Offline One-Dimenstional Bin-Packing with a Hybrid Grouping Genetic Algorithm

Jason van Hattum
u15027458
u15027458@tuks.co.za
University of Pretoria

***Abstract*—Genetic programming is a technique where some algorithm, program or process is evolved in order to better solve a specific problem, such as bin packing. In the bin packing problem, objects of different sizes must be packed into a finite amount of bins in such a way that the number of bins used is minimised. The genetic algorithm (or GA) is a heuristic modelled from natural evolution, and has been shown to be particularly proficient at solving the bin packing problem. In this paper genetic programming is used to evolve the objective function used by the genetic algorithm in an attempt to improve it's bin-packing ability.**

I. Introduction

Genetic algorithms is a field of computational intelligence modelled after the process of evolution. In a typical genetic algorithm, the characteristics of candidate solutions are described by genotypes. Genetic algorithms are primarily made up of two main operators - selection and crossover. The selection operator simulated the mechanism of survival of the fittest, while the crossover operator mimics reproduction and recombination. In addition, the mutation operator is sometimes used in order to increase diversity and variation in the solution population. Candidate solutions are represented as *chromosomes*, with solution attributes making up the *genes* of the chromosome.

Genetic programming (GP) takes these principles and applies them to the evolution of algorithms. In this model, a chromosome denotes an algorithm or method to solve the problem. Chromosomes are made up of a tree, composed of two types of node - terminals and operators. Operators describe the functions that can be performed in the algorithm, and terminals represent the values that are operated on.

The problem undertaken in this assignment is to use genetic programming to evolve an objective function which is used by a GA variant to solve the bin packing problem. The idea is to treat the chromosomes of the genetic programming algorithm as the fitness function for the bin-packing genetic algorithm, and evaluate their performance and effectiveness at solving the bin-packing problem.

The genetic algorithm that will be used in conjunction with the evolved fitness function is the Hybrid Grouping Genetic Algorithm (HGGA), proposed by Falkenauer in [1]. The implementation of this is discussed in more detail in [2].

This report will be structured as follows:

- Section III will thoroughly discuss the implementation details of the GP algorithm, as well as outline some facets of the HGGA.
- Section IV will present the results, as well as some analysis and insight regarding the results obtained.
- Section V will conclude the report with a conclusion.

II. Background

This section will illustrate some background of genetic programming algorithms.

A. Chromosome Representation

As briefly outlined above, chromosomes in a genetic programming algorithm are implemented as tree structures, consisting of *terminal* and *function* nodes.

1) Terminal Nodes: Terminal nodes are leaf nodes, and embody values - typically constant values and/or variables.

2) Function Nodes: Function nodes are internal or branch nodes, and represent operations to be performed on terminal nodes. Function nodes have an arity, which is the number of parameters the function requires (or alternatively the number of children of the function node).

These trees are able to be evaluated or executed (depending on the problem), which is typically some recursive traversal of the tree.

B. Grammar

A grammar needs to be defined that can be used to express a solution. These grammars are highly problem-dependent, and need to be able to represent any possible solution to the problem. The grammar consists of a function set and a terminal set, each representing the possible values for their corresponding nodes. There may also be a set of rules called the semantic set that define the possible combinations of the function and terminal sets.

C. Initial Population Generation

Initial population generation is restricted by three factors:

- The maximum depth of the trees.
- The semantic set.

- The desired population size.

For each non-root node, the node is initialized to a randomly selected element from the terminal and function sets. If the node is a function, a set of children are randomly selected for that node up to the arity of the function.

D. Fitness Function

The simplest form of a genetic programming fitness function tests the evolved program against a suite of test cases and comparing the results. More complex forms can test the runtime of the program or penalise semantically incorrect structures.

E. Operators

The operators for genetic programming are very similar to genetic algorithms - selection, crossover and mutation.

1) Selection: Any selection method used for genetic algorithms can be used for genetic algorithms. They are functionally identical.

2) Crossover: The crossover operator in genetic programming is done by selecting a random node from each parent and swapping them, generating an offspring from each modified parent. Another option is to simply return a single offspring by replacing only one parent's node with a randomly selected node from the other parent.

3) Mutation: Much like in genetic algorithms, mutation in genetic programming is done by selecting a random chromosome (that is, a tree) and changing it in some form. This can be done through a variety of methods, such as removing nodes, generating random subtrees or changing constant values.

III. Implementation

The algorithm implemented was a standard genetic programming algorithm, wrapping the HGGA as described above. Several key parts of genetic programming will be described.

