Bin Packing

An Evolutionary Algorithm

Igor Nelson Garrido Da Cruz
Faculdade de Ciências e Tecnologia
Universidade de Coimbra
Portugal
igorcruz@student.dei.uc.pt

***Abstract* — with the present paper we are going to implement and discuss the results obtained with a practical Evolutionary Algorithm implementation of the Bin Packing Problem, also known as the BPP. We are also going to discuss the objective function and the best operators, such as mutation and crossover, to achieve a balanced algorithm which in terms of performance can compete with more complicated algorithms.**

Keywords—Evolutionary computing, Bin Packing, Biology Algorithms, Genetic Algorithms

I. Introduction

Evolutionary computing is the name given to a sparse amount of problem-solving techniques which are based on biological evolution [1]. This subject is still very recent in the academy; however the good results achieved with such techniques and the simplicity subjacent to its implementation, made it practically possible to grow outside of the leading-edge scientific research and expand to practical applications in the industry and commerce.

In this paper we pretend to show and discuss the results achieved in a practical implementation of an Evolutionary Algorithm for the Bin Packing problem solving and achieve a solid algorithm which in a fair amount of time can give good results.

II. Backgroud

The Bin Packing problem belongs to the family of grouping tasks. The objective of this problem is to separate a set of elements into separate subsets not exceeding the bin capacities and minimizing the number of bins used to store the elements.

<u>Problem formulation:</u>

"We have N bins with the capacity of C and n indivisible elements, each having the weight of Wi <= C (1 <= i <= N). The elements have to be packed into the bins in such way that the minimal number of bins should be used and bins capacity should not be exceeded." [2]

There has been a lot of work related with this algorithm recently. **Garey and Jonhson (1979)** proposed multiple greedy algorithms for solving this problem. One of the best is the First Fit Decreasing that guarantees results near the optimal solution in a very small amount of time. In terms of Evolutionary Algorithms, **Falkenauer (1994)** proposed GGA, Grouping Genetic Algorithm, one of the best algorithms known up to the date for solving problems which belong to the categories of grouping tasks.

However there is still a lot of work to be done in this area of investigation like choosing the best parameters for mutation; propose good crossover techniques and choose the best algorithm setup. With this paper we pretend to present our problem implementation details and discuss the results achieved with it.

III. Methods

A. Solution Space and Representation

The solution space of the bin packing problem must obviously contain the optimal solution for the problem. So we must consider all the existing items belonging to all of the existing bins. The optimal solution will be one of this combinations.

In order to be evaluated, the individuals must be represented in a suitable way; the representation can also affect the problem complexity.

The representation chosen for solving the problem is very simple and intuitive. It consists in a Hash Map in which each key is a bin and the values represent the index of the items inside.

Consider the following representation:

"1: [50, 19, 10], 2: [75, 25]"

This representation means that the bin number 1 contains 3 items the 50, 19 and 10 and the bin number 2 contains 2 other items identified by 75 and 25.

For implementation of the recombination operator we converted this representation into a more intuitive one. For each item we assign 1 bin:

"1213231"

Meaning that the items 0, 2 and 6 go into the bin 1, the items 2 and 4 go into the bin 2 and the items 3 and 5 go in the bin 3. The usage of this representation instead of the first one for the recombination algorithm will be explained in more detail in the recombination sub-chapter E.

B. *Objective Function*

To solve this problem in an efficient way we cannot simply use the number of bins as an objective function to be minimized. In fact, using the number of bins to guide the algorithm behavior would lead to suboptimal solutions. Let's consider the optimal +1 solution for the number of bins. The amount of different individuals with that number of bins is very high and we have no way of differentiating them conducting the algorithm in the correct direction to find the optimal solution, since the cost would be the same for all of them.

The Objective function chosen was proposed by **Falkenauer in 1994** and it's given by:

$$FF = \sum_{i=1}^{nb} \left(\frac{F_i}{C}\right)^k$$

Where nb is the number of bins, Fi is the sum of weights for each bin, C is the capacity of the bin and finally k is a constant which **Falkenauer** suggests to equal to 2.

In this problem is necessary to put the maximum number of elements in each bin in order to create space in the other bins and consequently reduce the total number of bins. This is exactly what the above function does. In two similar solutions it prioritizes the one with more free space in the bins.

C. *Initial Solutions*

In our implementation the initial solutions are very simple to calculate. We perform a shuffle in the original array of weights and by the order the items are, we divide them by bins, not allowing the capacity restriction to be broken. With this initial solution we have a fast algorithm for generating initial solutions and they are already valid.

D. *Neighbourhood*

In order to solve our problem in an efficient way we have to explore our neighbourhood in a way that all the solution space can be explored. For that we must allow that any possible item can be assigned to any possible bin.

E. *Mutation Operators*

The mutation operators chosen to explore the neighbourhood were the swap which allows two items to be swapped simultaneously between two bins and the insert which given 1 item from a certain bin inserts this item in another randomly chosen bin. These operators are often used in permutation problems like this and its efficiency has already been demonstrated.

