Co-Evolutionary Automatically Defined Functions in Genetic Programming

ANTHONY LUKAS, FRANZ OPPACHER
School of Computer Science
Carleton University
1233 Colonel By Drive
CANADA
alukas2@connect.carleton.ca, franzo1@mac.com

Abstract: - We show how the addition of co-evolution to genetic programming (GP) overcomes the current limitations of GP as well as GP augmented with automatically defined functions (GP+ADF) with a method called co-evolutionary automatically defined functions (GP+CADF). We demonstrate that GP+CADF requires a lower computational effort to solve the parity, sum of bits, image recognition, lawn coverage and the bumblebee problems. To further improve GP+CADF, we discover that using elitism lowers the computational effort required. We also discover ways to improve the initial population and initial best individuals used for evaluation.

Key-Words: - Genetic programming, co-evolution, automatically defined functions

1 Introduction

Drawing inspiration from the concept of evolution has proven to be successful in solving challenging problems with genetic programming (GP). The addition of functions or subroutines to GP is an idea that allows a program to reuse modular functionality. It has been shown that when genetic programming is augmented with automatically defined functions (GP+ADF) we can solve problems beyond the capability of standard GP [1]. This is due to the fact that ADFs exploit underlying regularities and repetitions in the solutions of a problem. However, even with the addition of ADFs some problems eventually become too hard to solve. There is a certain threshold of difficulty where the GP algorithms require too much time and computation to yield an answer in any reasonable amount of time. In other words, the problem is how to keep discovering solutions where GP+ADF is unable to do so. This paper will explore how we can answer this challenge with the introduction of co-evolution to genetic programming where we devise multiple populations to work together toward solving a problem.

We do so with a method called co-evolutionary automatically defined functions in genetic programming, abbreviated as GP+CADF. The value of the contribution is that GP+CADF requires less computational effort to discover a solution than standard GP or GP+ADF. A comparative analysis of standard GP, GP+ADF and GP+CADF is performed on a wide variety of different problems that include up to 14-even-parity, sum of bits, image recognition, lawn coverage and the bumblebee problems. In addition, methods to improve GP+CADF are discovered which include improvements of initial populations, improvements in best individuals used for evaluation, effect of elitism and effect of multiple best individuals.

2 Description of Method

The following sections will review automatically defined functions and what their drawbacks may be. Following this, the GP+CADF algorithm will be introduced and we will describe in detail how fitness evaluation is performed in this multi-population system. Lastly, several techniques will be proposed as potential improvements for GP+CADF. We will also review earlier work in co-evolutionary GP.

2.1 Problems with ADFs

Genetic programming with automatically defined functions described by Koza in [1] is an attempt to improve GP by recognizing that many problems can be more readily solved by decomposing them into sub problems and assembling the sub problems into the final solution. An analysis by Koza of the Boolean Parity, Lawnmower, Bumblebee, Impulse Response Function, Artificial Ant, Obstacle-Avoiding Robot, Minesweeper, Letter Recognition and Transmembrane Domain Prediction in Proteins [1] among other problems has demonstrated that it is indeed easier to discover a subroutine and reuse it many times within a main body, than it is to

discover one complete tree as in the traditional GP approach. Nevertheless, there may be some problems that arise when subroutines are utilized in genetic programming.

2.1.1 Poor Main Body and Useful ADF

It may happen that a useful ADF has been discovered through evolution, but the main body calls it in the wrong place. The fitness of the individual will still be determined to be low. There is no partial credit given to the individual based on the fact that this ADF is useful. In point of fact, there is not even any way to determine if this ADF is useful because we would need to couple it with a main body that is able to exploit it to the fullest to determine its usefulness.

2.1.2 Fit Main Body and Poor ADF

It may also be the case that the main body is very fit but the ADF does not do anything useful. In this case there is no partial credit given to the good main body. In fact, without a fitting ADF, it can not even be determined how useful a main body is, assuming the main body relies on using ADFs. Another time, we see the importance of coupling ADFs and main bodies that work well together.