A. Chromosome Representation

The chromosomes in the implemented GP algorithm were implemented as parse trees, where the operators are functions and the terminals are values (as is standard in GP algorithms).

The terminal and function sets are given in Tables I and II respectively. θ is the average bin fitness, defined as:

$$\theta = \frac{\sum_{i=1}^{N}(F_i/C)}{N}$$

B. Initial Population Generation

The population generation method used was *Ramped Half-and-Half*, which produced a set of trees in varying sizes and shapes. This combines two common methods of initial population generation - grow and full. The full method generates trees of maximum depth, while the grow method generates trees up to a maximum depth.

In ramped half-and-half, trees are generated for every depth value from 2 to the maximum depth. For each of these depths, half the generated trees use the grow method and half use the full method. This helps to produce a diverse population.

Each terminal node and function node has an equal chance of being selected.

C. Selection

The selection strategy used is a simple tournament selection with a tournament size of 2, where half the population was selected to be parents and the other half discarded.

D. Crossover

Crossover occurs between two parents (bisexual), and generates two offspring. Crossover occurs exactly as described in Section II.E(2) - that is, a random subtree is selected from both parents and swapped.

E. Mutation

Several mutation operators were implemented and used:

- **Function node mutation**: A random internal node is selected and replaced with another function of the same arity.
- **Swap mutation**: A randomly selected subtree is swapped with a randomly generated subtree.
- **Grow mutation**: When a terminal node is replaced with a randomly generated subtree.

The mutation probability was 33%, and each mutation operator has a 33% chance of being applied.

F. Fitness Function

The chromosomes were evaluated by running a simulation HGGA with the objective functions evolved by the GP. This was done for a number of test cases where the optimal number of bins was known. The fitness of the objective functions were computed as the average difference between the known optimum and the result from running the HGGA with the objective function - that is:

$$f = \frac{\sum_{i=0}^{N} f_i}{N}$$

where N is the number of test cases, i is the ith test case and f_i is the fitness obtained by running the GA with the ith test case.

G. Parameters

The GP algorithm was tested with the following parameters:

- 100 iterations
- An initial maximum tree depth of 10 (it could expand infinitely during crossover/mutation).

- A population size of 18 (the strange number is due to the constraints brought by the ramped half-and-half method of initial population generation).
- A tournament size of 2 in selection.
- A 33% chance of a chromosome, and in that mutation each mutation operator had a 33% chance of being applied.
- A training set of 4 test cases (2 from the easy data set, one medium and one hard).
- A testing set of 2 test cases (1 easy, one hard).

IV. Results and Analysis

A. Datasets Used

The datasets used were from the Schalk dataset [3]. The dataset is split into easy, medium and hard sets, and the optimal number of bins is given for each dataset.

Two easy sets, one medium set and one hard set were used to train the GP. The imbalance towards the easy sets was due to the difference in execution time between the easy and hard sets - hard sets took much longer to complete.

One easy and one hard set were used to test the generalisation of the GP. Only two were used due to time constraints.

These datasets are given in Table III, along with their optimal number of bins.

The results from the study are tabled in Table IV. They are acceptable, with only the hard data sets not performing perfectly.

V. Conclusions

References

[1] E. Falkenauer, "A hybrid grouping genetic algorithm for bin packing," *Journal of Heuristics*, vol. 2, pp. 5–30, Jun 1996.

[2] V. Hattum, "Offline one-dimensional bin-packing with a hybrid grouping genetic algorithm," *COS710 Assignment 3*, 2018.

[3] "Scholl bin packing benchmark datasets." https://www2.wiwi.uni-jena.de/Entscheidung/binpp/index.htm. Accessed: 2018-05-23.

TABLE I
TERMINAL SET

Symbol	Description
B	Number of Bins
C	Bin Capacity
θ	Average Bin Fullness
-	Constant Values

TABLE II
FUNCTION SET

Symbol	Description	Arity
+	Addition	2
-	Subtraction	2
*	Multiplication	2
/	Division	2

TABLE III
DATASETS USED

Name	Number of Items	Bin Capacity	Item Size Variance	Optimal Number of Bins

TABLE IV
BEST RESULT, AVERAGE RESULT AND STANDARD DEVIATION OVER 10 RUNS OF EACH DATASET

Datasets	Best (Bins)	Best (Fitness)	Average (Bins)	Average (Fitness)	Deviation (Bins)	Deviation (Fitness)