Considerations on Using Genetic Algorithms for the 2D Bin Packing Problem: a General Model and Detected Difficulties

Gia Thuan Lam
Vietnamese-German University
Le Lai Street, Hoa Phu Ward, Thu Dau Mot City,
Binh Duong Province, Vietnam
cs2014_thuan.lg@student.vgu.edu.vn

Viet Anh Ho
Dept. of Computer Science and Engineering
Frankfurt University of Applied Sciences
Frankfurt am Main, Germany
vietho@stud.fra-uas.de

Doina Logofatu
Dept. of Computer Science and Engineering
Frankfurt University of Applied Sciences
Frankfurt am Main, Germany
logofatu@fb2.fra-uas.de

Costin Badica
Faculty of Automatics, Computers and Electronics
University of Craiova
Craiova, Romania
cbadica@software.ucv.ro

***Abstract*—The 2-dimensional bin packing problem appears in various fields across many industries such as wood, glass, or paper industries. They may differ in terms of specific constraints with respects to each area but they all share a common objective that is to maximize the material utilization. Belonging to the class of NP-Hard problems, there exist no efficient method to solve it, but only approximate solution by combining a greedy placement strategy with some optimization techniques such as Genetic Algorithms. That combination approach is very popular in this topic, but few researchers have clearly presented their method and no one has explained the difficulty of applying Genetic Algorithms to this problem, making it difficult for new researchers to reimplement the known algorithms. In this paper, in addition to proposing a general framework for applying Genetic Algorithms to solving this problem, we also identify the main difficulties of using this approach and propose 2 genetic operators, path recombination and hill-climbing mutation, to support our genetic model.**

Keywords—2-dimensional bin packing, genetic model, greedy placement strategy, optimization technique, Genetic Algorithms, path recombination, hill-climbing mutation.

I. BACKGROUND

The 2-dimensional (2D) bin packing problem is a combinatorial optimization problem concerned with allocating multiple objects into rectangle containers of known dimensions (bins) with the aim of minimizing the number of bins needed. This problem has a huge application domain spreading through many industries, from the wood, glass, paper industries where regular (rectangle) shapes are concerned, to steel, leader, textile industries in which irregular (non-rectangle) items are handled. It can even be generalized to higher dimension and extends its application domain to more areas such as logistics and transportation. The constraints in these industries may vary, but all share a common objective that is to optimize the material utilization.

In computational theory, bin packing belongs to the NP-Hard [1] class, implying that no computationally efficient solution exists. Many approximation algorithms have been invented over the past decades, some of which are already very close to optimality in practice. Nonetheless, researcher's interest for this problem always remains so strong that it is still intensively researched since every little improvement can generate enormous economic values in mass-production industries.

One of the most popular approach to this problem is to combine a greedy placement strategy with an optimization technique which will search for a placement of objects that can make the placement strategy yield the best possible results. Many researchers have applied Genetic Algorithms to this problem. Few, however, have presented their method in detail, making it very difficult for new researchers to reimplement the known algorithms. Even worse, no researcher has ever specified the difficulties when applying Genetic Algorithms to this problem, causing other researchers to reproduce the same mistakes. In one of our researches [14], we recognized a strange behavior of optimization techniques in this problem that they did not really optimize anything. That inspires us to the writing of this paper where we continue our investigation in greater details to discover the rationale behind the inefficiency of those optimization methods such as Genetic Algorithms in the context this problem.

In this paper, in addition to proposing a general framework for applying Genetic Algorithm to this problem with the objective of easing the implementation process for new researchers and encouraging more researches in this topic, our main contribution is the identification all the problems we encountered during our study so that new researchers can

978-1-5386-3842-2/17/$31.00 ©2017 IEEE

understand which direction they should continue their investigation. Our model is concerned with using Genetic Algorithm to optimize the order of placement of items, the sequence deciding which items to be placed before others, so that a given greedy placement strategy can produce the best possible results. We restrict ourselves only to irregular 2D bin packing in which only 2D objects are involved.

II. Genetic Algorithms

Genetic Algorithms are an optimization technique inspired by natural processes of evolution, such as natural selection, genetic recombination and genetic mutation. Although the principles are simple, the impact is profound. They provide a promising solution to problems that are too complicated for humans to solve. Their applications cover a variety of areas such as Economics, Image Processing, Vehicle Routing Problem, Scheduling Applications, Robot Trajectory Generation, Parametric Design of Aircraft, DNA Analysis, Machine Learning and Multimodal Optimization. An extended list of Genetic Algorithms' applications can be found in [15, 18, 19, 20, 21].

