Automatic Program Generation Based on the Swarm

Tim Charnecki[1] Donald Cooley[2]

[1]Computer Science, Utah State University, timc@cc.usu.edu
[2]Computer Science, Utah State University, don.cooley@usu.edu

Abstract

This paper describes and compares two implementations of a particle swarm optimization (PSO) approach to a symbolic regression problem (a subset of genetic programming). In these two techniques, particle encoding is based on multi-expression programming and Cartesian genetic programming chromosome encoding. It allows for the representation of both real and discrete values. The first PSO approach called pseudo swarm crossover uses a crossover based particle update function, while the second approach, Cartesian swarm programming, uses the standard PSO distance based update function modified for discrete variables. These proposed techniques are then compared with established genetic programming techniques.

Keywords: Particle swarm optimization, genetic programming, genetic algorithm Cartesian genetic programming, symbolic regression.

1. Introduction

Charles Darwin's Theory of Evolution inspired the branch of AI known as evolutionary computation (EC). Within this field are two important branches; namely, genetic programming (GP) [5] and particle swarm optimization (PSO) [3,4]. In this research we have designed and implemented swarm techniques that can be applied to automatic program generation. PSO served as the model of swarm intelligence applied to GP problems. The specific problem tested in this work was symbolic regression.

Genetic programming borrows from many of the techniques implemented in genetic algorithms (GA). However, genetic programming applies an evolutionary approach to computer program creation. John Koza proposed the first implementations of genetic programming [5] using a tree structure to represent computer programs. His programs were implemented in LISP. Today, GP has been implemented in many different computer languages with a variety of encoding schemes. Cartesian genetic programming and multi-expression programming are examples of such encoding schemes.

1.1. Cartesian GP

Cartesian genetic programming (CGP) was proposed by Miller and Thompson [6] as an alternate approach to the breeding of computer programs. CGP programs are directed acyclic graphs with nodes containing an operation, and pointers to other nodes, that are operands. The CGP chromosome is a string of integers sequentially containing each node's operation id and node pointers.

With linear, integer encoded chromosomes, popular GA-style crossover and mutation operators can be used. Miller's experiments in [6] used evolutionary algorithm techniques to evolve solutions to a symbolic regression problem and the Santa Fe ant trail problem.

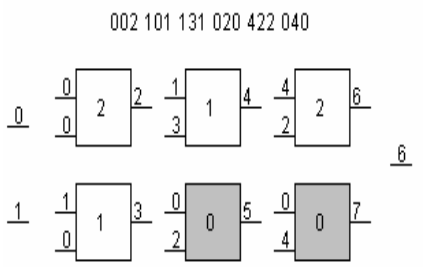

Fig. 1: CGP Chromosome encoding. Genotype (top) - Phenotype (bottom) mapping for a program with 2 inputs, 1 output, and three functions (0, 1, 2). The gray nodes are unconnected.

1.2. Multi-Expression Programming

Multi-expression programming [7] (MEP) is another GP technique. MEP is unique in terms of the encoding of the chromosome. The chromosome is arranged in a fixed number of genes, organized linearly. Each gene can represent either a terminal or an operation. Operator genes contain the operator and two operands, which are references to other genes in the sequence. It is important that all operator genes only operate on genes located previous in the sequence. This avoids cycles resulting in illegal programs. Another encoding rule is that the first gene must always be a terminal.

Different chromosome representations are shown in Figure 2.

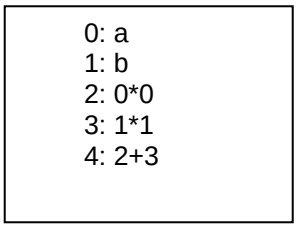

Fig. 2. MEP scheme. This encoding represents the function $a^2 + b^2$.

MEP chromosomes are evaluated from the top down, assessing the expression at each line. Since each chromosome can represent multiple expressions, it is called "multi". So, in Figure 2, the expressions a^2 and b^2 are also represented in the chromosome and can be considered for the final solution. The fitness can be determined by the set of all expressions or by the single best expression in the chromosome. Like CGP, MEP allows for a linear encoding of the chromosome. MEP can encode complex equations in fewer expressions than GEP. This can be beneficial in terms of performance and memory usage [7].

Much like introns in eukaryotes, at generation n, CGP and MEP chromosomes may contain genes that are not part of the encoded solution program. These unused genes are still important, as they could become part of a better solution in a future generation. MEP chromosomes can actually be mapped to a CGP chromosomes with a single row of nodes if CGP nodes are enabled to return terminals.

