complexity of the BLF algorithm is high, a computer efficient implementation of this algorithm is needed. Below is the Deepest BLF algorithm used in this work.

```
repeat
    get next box
    repeat
        get next position from the empty volumes list
        check if the empty volume is large enough
        if assigned empty volume is large enough
            intersection test with all boxes that could intersect at that position
                stopped when intersection is detected
            if intersection true
                update size of the empty volume at this position
            if intersection false
                insert box at this position
                iterate box into a deepest bottom-left position
                update position list by removing the inserted box's volume from that position
                delete unnecessary positions from list
                sort list of positions in deepest bottom left order
    until all positions are tried AND intersection is true
until all boxes placed
```

Alg. 1. The deepest BLF algorithm in pseudo code

In the beginning, the empty list has only one empty volume with dimensions same as the bin. As the algorithm iterates, new empty volumes are added to the list. The next box from the list is chosen to work on. Then each position in the list is checked to see if the current box fits in. If the empty volume is large enough, the position is checked for intersection with other boxes that were placed before. This check is needed because for an efficient implementation, all the volumes in list are not up to date. If there is no intersection at the chosen position, the box is inserted at this position and is iterated to the deepest bottom left position. Then the position list is updated and checked to see if there is any unnecessary position. In the final step, the new list is sorted in deepest bottom left order. Then the next box is taken from the list and the same selection procedure is repeated. Also as an additional constraint, the boxes are not allowed to rotate.

4 Generation of Test Data

Due to lack of shared test data for these kinds of studies, self generated test data are used. Three classes of test data are generated. The problems are generated with extension to algorithms used by [1] in 2D problem generation. These problem classes are generated so that an optimal solution is known for each problem. First two classes of problems are guillotineable (can be generated by straight cuts through remaining volumes). The algorithms are listed in Alg. 2 and 3.

```
user input (number of boxes to produce)
calculate number of rectangles to produce ($n = 1 + 7 * i$)
    where i is the number of loops required
repeat
    select randomly one of the existing boxes
    if start
        use object as large box
    place random point into large box
    construct 3 lines that intersect the box's width,length and height
    create 8 new boxes
until number of rectangles reached
```

Alg. 2. Algorithm for the creation of a guillotineable problem instance in pseudo code (method 1)

```
user input (number of boxes to produce)
calculate number of rectangles to produce ($n = 1 + 3 * i$)
    where i is the number of loops required
repeat
    select randomly one of the existing boxes
    if start
        use object as large box
    select a random  side of large box
    place a random point on this side
    select a random point along z axis
    construct 2 lines perpendicular to the point on the side and through the z point
    create 4 new boxes
until number of rectangles reached
```

Alg. 3. Algorithm for the creation of a guillotineable problem instance in pseudo code (method 2)

The third class of problems is non-guillotineable; hence are more difficult to solve. The algorithm is listed in Alg. 4.

```
user input (number of boxes to produce)
calculate number of rectangles to produce ($n = 1 + 10 * i$)
    where i is the number of loops required
repeat
    select randomly one of the existing boxes
    if start
        use object as large box
    place two random points into large box
    create a small box within these points
    extend from these two points to three axis and create 9 new boxes
until number of rectangles reached
```

Alg. 4. Algorithm for the creation of a nonguillotineable problem instance in pseudo code

Generated problems in each category have the following number of elements:

Table 1. Number of problems and dimensions in different categories

	Category 1 (Generated using algorithm 1)	**Category 2 (Generated using algorithm 2)**	**Category 3 (Generated using algorithm 3)**
Problem 1	15 (20 x 20 x 20)	16 (20 x 20 x 20)	21 (20 x 20 x 20)
Problem 2	29 (50 x 50 x 50)	25 (50 x 50 x 50)	31 (50 x 50 x 50)
Problem 3	50 (100 x 100 x 100)	52 (100 x 100 x 100)	51 (100 x 100 x 100)
Problem 4	106 (200 x 200 x 200)	100 (200 x 200 x 200)	101 (200 x 200 x 200)
Problem 5	155 (300 x 300 x 300)	151 (300 x 300 x 300)	151 (300 x 300 x 300)

The first number in the columns is the number of objects in the problem and the other three numbers in parentheses are the dimensions of the large box used as the basis for the algorithms. The differences of item numbers in the same class of problems in different categories are due to algorithms used in generation.

5 Implementation

The hybrid genetic algorithm used in this study consists of two stages. First, a genetic algorithm with order based encoding (OBE) is used to explore the search space using the encoded chromosomes. Then DBLF decoding algorithm is used to evaluate the fitness or the "quality" of the proposed packing pattern and transforms the pattern into the corresponding physical layout.

Order based encoding [15] is used for encoding of chromosomes in the algorithm. In OBE, each chromosome in the algorithm represents a candidate solution and is a permutation of all items in the problem. Hence the hybrid GA uses permutation representation.