In general optimization problems, Genetic Algorithm can be considered as a general meta-heuristic search technique which will search for a configuration or a set of parameters such that the performance of some system working on those parameters will improve. Consider the system as a function $f(x)$, $x \in X$ in which X is an n-dimensional domain of all possible configurations or sets of parameters of the function, a Genetic Algorithm will search for a vector $x \in X$ such that the value of $f(x)$ will be either minimized or maximized. This is normally impossible using complete search or random search in that the size of the search domain X in practice is usually immeasurable, whilst for optimization techniques like Genetic Algorithms, acceptable results can be generated in an acceptable time frame. There are numerous other optimization techniques such as Fuzzy Systems or Ant Colony Optimization, but the focus of this paper is Genetic Algorithms, hence other techniques will not be discussed. Further references to other optimization algorithms can be found in [16, 17].

The most fundamental structure of a Genetic Algorithm is consisted of a fitness function which evaluates a feature vector, namely individual, a selection procedure simulating natural selection to filter the individuals for the next generation, a recombination operator and a mutation operator that resemble the same recombination and mutation processes during biological birth. To apply Genetic Algorithms in any problem, the fitness function, the recombination operator and the mutation operator must be clearly defined with respects to certain constraints of the problem itself.

III. 2D Bin Packing

The 2D bin packing problem is defined as the problem of packing a given list 2D objects into as few rectangle bins of known dimensions as possible. An instance of the problem can be defined as $P = \{p_1, p_2 \ldots p_n\}$ in which n is the size of the given list of objects, each is represented as a polygon of k vertexes $p_i = \{(x_1,y_1), (x_2,y_2), \ldots (x_k,y_k)\}$ for some k. A solution to such an instance of the problem is the return value of the function $f(P)$ which returns the smallest possible number of rectangle bins required to pack all polygon $p_i \in P$. There are many greedy solutions available for this problem such as First Fit or Best Fit [12] which have been demonstrated in practice to be excellent solutions. It is, nonetheless, believed that there are certain orders of the initial input that may utilize more potential of the greedy placement strategy. Hence, over the past decades, many researchers have investigated into optimization methods that can optimize the initial order of placement and Genetic Algorithm is one of the most promising candidates.

Figure 1 and figure 2 shows an example of the input and output of a 2D bin packing algorithm, respectively. The first line of the input gives dimensions, the width and height of all rectangle bins, the second line represents the number of items to place n, followed by n lines, each of which describes an item as a polygon by listing a sequence of points.

```
1  3500 3000
2  5
3  0.0,0.0 1500.0,0.0 1500.0,1500.0 0.0,1500.0
4  0.0,0.0 1500.0,0.0 1500.0,1500.0 0.0,1500.0
5  0.0,0.0 1500.0,0.0 1500.0,1500.0 0.0,1500.0
6  0.0,0.0 1500.0,0.0 0.0,1500.0
7  0.0,0.0 1500.0,0.0 0.0,1500.0
```

Fig. 1. Sample input for 5 items.

The output in figure 4 is simply a graphical visualization of an optimal placement. In this case, we only need 1 bin, which is also an optimal solution.

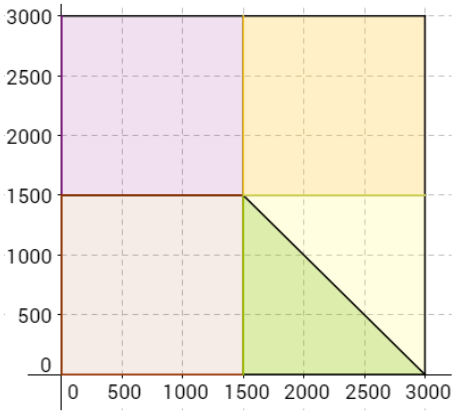

Fig. 2. Sample output for the placement of 5 items.

Albeit the optimal solution to the above example can be found easily, the complexity of this problem in theory is NP-Hard, implying that it is only possible to find a good approximation in general case.