1.3. Swarm Intelligence

Swarm intelligence is a branch of artificial intelligence inspired by the organizational behavior of groups of insects or birds. Simple rules can be applied to primitive individuals to achieve more complex behaviors, suggesting the whole is greater than the sum of its parts. Swarm techniques have been applied to solve problems, such as evolving neural networks [4], machine scheduling, and robot coordination. A subcategory of swarm intelligence based on the idea that a group of individuals can work together to find an optimal position by sharing knowledge is known as particle swarm optimization.

Particle swarm optimization (PSO) was introduced by James Kennedy and Russell Eberhart in 1995. It attempts to solve problems using a swarm intelligence dichotomy [3]. Like GA's the algorithm begins with a population of random solutions. In PSO, each member is known as a particle and has a position and velocity in each dimension of the parameter or search space. Iterating through the algorithm, the particles fly around in the parameter space looking for the best solution.

At each time step, each particle's position in the search space is updated based on the position of that particle's best position to date and the position of the global best. Early experiments successfully applied PSO to such problems as nonlinear function optimization [10], and neural network training .

1.4. Discrete PSO

In our PSO implementation of automatic program generation, an important consideration was particle encoding. PSO is usually encoded as a vector of real values, making position and velocity intuitive, and, thus, distance is a simple Euclidean-based calculation. If particles are represented as vectors of discrete values, distance and velocity must be handled differently. Fukuyama, et al. used a mixed approach to optimize a mixed-integer nonlinear optimization problem regarding reactive power and voltage control in electric power systems [2]. Their approach assumed that discrete variables could be treated the same as continuous variables, as long as the velocity and position parameters were discretized and kept within an acceptable set of values. Their approach was also implemented in this work.

2. Pseudo Swarm Crossover Based on MEP

The first proposed technique, which we call pseudo swarm crossover, is based on the MEP concept of swapping program lines between chromosomes, and thus, it can be applied to any GA.

Each particle knows the subroutines and functions or its most successful program in the past, and the subroutines and functions of the most successful member of its group. Therefore, by receiving subroutines from the particle's best program and the dominant program of the population, each particle will try to behave more successfully. This is attempted while maintaining some of its own unique subroutines and functions (genetic composition) to ensure diversity, and hopefully find an even better solution.

3. Cartesian Swarm Programming

In CGP, an important step in being able to define a program's position and velocity is to quantify all operations and terminals in a program. All operators and operands in the chromosome are represented by a discrete or real value. These values represent a position in an n-dimension space. All particle

positions have a corresponding real valued velocity. This proposed method uses the same population approach as traditional PSO.

In CGP, like GP or PSO, initially a population of random solutions must be generated. The only modification made to the standard PSO algorithm is the ability to handle discrete positions. It is similar to the algorithm used in [2]. After the velocity and position of the discrete variable are calculated using the standard PSO update function, the position is discretized to an acceptable value by applying the floor function to the real valued position.

Particles can be thought of as having position in a multidimensional solution space. Using these integer distances, particles will explore the intermediate states between the present state and most successful state. This provides good exploration properties for the particles. With a scheme to provide quantifiable values for position and velocity, one can use the same techniques as standard PSO techniques.

4. Symbolic Regression

The specific problem in genetic programming tested was symbolic regression. This is the problem of finding a function or equation from a set of operators that best matches a set of data points. Symbolic regression was tested in the initial GP experiments performed by Koza [5]. It was also used to test the effectiveness of CGP in [6] and MEP in [7]. Both test functions used in these original experiments were implemented along with three others.

4.1. Regression Functions

A variety of test functions has been chosen. Some functions were selected as benchmark functions that have been featured in other papers, and others provide unique characteristics that will be analyzed in the results. The test functions and reasons for their use are listed in Table 1.

Name	Function
function1	$f(x) = 4x^4 - 3x$
function2	$f(x) = x^4 + x^3 + x^2 + x$
function3	$f(x) = \pi r^2$
function4	$f(x) = x^6 - 2x^4 + x^2$
function5	$f(x) = \sin(\pi x)$

Table 1. Regression Functions

4.2. Implementation

In each experiment, a set of fifty X values was randomly generated over a real valued range [-1.0, 1.0]. Corresponding Y values were then calculated using the associated test function. The resulting x,y pairs were used to determine the fitness of the equations generated by each approach. The fitness was defined as the sum over all test values of the absolute difference between the output of the test equation and the output of the program generated equation. A solution is considered acceptable when the fitness value is less than an accepted error value E.