The genetic algorithm starts by placing objects into chromosomes by feeding using the heuristics used for comparison, other chromosomes are generated randomly. The population is evaluated by a fitness function. Four different fitness functions are used in this work for comparison with each other: maximum height of the bin, the amount of wasted volume, ratio of volume of packed items to wasted volume and a combination of maximum height and ratio of packed items (70% of height and 30% of the ratio of packed items). Fitness functions are based on evaluation results from the Deepest Bottom Left with Fill method.

Each candidate solution is also checked for availability of reduction. If there is a subset of placement scheme so that there is no wasted value, that subset is marked in the algorithm so that those boxes in that subset are reduced from the problem space only for that specific solution. This knowledge is also kept during crossover and mutation operations (the genetic algorithm does not manipulate the reduced part of the solution). The main problem with reduction is: when some items are removed from the solution, it may not be possible to obtain the optimal solution; so reduction is applied on chromosome basis not on all of the population.

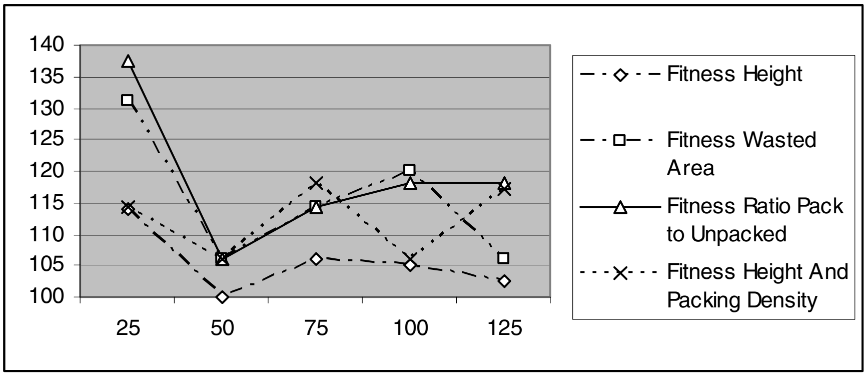

Fig. 1. Genetic algorithm solutions for the 3rd problem in the 1st category using different population sizes

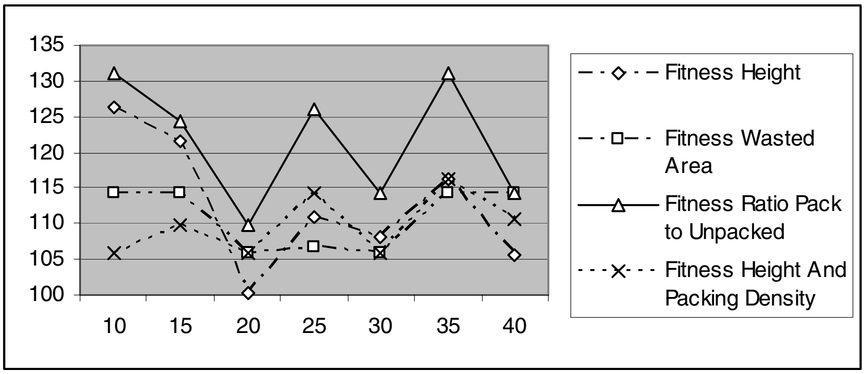

Fig. 2. Genetic algorithm solutions for the 3rd problem in the 1st category using different crossover rates

Population size used in this work is 50. A temporary population is generated using roulette wheel selection method. This method, as its name implies, simulates a roulette wheel. All chromosomes are placed on a wheel proportional to their fitness and new chromosomes are selected randomly. Elitist selection is used to preserve the best individual from the previous step. Crossover and mutation operators are applied with probabilities 20% and 0.05% respectively. The maximum number of generations is 1000. The algorithm stops if the optimal solution is found or number of generations is reached. Several experiments have been made to determine the population size and crossover rate. Solutions obtained for different population sizes and crossover rates are presented in Fig. 1 and Fig. 2.

OX operator proposed by Davis [13] is used for crossover operations. OX builds offsprings by choosing sublets of packages from one parent and preserving the relative order of other sub packages from the other parent. OX operator is chosen because it can preserve the ordering of the chromosomes that is important for these kinds of order based representations. Two points chosen at random are used for the crossover operator.

The genetic algorithm is presented in Alg. 5:

```
seed 4 chromosomes using 4 heuristics
initialize the rest of chromosomes by randomly exchaning two items per chromosome
repeat
    calculate each fitness of chromosome by decoding each chromosome using the DBLF algorithm
    setup the roulette wheel
    select temporary new population
    apply crossover
    apply mutation
    use temporary new population as new population
until number of generations are reached or optimal solution is found
```

Alg. 5. The hybrid genetic algorithm

6 Conclusion

The maximum heights of the bin found in each method are presented in the appendix. Genetic algorithms with height used as fitness function achieved to find the previously known optimal solution or solutions close to optimal solution. Genetic algorithms performed better than proposed heuristic algorithms. Results for the third class of problems are some far away from the optimal solution when compared to the first two classes. As the third class of problems is nonguillotineable, they are more difficult to solve.

Deepest bottom left with fill method combined with a hybrid genetic algorithm has been successfully applied to three dimensional strip packing problem. The proposed method produced high utilization rates. Genetic algorithms performed better than the heuristics methods as expected. Among the genetic algorithms, maximum height fitness function performed better than the other functions.