IV. General Genetic Framework

To derive the genetic model for an optimization problem, firstly, the objective function needs to be defined. Let P be the given list of 2D objects of size n, $P = \{p_1, p_2 \ldots p_n\}$, in which p_i is a polygon $p_i = \{(x_1,y_1), (x_2,y_2), \ldots (x_k,y_k)\}$ where k is the

number of vertexes, *Per*(*P*) be a permutation of elements of *P* and the objective function *f*(*P*) is defined as the function that processes each element p_i in the relation to the its order in *P* and outputs the smallest possible number of bins required for packing all polygons of *P*. The optimization problem is defined as follows:

$$\min_{P' = Per(P)} \left(f(P') \right)$$

The domain of *P'* is all possible permutations of *P* and the role of the Genetic Algorithm in the context of this problem is to search for a permutation that minimizes *f*, the smallest possible number of bins for a placement of objects and also the fitness function in our Genetic Algorithm, as much as possible.

The general framework is presented by, first showing the layout of a Genetic Algorithm, then specifying the detailed adaption at each step. Figure 3 below shows a flowchart depicting the general model for a Genetic Algorithm.

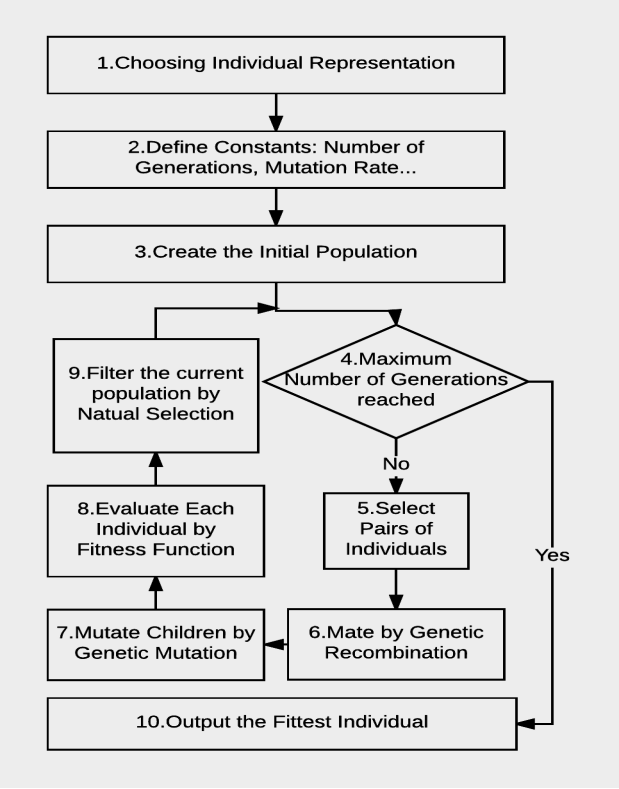

Fig. 3. General Model of Genetic Algorithm.

Let's consider the following part, which further explains for each step more in detail.

1. Choose individual representation scheme: Since in our problem, the targeted objective of optimization is the placement of object, a good representation should be a permutation of indexes of the given objects.
2. Define constants: There are various parameters required for a Genetic Algorithm, the selection of which depends on experience and the problem itself; one possible selection method is to try as many sets of parameters as possible and select the best set.
3. Generate the initial population: Since the individual is just a random permutation, the population is just a list of random permutations, but each of them needs to be evaluated using the fitness function f already defined.
4. Enter the loop until the maximum number of generations (iterations) is reached.
5. Select pairs for mating (recombination of genes): there are many standardized methods based on the fitness value computed by *f* such as Roulette Wheel Selection, Tournament selection or Best Fit Selection.
6. Mating using an appropriate recombination method: this is one of the main difficulty when applying Genetic Algorithms in this problem and will be discussed in section V; the used method must return valid children, which are valid permutations without duplicates or missing values.
7. For each child born from the mating process, use an appropriate mutation method to mutate its structure in a way that the mutated child is also valid. This is also one of the main difficulties of this algorithm and will be discussed in section V.
8. Reevaluate the population using the fitness function *f*.
9. Replace the old population with the new one. Go to 4.
10. Output the fittest individual during the whole process.