Other problems that may arise could be that a fit main body and a fit ADF do not fit together, ADFs are not called at all or that an ADF is passed the wrong arguments.

2.1.3 Consequences of Miss Fitting ADFs and Main Bodies

The consequence is that good parts of an individual may be lost in following generations since their worth was not recognized. We claim that the right coupling of an ADF and a main body is just as important as discovering useful subroutines. In addition, due to the architecture of the GP+ADF approach, an ADF is always constrained to a single individual. If a very fit main body or ADF is found in one individual, this discovery may not even be recognized since only one other counterpart is used to couple and evaluate it. Furthermore, these good subroutines and main bodies will not be available to all the other individuals except with crossover exchange of some parts in very few other individuals. With these considerations in mind, we can proceed to describe the co-evolutionary approach and how we will try to overcome these limitations.

2.2 Description of the GP+CADF Algorithm

The GP+CADF algorithm works with n populations $P_0 \ldots P_n$ where P0 is the main body population and $P_1..P_n$ are the ADF populations. There are as many ADF populations as the number of ADFs we choose to have in our architecture. The total number of individuals in all populations is M, and therefore the number of individuals in each population is M/n.
Figure 1 presents the basic GP+CADF algorithm:

1: Let Bi be random individual of P_i for i=0..n
2: For G generations or until solution found:
3: Update B_i to be the best individuals found so far in P_i for i=0..n
4: Evaluate each individual in P_i by coupling it with B_j where j=0..n and $j \neq i$
5: Breed each population and apply genetic operators
Fig. 1 - Basic GP+CADF algorithm

One problem that becomes apparent is that the individuals in the main body and ADF populations can not be evaluated just by themselves. We need to have all the ADFs for a main body and we also need a main body and all other ADFs for an ADF in order to couple them together and form a complete individual that can be evaluated as stated in line 4.

2.2.1 Fitness Evaluation of Individuals in the GP+CADF Approach

Suppose that we decide in our architecture that we will have two ADFs. Then, let us say we decide that there will be a total of 3 populations each with M/3 individuals. If we consider all the ways that these individuals can be coupled, there are a total of $M^3/27$ combinations. If we wanted to evaluate all of these combinations at every generation this would quickly become infeasible due to exponentially increasing evaluation times.

A solution to this problem is to couple all of the individuals in a population with just one individual Bi from all other i populations. It is immediately apparent that this choice of Bi is very important since this individual will be used in all fitness evaluations. Furthermore, the fitness will be an evaluation of how well an individual fits with all our chosen Bi. A reasonable assumption is that a good choice for Bi would be the current best individual found so far in each of the populations. This choice makes sense because the best individual has the highest fitness and does most of what is required by our problem.

2.2.2 How to Evaluate ADFs and Main Bodies

Figure 2 shows a diagram of how evaluation of ADFs is done in the co-evolutionary multi-population system. The setup for that example is that we have one main body and two ADFs. Therefore, there will be a total of 3 populations: one for the main body and one for each ADF. In the top row are the best individuals from each population. In the diagram, the best main body is a 6 node program tree and it calls both ADFs in its terminals. The best ADFs are both 3 node program trees in this example.

