Genetically Designed Heuristics for the Bin Packing Problem

Oana Muntean
Department of Computer Science
Faculty of Mathematics and Computer Science
Babes-Bolyai University
Kogalniceanu 1, Cluj-Napoca, 400084
Romania.
oana_muntean85@yahoo.com

ABSTRACT

The bin packing problem (BPP) is a real-world problem that arises in different industrial applications related to minimization of space or time. The aim of this research is to automatically design an algorithm that places a collection of objects into the minimum number of fixed-size bins. For generating this heuristic we use Genetic Programming (GP) with a special set of terminals. The evolved strategy is compared to Best Fit Descending (BFD), a well-known heuristic for the bin packing problem. The results emphasize that evolved GP heuristics can perform equally and sometimes even better than BFD for the considered test problems.

Categories and Subject Descriptors

I.2.6 [**Learning**]; I.2.8 [**Problem Solving, Control Methods and Search**]

General Terms

Algorithms

Keywords

Genetic Programming, Heuristics, Bin Packing

1. INTRODUCTION

Bin packing arises in a variety of packaging and manufacturing problems, dealing with distribution of objects into fixed-capacity bins. To minimize cost and waste, it is demanded to lay out the objects so as to use as few bins as possible.

Many human-designed heuristics have been proposed for solving this problem. The most popular ones are Best Fit Descending (BFD) and First Fit Descending (FFD).

Genetic Programming (GP) [3] technique is used here to detect automatically Bin Packing Problem (BPP) heuristics for different classes of given objects. An evolved heuristic is a function which depends on the current state of the placement. The function will help us to choose which bin is suitable for placing the current object.

Several heuristics have been designed using this procedure. The heuristics have usually been trained on some instances and later they have been tested for some other instances in the same class. Numerical experiments have shown that the evolved heuristics can perform better than BFD for instances having certain properties. However, a single evolved heuristic was not able to conclusively beat BFD for all instances. This is why the best strategy is to combine the evolved heuristics: each of them is applied to a particular instance and the best solution (with minimum number of bins) is given as output.

The paper is organized as follows: Genetic Programming is briefly described in Section 2. The Bin Packing problem and the Best Fit Descending algorithm are described in Section 3. Related work in the field of evolution of heuristics is overviewed in Section 4. The proposed method is deeply presented in section 5. Several numerical experiments are performed in Section 6. Conclusion and further work directions are given in Section 7.

GECCO'07, July 7–11, 2007, London, England, United Kingdom.

2. GENETIC PROGRAMMING

Genetic Programming technique provides a framework for automatically creating a working computer program from a high-level problem statement of the problem [3]. Genetic Programming achieves this goal by genetically breeding a population of computer programs using the principles of Darwinian natural selection and biologically inspired operations. Five major preparatory steps [3] must be specified in order to apply a GP technique to a particular problem:

1. the set T of terminals (e.g., the independent variables of the problem, zero-argument functions and random constants)
2. the set F of primitive functions,
3. the fitness measure (for explicitly or implicitly measuring the quality of individuals in the population),
4. certain parameters for controlling the run
5. the termination criterion and the method for designating the result of the run.

These preparatory steps are problem dependent so they must be specified, by a human user, for each particular problem.

3. BIN PACKING PROBLEM

Several types of BPP are well-known in practice: 1-dimensional (1D), 2-dimensional (2D) and 3-dimensional (3D). In this paper only the case of 1D packing is considered. The problem is stated as follows:

Given a set of n objects, having the sizes (volumes) $d_1, \ldots, d_n$ and any number of fixed capacity (K) bins, it is required to place all the objects in the smallest number of bins.

BPP is an NP-complete problem [2]. No polynomial-time algorithm for solving BPP problem is known. Several heuristics for solving BPP problem have been proposed. The most important are First Fit (FF) and Best Fit (BF). These algorithms can be improved by sorting the objects in decreasing order. The corresponding variants of the algorithms are called First Fit Descending (FFD) and Best Fit Descending (BFD).

Best Fit Descending can be described as shown in Algorithm 1.

Algorithm 1 Best Fit Descending algorithm

1: sort the objects descending according to their size
2: **for** each object i **do**
3: find bin j such that object i maximizes filled capacity
4: put object i in bin j
5: **end for**

4. RELATED WORK

A different approach for evolving BPP heuristics using GP has been proposed in [1]. The differences between the related approach and the one described in this paper are given in Table 1.

5. GP FOR BPP

In this section we address the problem of finding heuristics that can solve BPP rather than solving a particular instance of the problem. GP technique is used for evolving a function f that gives a way to place the objects such that a minimum number of bins is used.

The idea behind the usage of this function is the fact that Step 3 of the Algorithm 1 (described in Section 3) can be rewritten as:

find bin j such that K - $filled_j$ - d_i is minimal,

where $filled_j$ is the filled capacity of bin j.