This model follows the standard model as first proposed by John Holland [3], but instead of using the standard operators for binary bit strings, it is emphasized in step 6 and 7 that more appropriate methods must be used. The reason will be discussed in section V. In addition to this model, there are many other models of applying Genetic Algorithms, but they are all based on this standard model. In practice, researchers should not implement Genetic Algorithms themselves, but it is better to utilize standard libraries such as Pyvolution [4], Pyevolve [5], Pyeasyga [6], or our recommendation is the Distributed Evolutionary Algorithms in Python (DEAP) [7]. When implementing the algorithm for experimental purposes, we should only focus on the main problems which occur at step 6 and 7 and the choice of an appropriate fitness function *f*, which is the biggest problem.

V. Difficulties When Applying Genetic Algorithms To 2D Bin Packing

This is the central component of this paper where we will discuss the problems with applying Genetic Algorithms to solving the 2D bin packing problem. In section IV, we have identified three possible problems: the fitness function, the recombination method and the mutation method. In this section, we are going to why they can become a problem and propose some temporary methods to handle them. However, these problems cannot be solved completely because of both the nature of the 2D bin packing and Genetic Algorithms.

A. Problem with the fitness function

The fitness function is directly involved in the selection process which plays an imperative role in any genetic algorithms. The selection process requires the values of the fitness function to be diverse. The more diverse the possible values of the fitness function, the better the selection works. This is where the problem evolves.

It is a well-known fact in the problem of 2D bin packing that greedy appropriate placement strategy without any optimization techniques are already excellent in practice. It has been demonstrated that intuitively simple algorithms such as First Fit or Best Fit require no more than 11/9 times the optimal number of bins in the worst case [8]. Therefore, if we only focus on the number of bins for the fitness function, it may lead to the case that all the fitness values may become equal, especially when they get closer to the optimal solution, making the selection strategy look like a random selection and the whole algorithm not better than a random search.

In 2015, Duarte Nuno Gonçalves Ferreira [9] implemented the First Fit strategy and compared it with the version optimized by Genetic Algorithm. His results are summarized in the table 1. The field input size stands for the number of input objects. For each input size, there are multiple test cases, but table I only shows the average values of them. As it can be seen, there is little to no difference between First Fit algorithm and Genetic Algorithms. Ferreira did not explain the reasons, but during our study, we believe part of the reason is related to the diversity of fitness function. This is a serious problem because researchers who only apply the algorithm but do not carefully analyze the internal processes happening inside may never discover why the optimization techniques have not optimized anything.

TABLE I. SUMMARY OF FERREIRA'S COMPUTATIONAL RESULTS

Input Size	Number of bin generated	
	First Fit Decreasing	*Genetic Algorithms*
20	3.78	3.65
40	6.96	6.94
60	10.34	10.38
80	13.87	14.13
100	16.35	16.70

One way to get around with this problem is to introduce different parameters for the return values of the fitness. During our experiment, we concatenated the number of bins n generated from the placement algorithm with the average density of free space (the ratio between the free space in each bin and the bin area) in the first n-1 bin(s). The algorithm works better this way, but the speed of convergence is terribly slow.

For implementation of the placement strategy, we highly recommend the open source implementation of Moises Baly [10]. His implementation is a variant of the First Fit algorithm for the 2D irregular bin packing (non-rectangle bin packing). Albeit it is not very efficient, it is a very good start for new researchers who have little experience in this field. The idea is to treat each item as a rectangle by finding its rectangle bounding box. By so doing, known and simple algorithms for regular bin packing (rectangle bin packing) can be directly applied to the newer and more complicated irregular bin packing. One method to increase the efficiency of this method is to find the minimal enclosing box [11] instead of any rectangle bounding box.

B. Recombination and Mutation

Unlike the other problem, operators that work with binary strings will not work in this case because the resulting child maybe not valid. Let n be the number of items, a valid child is a permutation $\{a_1, a_2 \ldots a_n\}$ such that $1 \le a_i \le n$ and there are no $1 \le i \le j \le$ n such that $a_i = a_j$. Binary mutation operator may make the child has element bigger than n or equal to 0 and recombination method such as crossover may give birth to a child with duplicated values.

Our genetic algorithm is concerned with the ordering of placement, implying that it belongs to the class of ordering genetic algorithms [13] with the ordering being represented as a permutation. This class of genetic algorithms is particularly difficult in that it is hard to define suitable recombination and mutation operators. The Travelling Salesman problem is a typical example of this class.