In addition, there are moves which are forbidden. The moves within the bins are forbidden, because they wouldn´t improve the solution in any aspect.
These operators can lead to invalid solutions. That solutions are evaluated with a negative value, which is as lower as the number of bins that are broking the restriction.

The number of possible inserts is given by the number of elements of the problem multiplied by the number of bins minus one. Because considering one item from any bin then we would have the number of bins minus one possibility of choosing the destination bin.

On the other hand, the number of swaps possible is given by:

$$\binom{N}{2} - \sum(\binom{ni}{2})$$

We calculate the combinations of 2 from all the items and subtract the sum of combinations of 2 for each bin because we don't want to perform swaps inside the same bin.
After that, we already have the total number of inserts and swaps that are possible for a given solution; we just need to use the operators with a proportional probability in order to explore a uniform neighbourhood.

F. *Recombination Operators*

To implement the recombination operators we changed our representation into a more intuitive one. We used an array that assigns a bin to each element. In the bin packing problem we have the assignment problem, which can be easily explained with an example:

1) *"00112103"*
2) *"11002013"*

Although these representations are different, for us they represent the same solution, for example if we start with the representation 1 and change its zeros to ones and ones to zeros, we would end up with the representation 2.

To solve this problem we built a matrix that for each bin calculates the best correspondence of bins and after we calculate 1 / the amount of correspondences or the number of bins+1 if there is no correspondence. We do this transformation because the Hungarian Algorithm for solving the assignment problem receives a matrix for minimization.

After this we just perform a uniform transformation from the father number one in direction of the father number two, creating an individual which combines elements of both parents.

In the results chapter we also present results comparing an algorithm which uses assignment and other who doesn't to demonstrate the difference.

IV. Implementation

The algorithm used for this work is based on the following perspective.

1. A random population is generated;
2. The population is evaluated;
3. The population is ranked according to the utility of each individual;
4. The individuals who are going to generate descendants are selected

5. Children are generated from the parents selected in step 4
6. Children are evaluated
7. The best from both parents and children are selected generating the next generation
8. Go to step 2 until we achieve the desired amount of generations.

From this algorithm we created a second one, eliminating the steps 6 and 7.

So, let's call the Algorithm in which all the population is replaced with a new one ALG1 and the one in which each new population is generating choosing the best of both parents and children population ALG2.

V. Experimental Results

The experimental results are in the annexes due to its large size.

VI. Discussion

With the experimental results obtained we can see that in general the algorithm seems to perform in an efficient way.

In Annex 1 and 2 we can see that the ALG2 seems to have a faster convergence than the ALG1. This happens because from both parents and children we choose only the best and ignore the other individuals.

When we study the Annex 3 in detail we can see that the number of individuals of the populations seems not to influence neither the speed of convergence of the algorithm neither the proximity of the optimum value.

Finally in the annex number 6 we can see, in color blue, ALG1 with mutation only, serving as a guideline for the ones with recombination. In red, the algorithm which uses normal uniform recombination performs worse than the this one, since it causes noise by trying to approximate two solutions that can represent the same individual.

The more we increase the recombination probability, slower the algorithm converges, this can happen because adding more recombination deteriorates the solutions achieved with mutation and makes the algorithm slower achieving good results, however the recombination can be useful when the algorithm starts to stabilize making a new promising individual.

In green we see the algorithm which uses the best assignment that performs slightly better than all the others.

VII. Conclusions

With this report we intend to study the efficiency of an Evolutionary algorithm for the Bin Packing problem.

The implementation of such algorithm was obviously achieved, however the performance of the recombination operators does not perform as good as firstly expected and in contrast the performance of the mutation operators was better than what we though. This seems to happen because the addiction of crossover in this kind of evolutionary algorithm adds some noise to the population which is hard to deal with. Although using recombination and choosing the best assignment for the representation we can produce a more reliable algorithm because we reduce the noise added by the operations of recombination.

The new individual created with recombination is most of the times invalid, due to the restrictions of capacity of each bin. However the usage of this operator is important to allow new solutions to be found when the algorithm starts to stabilize since it gives the opportunity of finding a new individual with more than a simple mutation step.

The algorithm which implements the best of parents and children for the next generation performed better than the algorithm that mutates and recombines only the elements in 1 generation. The convergence of this algorithm was faster since it works with a wider spectrum of individuals per iteration and gives more priority to the most apt ones.

Finally it was important to implement an evolutionary algorithm to have contact with such techniques and improve our theoretical knowledge, having the opportunity to experiment multiple details in a real context.

References

For the execution of the present work and writing of this paper, multiple references were consulted.

[1] A.E. Eiben and J.E. Smith, Introduction to Evolutionary Computing, Springer, Natural Computing Series

[2] Adam Stawowy, A Evolutionary based heuristic for bin packing problem, Faculty of Management, AGH University of Science and Technology, Gramatyka 10, 30-067 Krakow, Poland

[3] Bin Packing instances by Prof. Dr. Armin Scholl & Dr. Robert Klein, can be found at http://www.wiwi.uni-jena.de/entscheidung/binpp/bin1dat.htm at June 2013