Identical chromosome/particle structure was used for all tests. Each particle in the swarm was composed of a fixed number of indexed lines equal to PARTICLE_LENGTH. Each line contains four values: one operator index value (integer), two operand index values (integer) and one real value. Because all operators are binary operators, only two operand parameters are required. Operand index 0 is always used for the first line, and following lines use operator indices 1-5.

Equations were formed from the following set of operators: addition (+), subtraction (-), multiplication (*), and division (/). For the division operator, if the program attempts to divide by zero, the numerator is returned. Terminals can be either input value X or a constant real value [-5.0, 5.0]. This range for real values is very important to the performance of the optimization, especially when the solution is dependant on one or more real values with a small range of acceptable values. This range was used for all experiments. A sample particle or chromosome is shown in Figure 4.2.

0	0	-	-	-	X
1	3	0	0	2.11	x * x
2	1	0	1	0.23	x + 2.11
3	5	2	1	3.14	3.14
4	3	3	1	-1.05	3.14 * x * x

Fig. 3. Sample PSC/CSP particle representing πr^2. The result of line 4 is returned for results.

5. Optimization Techniques Tested

The four techniques described in this section were tested on each of the functions in Table 1. Chromosome encodings for the genetic techniques are identical to the PSC/CSP particle encoding. The selection and crossover functions chosen for MEP and CGP are similar to those used by the original authors, and preliminary experiments were performed to verify that MEP and CGP would perform similarly to that of the original experiments. The mutation rate used is similar to that in [7]. Greedy parent selection was done by sorting the population by fitness. A parent is chosen randomly from the more fit half 80 percent of the time and from the less fit half 20 percent of the

time. Tournament parent selection is when two members are chosen at random from the population. Their fitness levels are compared, and the more fit member will be selected for parenthood 70 percent of the time. Tournament selection for next generation is when two members are chosen at random from the population of offspring and parents. Their fitness levels are compared, and the more fit member will be selected for the next generation 70 percent of the time. For CSP, parameters were chosen based on successful PSO tests done in [1].

6. Results and Conclusions

The quality of the optimization techniques was influenced by the difficulty of the test functions. It is important to note that similar solutions were found by all of the optimization techniques.

This paper provides a comparison of the relative success rates of the proposed optimization techniques as well as an evaluation of the effect of population size. From the results, we found that there is not one dominant or best optimization technique for all cases. Performance is somewhat consistent between sets, but it varies drastically between functions. Because of the success variations, it can be concluded that the ideal optimization technique used is problem specific.

	Strong (*functionAvg*>=90)	Moderate (90>*functionAvg*>=75)	Weak (*functionAvg*<75)
function1	CGP	CSP, PSC ,MEP	-
function2	MEP	CSP,PSC,MEP	-
function3	CSP	CGP,MEP	PSC
function4	CSP	CGP,PSC	MEP
function5	MEP,PSC,CGP[a]	-	CSP,CGP[a]

a. CGP performance was improved when next generation selection was changed to *sort all and save best*.

Table 2. Relative success rates

PSC does not perform any better than the other genetic techniques, CGP and MEP, but is consistently competitive. The success of this supports the idea that cognitive and social influences are beneficial to the search procedure

From the results for *function3* and *function4*, the swarm influenced method of automatic program generation is promising for simple functions, but the results for *function5* are discouraging. CSP could be a recommended optimization technique for finding simple programs when real valued parameters are required. However, further experimentation should be performed on a wider variety of problems to determine the robustness of CSP.

7. References

[1] Carlisle, A. and Dozier, G. An off-the-shelf PSO. In *Proc. of the 2001 Workshop on Particle Swarm Optimization*, 2001.

[2] Fukuyama, Y., Takayama, S., Nakanishi, Y. and Yoshida, H. A Particle Swarm Optimization for Reactive Power and Voltage Control in Electric Power Systems. in *Proceedings of the Genetic and Evolutionary Computation Conference*, (1999).

[3] Kennedy, J. and Eberhart, R. Particle swarm optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, 1995.

[4] Kennedy, J., Eberhart, R. and Shi, Y. *Swarm Intelligence*. Morgan Kaufmann Publishers, San Francisco, 2001.

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

[6] Miller, J. and Thomson, P. Cartesian genetic programming. In *Genetic Programming, Proceedings*, 2002. 121-132.

[7] Oltean, M. and Dumitrescu, D. Multi Expression Programming. http://www.mep.cs.ubbcluj.ro/oltean_pdf.pdf 2002.