For the algorithm to work, the operators must satisfy two conditions, the first is that the resulting child after recombination or mutation must be a valid permutation, and the second is that after recombination, the child must possess to some extent inherited genetic information from its parents. There have been many investigations into the suitable encoding scheme where the standard operators can work. One notable result is the ordinal representation, which is, however, a disappointing method. Let's take an example to understand how ordinal representation works and why it is not useful.

TABLE II. EXAMPLE FOR ORDINAL REPRESENTATION

Permutation	Remaining Numbers	Ordinal Order
1 3 5 6 2 4	**1** 2 3 4 5 6	**1**
3 5 6 2 4	2 **3** 4 5 6	1 **2**
5 6 2 4	2 4 **5** 6	1 2 **3**
6 2 4	2 4 **6**	1 2 3 **3**
2 4	**2** 4	1 2 3 3 **1**
4	**4**	1 2 3 3 1 **1**

Table II illustrates the method to transform a permutation to an ordinal order in which normal crossover and mutation are applicable by an example. After applying genetic operators on this form, it is converted back to a valid permutation. The nice property of this encoding scheme is that the resulting child is always a valid permutation. It has been, however, demonstrated in practice that this method generates very poor result. One simple reason is that each value of the permutation

depends heavily on the previous values. Therefore, one value changes and the remaining values also change, which means that the genetic information cannot be passed down from parents to children efficiently.

Some other notable encoding schemes such as path encoding and adjacency matrix representation yield much better results in practice, but their results are still very modest considering the potential of Genetic Algorithms. Further references relate to those methods can be found in [2]. In our study, we do not choose any other representation other than the pure permutation. We also propose two operators for it, which are some variants of the path encoding operators. We would like to outline our operators as a reference for new researcher.

Our recombination operator inputs two parent permutations and output one child. It is represented as follows.

1. Create a graph of n (the number of items) nodes, set an undirected edge between i and j if they stand next to each other in one of the two parent permutations.
2. Create an empty child permutation.
3. We select a random unvisited node in the graph and travels into any unvisited neighbor node. We will travel until there is no more neighbor to go and add all nodes in the way during our travel to the child permutation.
4. If the child permutation is incomplete, go back to 4. Otherwise, output the resulting child.

To produce 2 children as it does in the model in section 3, we simply do this procedure 2 times. The algorithm looks very simple but an important property of this recombination algorithm is that the child created by this recombination algorithm has a stable feature: the average path length transferred is always approximately 10. Regardless of the permutation length and structure, regardless of which node to begin, even though there are still many path consisted of only one number, the average length remains stable. If we consider only directed edge then that value is approximately 5 and is also very stable. This result has been proved by trying a thousand random permutations of varying lengths from a 10 to 1000000. Thanks to this stable feature, genetic data are partially passed down from parents to children.

For the implementation of mutation operator, we use a variant of Hill Climbing algorithm and consider the swap operation instead of the change operation like in the standard mutation operator to make sure that the resulting child is valid. During each run, the algorithm will try at most 10 random swaps in the permutation, and take the local minimum fitness value of each resulting permutation. If minimum value is smaller than the current value, real changes will be made on the original permutation. It only stops when no more improvement can be done. The hill-climbing mutation operator can be described as follows.

1. Create 10 copies of the current permutation.
2. Pick two random positions in each copy and swap.
3. Compute the fitness function for each copy and take the copy with minimum value.
4. Compare the minimum value with the current permutation. If it is better, replace the current permutation with the copy and go back to 1. Otherwise, output the current permutation.

These operators work in our test cases, but we cannot guarantee it will work for all general cases. New researchers can find other operators for ordering genetic algorithms that are more suitable for their need in [13].

VI. Results And Conclusion

The aim of our work is to propose a general model for applying genetic algorithm in the problem of 2D bin packing. By so doing, we can identify the main problems with this genetic approach and explain the myth that has been found by previous researchers, why greedy placement strategy works so well compared to those using optimization methods such as Genetic Algorithms. We hope that it will simplify the task of implementation for new researchers in this topic and help them identify what needs to be improved. We are not certain that there are no other problems, but those identified problems should be most likely to occur as they must be involved in all Genetic Algorithms, especially the problem with the fitness function which implies that Genetic Algorithms work well only if the placement strategy is poorly implemented, but if the placement strategy is well-implemented, one should not use Genetic Algorithms for 2D bin packing.