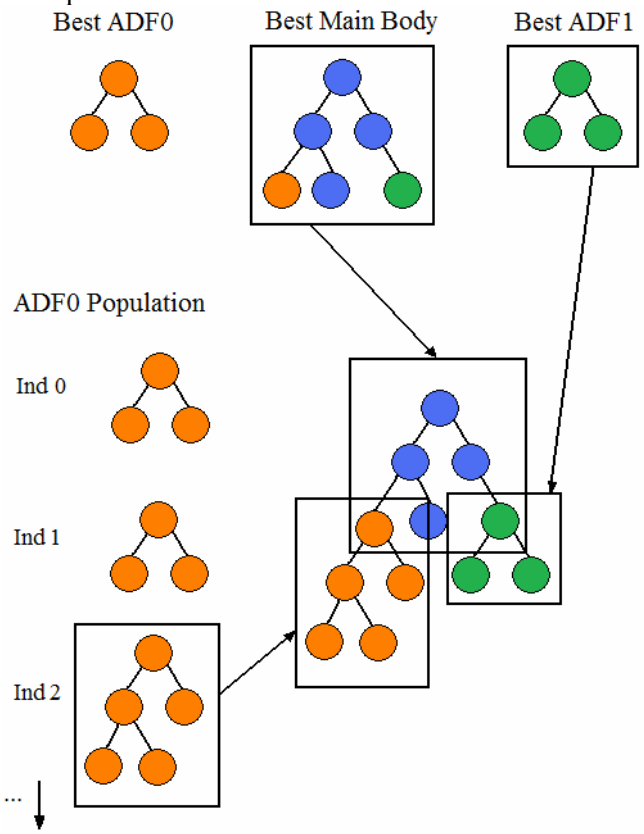

Fig. 2. Evaluating ADFs in a multi-population system.

The ADF0 population is represented as a column of individuals and for the sake of simplicity there are only the first three individuals shown. The diagram shows how individual 2 would be evaluated: it will be plugged into the best main body along with the best ADF1. This complete individual will be evaluated and the fitness will be assigned to individual 2. The same process will be repeated for all individuals in the ADF0 population. Similarly, this process will be applied to all the individuals from the ADF1 population with Best ADF0 used where ADF0 is called.

Following this method we may also evaluate all individuals in the main body population by coupling them with the best ADFs.

3 Improvements for GP+CADF

The next sections will examine potential improvements for the GP+CADF method. These include using elitism in evaluation, finding better initial best individuals and generating a better initial population.

3.1 A. Co-evolutionary Elitism

One strategy for evaluating individuals is to use the best individuals that were determined from the previous generation as our B_i. The motivation for this design decision is that the best individuals must be updated as evolution progresses in order to use better, newly evolved ADFs and main bodies and to find the best coupling between them.

On the other hand, we could also use an elitist strategy where we only keep track of the best individual found so far in the entire evolutionary run. The motivation here would be that we want to use the absolute best main body and ADF that we can find in the entire run. In addition, another hypothesis would be that this will have a stabilizing effect because the best individuals we are trying to couple and fit with all other individuals will not change as rapidly.

Experimental results with the parity problem showed that the elitist strategy was better by an order of magnitude and provided better stability by changing the best individuals less often.

3.2 Better than Random Initial Best Individuals

One way to boot start the process of evaluation is to choose the initial best individuals randomly from generation 0. The problem with this approach is that since these individuals are random, they might be particularly unfit and it might not be worthwhile to start the evolution process with these poor individuals. A different approach would be to initially try M couplings between all individuals and choose as best those individuals that produced the highest fitness when coupled together. This would amount to sacrificing one generation of evaluation in order to determine better initial best individuals.

Experimentation with the parity problem concluded that the computation effort was reduced by a factor of 1.6 when choosing the initial best individuals after M couplings.

3.3 Checking for ADF Calls

Since using ADFs and coupling main bodies and ADFs is the heart of the GP+CADF approach, then it is essential that ADFs be used by main bodies. This is because most of the computational effort in GP+CADF is spent in evaluating couplings between best individuals and populations. It will be wasted effort to couple a main body with all ADFs in an ADF population when an ADF is not even referenced.

Therefore, an improvement to the initial population would be to regenerate a main body that does not call ADFs until one is made with ADF references. Experimentation with the parity problem concluded that this strategy reduced the computational effort by a factor of 1.7.

4 Earlier Work with Co-Evolutionary GP

The literature in both competitive and co-operative co-evolution was surveyed for both GA and GP approaches. Most works dealt with competitive co-evolution of host-parasite systems. Issues that arise in competitive co-evolution such as disengagement and determination of the appropriate test set were dealt with in a number of works. [4, 5, 6].