Only bins where object i can be placed are considered. All the others are ignored.

The optimal bin is the one for which the function K - $filled\ j$ - d_i is minimal. Thus we have a function depending on 3 parameters: K, $filled_j$, d_i. Our idea is to generate a function f by using Genetic Programming.

The new algorithm (called BFD-f) involving function f is given in Algorithm 2.

Function f depends on information about the current state of the placement. Ideal case would be to send all information available at current moment as input to that function. This means that we have to send the size of each object, bin size, unfilled capacity for each bin, etc. But such a function would have a variable number of arguments because the instances have a variable number of objects. Moreover, the number of existing non-empty bins is also variable. This number depends on the current state of the placement.

Algorithm 2 Best Fit Descending algorithm involving function f (BFD-f)

1: sort the objects decadently according to their size
2: **for** each object i **do**
3: find the bin j such that f is minimal
4: put object i in bin j
5: **end for**

It would be difficult for GP to evolve such function (especially due to the variable number of arguments). Thus just some partial information from the current state of the placement process is extracted. This information should be independent with respect to the instances size. Moreover, it must be computed easily because it is used each time a new object is placed in some bin.

The partial information set to GP is described in section 5.2.

5.1 Fitness Assignment

The fitness is assigned to the function f in the current population by applying BFD-f on several data sets (training set) and evaluating the results.

The fitness of a GP chromosome encoding a function f is equal to the total number of utilized bins over total number of instances in the training set. Because the minimum number of bins is required, the entire fitness function has to be minimized.

5.2 Terminals and Functions Sets

For evolving path function f we consider a set T of terminals involving the following elements:

1. Bin size,
2. Occupied volume of the current bin,
3. Size of the current object,
4. Sum of the already placed objects,
5. Sum of the objects not placed yet,
6. The smallest not filled volume,
7. The emptiest bin containing at least one object,
8. The size of the next object to be placed.

Set T of terminals is chosen in such way to be independent of the instance size. This choice confers flexibility and robustness to the evolved heuristic.

For evolving a GP function for BPP problem we may consider the following set of function symbols:

$$F = \{+, -, \%, *, max, min\}.$$

6. NUMERICAL EXPERIMENTS

In this section we evolve a function f which is embedded into the BFD-f heuristic. Several BBP instances are used for training purposes. The best obtained function is applied, at the end of the search process, for solving several other BPP instances (test set).

6.1 BPP instances

Several difficult data sets, taken from [4, 5] have been used in these experiments. The data are divided in different classes having different properties.

The first set of problems (BPP_1) has the following properties: $K = 1000$, $n = 50, 100, 200, 500$ and d_i are randomly generated over different intervals.

Table 1: The differences between our approach and the one proposed in [1].

The approach proposed in [1]	Our approach
The modified algorithm was First Fit.	The modified algorithm was Best Fit Descending.
The best obtained heuristic performs equally to First Fit.	Our evolved heuristics perform (in some cases) better than Best Fit Descending. Note that BFD performs at least as well as FF.
Invalid individuals (i.e. put all objects in the first bin) can appear in the system. These individuals are highly penalized.	No invalid individuals can appear in the system.
Complicated formula for fitness computation.	Fitness of a GP tree is equal to the number of utilized bins.
Only 3 terminals have been used (the filled capacity, bin capacity, size of current object).	9 different terminals have been used.
Function set have been $F = \{+, -, *, /, abs\}$.	Function set is $F = \{+, -, *, \%, min, max\}$.
A fixed number of objects (120) was used in all experiments.	The number of objects varies between 50 and 500.
The bin capacity was fixed to 150.	Bin capacity varies between 1000 and 100000.
Object sizes vary between 20 and 100.	Object sizes vary between different ranges of values, depending on the experiment.

In the first class here (BPP_{11}) we have d_i values uniformly generated with maximal deviation of 20% of the average object size. Training set consists in 4 instances and the test set consists in 36 instances.

In the second class (BPP_{12}) we have d_i values uniformly generated with maximal deviation of 50% of the average object size. Training set consists in 4 instances and the test set consists in 36 instances.

In the third class here (BPP_{13}) we have d_i values uniformly generated with maximal deviation of 90% of the average object size. Training set consists in 4 instances and the test set consists in 36 instances. For this class the BFD heuristic performs very well (for training set it obtains with only one bin more than the optimal result (287 bins instead of 286)). In this case it is quite difficult for our heuristic to find better solutions. This is why we have not applied our algorithm to this class of problems.

The second set (BPP_2) of 10 difficult instances has the following properties: $K = 100000$, $n = 200$ and d_i are randomly generated between 20000 and 35000.