References

1. Hopper, E: Two-dimensional Packing Utilising Evolutionary Algorithms and other Meta-Heuristic Methods., A Thesis submitted to University of Wales for the Degree of Doctor of Philosophy (2000).
2. Fowler, R.J., Paterson, M. S. and, Tatimoto, S.L.: Optimal Packing and Covering in the Plane Are NP-Complete. Information Processing Letters, Vol. 12 (1981) 133-137.

3. Oluf Faroe, David Pisinger, Martin Zachariasen: Guided Local Search for the Three-Dimensional Bin Packing Problem. INFORMS Journal on Computing (2002).
4. Corcoran III A. L. and Wainwrigth R. L.: Genetic algorithm for packing in three dimensions. Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing SAC'92, Kansas City (1992) 1021-1030.
5. Ikonen, Ilkka and Kumar, A.: A Genetic Algorithm for Packing Three-Dimensional Non-Convex Objects Having Cavities and Holes. Proceedings of the 7th International Conference on Genetic Algorithms (1998) 591-598.
6. Martello, Silvano, Pisinger, David, Vigo, Daniele: The Three-Dimensional Bin Packing Problem. Operations Research, Vol. 48 (2000) 256-267.
7. Günther R. Raidl: A Weight-Coded Genetic Algorithm for the Multiple Container Packing Problem. SAC'99 San Antonio, Texas (1999).
8. David Beasley, David R. Bull and Ralph R. Martin: an Overview of Genetic Algorithms: Part 1, Fundamentals. University Computing, Vol. 15(2) (1993) 58-69.
9. Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor (1975).
10. Schaffer, J.D., and Selman, L.: Some Effects of Selection Procedures on Hyper plane sampling by Genetic Algorithms. Genetic Algorithms and Simulated Annealing (1987).
11. Goldberg, D: Genetic Algorithms in Search, Optimization and Machine Learning. Reading, MA, Addison-Wesley (1989).
12. Bram Verweij: Multiple Destination Bin Packing, Report, http://citeseer.ist.psu.edu/verweij96multiple.html (1996).
13. Davis L.: Applying adaptive search algorithms to epistatic domains. Proceedings of the 9th International Joint Conference on Artificial Intelligence, Los Angeles (1985) 162-164.
14. Jakobs, S: On Genetic Algorithms for the Packing of Polygons. European Journal of Operational Research Vol 88 (1996) 165-181.
15. Raidl G. R., Kodydek, G: Genetic Algorithms for the Multiple Container Packing Problem. Proceedings of the 5th Int. Conference on Parallel Problem Solving from Nature, Amsterdam, The Netherlands, (1998) 875-884.

Appendix: Test Results

Table 2. Problem class 1 test results

Method	Prob. 1 optimal: 20	Prob. 2 optimal: 50	Prob. 3 optimal: 100	Prob. 4 optimal: 200	Prob. 5 optimal: 300
GA with Fit. Fn: Height	20	50	100	204	300
GA w. Fit. Fn:Wasted Area	24	62	104	262	350
GA w. Fit. Fn: Ratio Packed – Unpacked	20	58	112	264	352
GA w. Fit. Fn: Height & Packing Dens.	24	57	112	256	342
Heuristic: Sorted by Height	24	71	124	348	435
Heuristic: Sorted by Width	32	75	136	437	448
Heuristic: Sorted by Volume	24	85	112	296	403
Heuristic: Sorted by Bottom Area	28	86	118	476	440

Table 3. Problem class 2 test results

Method	Prob. 1 optimal: 20	Prob. 2 optimal: 50	Prob. 3 optimal: 100	Prob. 4 optimal: 200	Prob. 5 optimal: 300
GA with Fit. Fn: Height	20	54	101	208	312
GA w. Fit. Fn:Wasted Area	22	62	112	251	339
GA w. Fit. Fn: Ratio Packed – Unpacked	20	62	112	251	339
GA w. Fit. Fn: Height & Packing Dens.	20	62	109	247	338
Heuristic: Sorted by Height	22	83	116	281	352
Heuristic: Sorted by Width	25	73	151	457	447
Heuristic: Sorted by Volume	23	66	126	343	437
Heuristic: Sorted by Bottom Area	24	84	133	404	405

Table 4. Problem class 3 test results

Method	Prob. 1 optimal: 20	Prob. 2 optimal: 50	Prob. 3 optimal: 100	Prob. 4 optimal: 200	Prob. 5 optimal: 300
GA with Fit. Fn: Height	20	53	108	219	330
GA w. Fit. Fn:Wasted Area	20	62	124	231	362
GA w. Fit. Fn: Ratio Packed – Unpacked	20	63	137	246	358
GA w. Fit. Fn: Height & Packing Dens.	20	63	138	248	339
Heuristic: Sorted by Height	31	77	188	388	484
Heuristic: Sorted by Width	31	75	237	388	525
Heuristic: Sorted by Volume	30	63	154	376	514
Heuristic: Sorted by Bottom Area	28	63	225	394	584