# OSU-GP: Attribute Selection using Genetic Programming

**Josh King**
Department of Computer Science and Engineering
The Ohio State University
Columbus, OH 43210, USA
kingjo@cse.ohio-state.edu

## Abstract

This system's approach to the attribute selection task was to use a genetic programming algorithm to search for a solution to the task. The evolved programs for the furniture and people domain exhibit quite naive behavior, and the DICE and MASI scores on the training sets reflect the poor humanlikeness of the programs.

| | |
|---|---|
| valueIn | {variables} |
| target | sequence |
| distractors | canUseLoc |
| eqA | ifte |
| set_empty | for |
| set_add | and, or, not |
| set_remove | {attributeERC} |
| add | {attributeListERC} |
| remove | |

Table 1: Functions supplied to the GP algorithm

## 1 Genetic Programming

Genetic programming is a form of evolutionary computing in which a meta program evolves another program to solve a problem. Creating by hand a program which solves the problem may be possible, but if the problem has many parameters which contribute to a solution's quality, the program designer may miss some subtle interplay in favor of an expected solution. Genetic programming evolves a pool of programs to optimize a user supplied fitness function which gives some indication of how well a program performs on the problem.

This report in no way attempt to fully explain genetic programming. See, *e.g.*, (Koza, 1992) for a better understanding of genetic programming. The evolutionary computation toolkit ECJ available from `http://www.cs.gmu.edu/~eclab/projects/ecj/` was used for the genetic programming algorithm.

## 2 Functions

The genetic programming algoritm had at its disposal the functions listed in Table 1. The function eqA tests whether two attribute *values* are equal. The functions add and remove operate on attribute lists, whereas the similarly named set_add and set_remove operate on entity sets. The function ifte is an if-then-else construct, and the function for is a foreach operator on sets of entities or lists of attributes and requires both a variable to change the value of (the iterating variable) and the set or list to operate over. The functions surrounded by braces represent various things: for {variables}, various variables of the different types were globally accessible. {attributeERC} and {attributeListERC} represent random variables that take on attributes from the trial set and lists of those attributes. A mutation of an ERC (ephemeral random constant) simply sets it to a different random value from the random constant's domain.

The functions supplied to the genetic programming method were more than expressive enough to formulate, *e.g.*, Dale and Reiter's Incremental Algorithm (Dale and Reiter, 1995), a well known standard algorithm for doing this task. During the run of the genetic programming algorithm, every function shown in Table 1 was explored in the search, even though many of them are not used in the solutions found.

| | |
|---|---|
| diceDist | locationUse |
| masiDist | inappropriate |
| noHit | length |
| bloat | empty |
| similar | excessive |

Table 2: The costs GP should minimize

## 3 Fitness Function

The genetic programming algorithm must be supplied with a fitness function which judges how well a program performs on the problem. For this task of attribute selection, there are multiple dimensions to optimize in. For simplicity, a linear combination of the various dimensions was used for the fitness function. The dimensions to optimize on were represented as costs or penalties that the genetic programming algorithm sought to minimize. The various costs are listed in Table 2. The costs were devised with an eye towards optimizing quantities that existing algorithms for this task also attempt to optimize.

DICE and MASI scores on the training data were converted to costs by subtracting from one. The noHit cost is for not forming a distinguishing collection of attributes and was given a high weight. The bloat penalty penalizes long programs (to make interpreting the result easier).

The fitness function is a weighted sum of these costs. A fitness of zero would mean that each dimension is optimized fully. This is not possible, as some of the costs conflict with each other: for example, length and empty. The weights for the fitness function were hand tuned, which is counter to the desire to use genetic programming rather than hand tune an algorithm. The use of evolutionary computing to tune these weights may produce better results, but was not tried here.

## 4 Resulting System

The program which evolved for the training trials of the furniture domain is shown here:

```
(ifte canUseLoc
 (var_aList_set al0
  (add [ orientation size ]
      colour))
 (var_aList_set al0
  (add (add [ orientation size ]
      colour) type)))
```

The program found for the people domain is shown here:

```
(ifte canUseLoc
 (var_aList_set al0 (add
  (add [ hairColour hasGlasses ]
      hasBeard) hasSuit))
 (var_aList_set al0
  (add [ hairColour hasGlasses ]
      hasBeard)))
```

Both of these programs seem quite unintelligent in their behavior. They only ever set the attribute list to one of two lists, depending only on whether the location condition is flagged or not, and don't even make use of the location information. Further tweaking of the fitness function or using different schemes for evolution may improve the performance of an evolved solution.

The performance metrics of DICE score and MASI score on the training set are shown in Table 3. The accuracy of the realization[1] of the attributes is both zero. The scores reflect the obvious point that the evolved programs have plenty of room for improvement. A fitness function which is less reliant on the user to specify weightings for optimization may be necessary, or an evaluation which tests the program's ability to generalize may catalyze program evolution into less naive directions than those programs shown here.

| | DICE | MASI |
|---|---|---|
| Furniture | 0.61 | 0.30 |
| People | 0.35 | 0.10 |
| Average | 0.49 | 0.21 |

Table 3: DICE and MASI scores on training set

## References

Robert Dale and Ehud Reiter. 1995. Computational interpretations of the gricean maxims in the generation of referring expressions. *Cognitive Science*, 19(2):233–263.

John R. Koza. 1992. *Genetic Programming: on the Programming of Computers by Means of Natural Selection*. The MIT Press.

[1]The realizer was the simple template-based realizer written by Irene Langkilde-Geary, Brighton University for the ASGRE 2007 challenge.