6.2 Evolving an Heuristic

Parameters of the GP algorithm, used for evolving an heuristic function f, are given in Table 2.

First we have evolved an heuristic function for class BPP_{11}. In this class 4 instances were used for training purposes and 36 for testing purposes. Twenty runs having different seeds have been performed. In all cases a better heuristic than BFD has been obtained for the test set. The best heuristic generates 2633 bins for test set and 293 bins for the training set. By applying BFD on the same data we obtain 2852 bin for the test set and 316 bins for the training set. This means that we have obtained an improvement with 219 bins for the test set.

We have also applied the evolved heuristic for problems belonging to other classes (BPP_{12}, BPP_{13}, BPP_2), but we have not been able to obtain better results than BFD.

Secondly we have evolved an heuristic function for the class BPP_{12}. In this class 4 instances were used for training purposes and 36 for testing purposes. Twenty runs having different seeds have been performed. In all cases we have been able to find a heuristic which performs better than BFD for the test set. The best heuristic generates 2671 bins for test set and 294 bins for the training set. By applying BFD on the same data we obtain 2675 bins for the test set and 299 bins for the training set. This means that we have obtained an improvement with 4 bins for the test set.

It seems that increasing the range where size of objects lies, also increases the quality of solutions obtained by the BFD heuristic. In this case the solutions obtained by BFD are closer to the optimal values. This means that our procedure has smaller chances to discover a better heuristic.

Thirdly we have evolved a heuristic for the second set of instances (BPP_2). These instances are more difficult than the first ones because only 2 up to 5 objects can enter in a bin.

The first 5 instances were used for training purposes and the other 5 instances were used for testing purposes. Twenty runs have been performed.

One of the best evolved heuristic has the following performance: fitness for training set is 285. At the end of the search process we have applied the best individual on the test set and we have obtained 286 bins. By comparison, the Best Fit Descending algorithm was able to generate 298 bins for the training set and 298 bins for the test set. Thus, for the test set, our heuristic was able to obtain an improvement of 12 bins.

7. CONCLUSIONS AND FURTHER WORK

Genetic Programming has been used for evolving heuristics for the Bin Packing problem. Several different heuristics have been designed, each of them being able to perform better than Best Fit Descending for several cases. However, no heuristic was able to perform better than BFD for all cases. This is why it is recommended to combine the evolved heuristics in order to obtain a better one. For achieving this each of the evolved heuristics was run separately. For each heuristics we obtain a solution of the considered BPP instance. The solution with the minimal number of bins is finally output.

Table 2: General parameters of the GP algorithm

Parameter	Value
Population size	500
Number of generations	51
Mutation probability	0.01
Crossover probability	0.9
Selection	Binary Tournament
Terminal set	9 terminals (see section 5.2)
Function set	$F = \{+, -, *, \%, max, min\}$
Initial maximum GP tree height	4

Since the BPP is NP-Complete we cannot realistically expect that an absolutely correct evolved expression has polynomial complexity. Although the supposition that evolution could overcome the exponential complexity problem is very exciting, we cannot expect that this direction could lead to important theoretical results. Searching for heuristics that give near optimal solutions seems to be a more realistic approach.

Moreover that taking No Free Lunch (NFL) theorems [6, 7] we cannot expect to be able to design an heuristic that performs perfectly for all test cases. All we can do is to design optimal or near-optimal algorithms for some particular problems, without any guarantee related to their generalization ability on new and not seen test problems.

Further efforts will be focused on:

- Evolving strategies starting from others heuristics,
- Using an extended set of terminals,
- Extending the data set by including more difficult BPP instances,
- Evolving heuristics for other NP-complete problems.

8. REFERENCES

[1] Burke E. K., Hyde M. R., Kendall G., Evolving Bin Packing Heuristics with Genetic Programming, *Parallel Problem Solving from Nature - PPSN IX, LNCS*, Vol. 4193, pp. 860-869, Springer-Verlag, 2006

[2] Garey, M.R., Johnson, D.S., *Computers and Intractability: A Guide to NP–Completeness*, Freeman & Co, San Francisco, CA, 1979.

[3] Koza, J. R.: Genetic Programming, *On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, 1992.

[4] Martello, S., Toth, P., *Knapsack problems.* Wiley, Chichester, 1990.

[5] Scholl, A., Klein, R., Jurgens. C., BISON: a fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers & Operations Research,* 24, 627-645, 1997.

[6] Wolpert, D. H., McReady, W. G. No Free Lunch Theorems for Search, Technical Report SFI-TR-05-010, Santa Fe Institute, USA, 1995.

[7] Wolpert, D. H., McReady, W. G., No Free Lunch Theorems for Optimization. *IEEE Transaction on Evolutionary Computation*, 1:67-82, IEEE Press, NY, USA, 1997.