Last, but not least, we would like to show a few examples where Genetic Algorithms really improves the normal greedy strategy. Our implementation is consisted of all the proposed operators and fitness functions we described in section V.

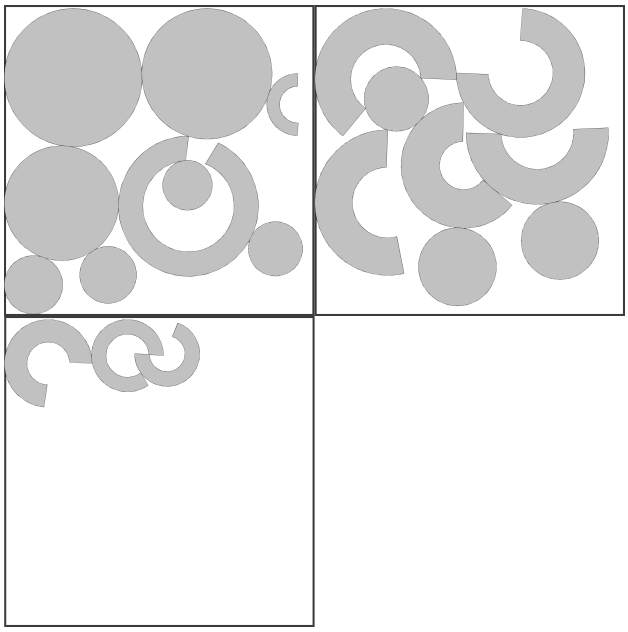

Fig. 4. An example placement of 20 shapes and circles using only greedy strategy

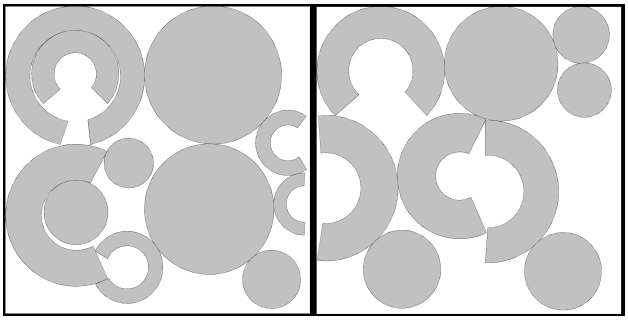

Fig. 5. An example placement of the same input from figure 3 but using Genetic Algorithms

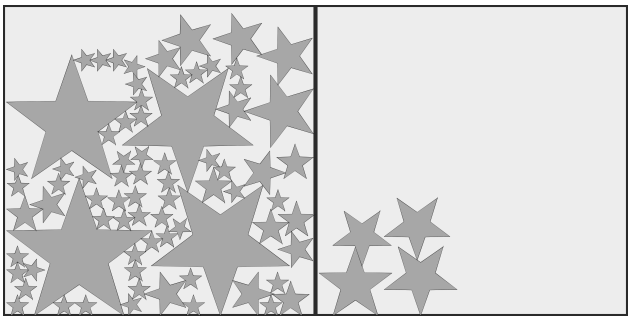

Fig. 6. An example placement of stars using only greedy stratefy

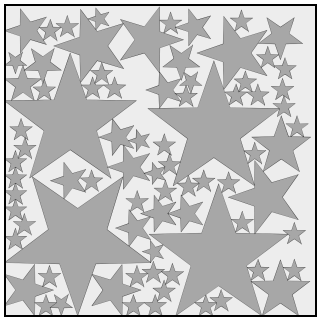

Fig. 7. An example placement of the same input from figure 1 but using Genetic Algorithms

Figures 4 and 5 and figures 6 and 7 show two examples where genetic algorithms improve the number of bins used. Those outputs are also visually optimal solutions to the given inputs. The problem of slow convergence due to the fitness function still exists and especially terrible in these test cases where the numbers of bins used are so small, but using the tweaks we introduce in section V and extreme patient and luck, we are finally able to generate good results. In practice, a generally well implemented placement strategy may be able to generate the optimal solution to the above examples in one run, whilst in this case, the fitness function efficiency must be decreased to avoid the case that it can generate the optimal output in the first run.

REFERENCES

[1] David S. Johnson and Michael Garey, "Computers and Intractability: A Guide to the theory of NP-Completeness," San Francisco, CA; W.H. Freeman and Company, 1979.

[2] Thomas Back, David B. Fogel, Zbigniew Michalewicz, "Handbook of Evolutionary Computation," IOP Publishing Ltd. Bristol, UK, 1997.

[3] Holland, J.H, "Adaption in Natural and Artifical Systems". Ann Arbor, MI: The University of Michigan Press. 1975.

[4] Pyvolution, "An Evolutionary Algorithms Framework in Python," Retrieved 20 April 2017, from http://pythonhosted.org/Pyvolution/

[5] Pyevolve, "An Evolutionary Algorithms Framework in Python," Retrieved 20 April 2017, from http://pyevolve.sourceforge.net/intro.html.

[6] Pyeasyga, "A simple and easy-to-use implementation of a Genetic Algorithm library in Python," Retrieved 20 April 2017, from https://github.com/remiomosowon/pyeasyga.

[7] DEAP, "A Novel Evolutionary Computation Framework for Rapid Prototyping and Testing of Ideas," Accessed on 20 April 2017, Available from https://github.com/DEAP/deap.

[8] Johnson, David S., "Near-optimal bin packing algorithms," Thesis (Ph. D.) - Massachusetts Institute of Technology, Dept. of Mathematics, 1973, Pages 398-399.

[9] Duarte Nuno Gonçalves Ferreira, "Rectangular Bin-Packing Problem: a computational evaluation of 4 heuristics algorithms," U. Porto Journal of Engineering, Pages 35-49, 2015.

[10] Moises Baly, "Java Implementation for 2D Bin Packing," Accessed on 20 April 2017, Available from https://github.com/mses-bly/2D-Bin-Packing.

[11] Joseph O'Rourke, "Finding minimal enclosing boxes," International Journal of Computer & Information Sciences, Volume 14, Issue 3, Pages 183–199, June 1985.

[12] Bennell, J.A, and Oliveria, J.F, "50 years of irregular shape packing problems: a tutorial," Southampton, UK, 2018 (Discussion Papers in Centre for Operational Research, Management Science and Information Systems, CORMSIS-08-14).

[13] Thomas Baeck, D.B Fogel, Z Michalewicz, "Evolutionary Computation 1: Basic Algorithms and Operators," January 1, 2000 by CRC Press

[14] Gia Thuan Lam, Doina Logofatu, Viet Anh Ho, "Considerations on 2D-Bin Packing Problem: Is the Order of Placement a Relevant Factor?," SIMULTECH, 2017, in press.

[15] C. Bastos-Filho, D. Chaves, F. e Silva, H. Pereira, J. Martins-Filho, "Wavelength Assignment for Physical-Layer-Impaired Optical Networks Using Evolutionary Computation," Journal of Optical Communications and Networking, Volume 3, Issue 3, Pages 178-188, March 2017.

[16] Mustafa ServetKıran, OğuzFındık, "A directed artificial bee colony algorithm," Applied Soft Computing, Volume 26, Pages 454-462, January 2015.

[17] Radu-EmilPrecup, Radu-CodruţDavid, Emil M.Petriu, StefanPreitl, Mircea-BogdanRădac, "Novel Adaptive Charged System Search algorithm for optimal tuning of fuzzy controllers," Expert Systems with Applications, Volume 41, Issue 4, Part 1, Pages 1168-1175, March 2014.

[18] Barrie M.Baker, M.A.Ayechew, "A genetic algorithm for the vehicle routing problem," Computers & Operations Research, Volume 30, Issue 5, Pages 787-800, April 2003.

[19] Rafael StanleyNúñez Cruz, Juan ManuelIbarra Zannatha, "Efficient mechanical design and limit cycle stability for a humanoid robot: An application of genetic algorithms," Neurocomputing, Volume 233, Pages 72-80, April 2017.

[20] Alisson S.C. Alencar, Ajalmar R. Rocha Neto, João Paulo P. Gomes, "A new pruning method for extreme learning machines via genetic algorithms," Applied Soft Computing, Volume 44, Pages 101-107, July 2016.

[21] Aref Miri, Karim Faez, "Adaptive image steganography based on transform domain via genetic algorithm," Optik - International Journal for Light and Electron Optics, Volume 145, Pages 158-168, September 2017.