Our GP+CADF method stands firmly in the less explored co-operative realm of co-evolution. The main challenges in co-operation are of problem decomposition and methods of individual interaction which are explored in existing works [7, 8, and 9]. The co-operative co-evolution approach we take to decomposing problems into multiple co-operating populations is with ADF subroutines in GP.

The idea of co-evolving main bodies and ADFs in GP has already been dealt with in [3] by Aler. In our approach, an n population system will be set up to have M/n individuals in each population where M is the total number of individuals desired in all populations. Aler's approach is to have M individuals in each population but run each population for G/n generations where G is the total number of generations desired. In order to compensate for the larger number of evaluations required in GP+CADF we reduce the number of individuals in each population while Aler reduces the number of generations that each population is run. In all approaches: this, Aler's and GP+ADF, there are the same number of evaluations per generation performed.

Consequently, Aler experiments with a small population and long runs, which means that M is 200 and 400, and G is 150. The large number of generations used in his setup also compensates for the design decision that each population is run for G/n generations.

In addition, while Aler only experiments with 5 and 6-parity we perform a comparative analysis on a wide range of problems. Lastly, we implement and test a number of improvements for the GP+CADF method as outlined in the previous sections.

Experimental results with the parity problem showed that the elitist strategy was better by an order of magnitude and provided better stability by changing the best individuals less often.

5 Experimental Results

In order to demonstrate that the GP+CADF method has an advantage over standard GP and GP+ADF, we tested a wide variety of problems that include even-n-parity, sum of bits, image recognition, lawn coverage and the bumblebee problem. A comparative analysis between the methods was performed for each problem that utilized the metric of computational effort for comparison.

5.1 The Even-n-Parity Problem

The even-n Boolean parity problem is the task of recognizing whether a bit string of length n consisting of 1's and 0's contains an even number of 1's. We see that if even just one bit is changed in the bit string, then the output changes. Additionally, in order to find the output we need to consider all of the inputs. For that reason, the parity function is hard to learn for machine learning algorithms as discussed in [2]. Furthermore, Koza reports that not a single program solved the even-3-parity function in a blind random search of 10,000,000 programs that work with 3 input bits and the Boolean functions AND, OR, NAND, NOR [1].

Table 2 shows the minimum computational efforts obtained in the experimental comparison of standard GP, GP+ADF and GP+CADF for the even-n-parity problem. There were no solutions obtained for standard GP with 300 runs past even-6-parity, and GP+CADF was the only method to yield solutions after 300 trials past even-10-parity.

The immediate conclusion is that as we increase n, the computational effort increases most quickly for regular GP followed by GP+ADF and GP+CADF. Correspondingly, there is always less computational effort required to solve the even-n-parity problem with GP+CADF than with GP+ADF.

Problem	Even n-parity
Number of bits (n)	3 to 14
Population Size (M)	1800 for n<=13, 3800 for n=14
Generations (G)	60
Runs	300 for n > 6, 500 for n <=6
Number of ADFs	2
Function Set (no ADF)	D0 to D13, AND, NAND, NOR, OR
Function Set (with ADF) Main Body	D0 to D13, AND, NAND, NOR, OR, ADF0, ADF1
Function Set for ADF0	AND, OR, NAND, NOR, ARG0, ARG1
Function Set for ADF1	AND, OR, NAND, NOR, ARG0, ARG1, ARG2
Fitness Cases	All 2^n possible bit strings
Fitness	2^n – correct outputs
Success Predicate	0 (all fitness cases must be perfect for 0 fitness)

Table 1 – Experimental setup for the even-n-parity problem.

n	Runs	GP	GP+ADF	GP+CADF
3	500	14,400	75,600	45,000
4	500	234,000	182,700	126,000
5	500	No solution	553,500	302,400
6	500	219,515,400	1,357,200	446,400
7	300	No solution	2,754,000	693,000
8	300	No solution	4,050,000	1,044,000
9	300	No solution	10,054,800	1,461,600
10	300	No solution	24,786,000	3,240,000
11	300	No solution	No solution	4,248,000
12	300	No solution	No solution	9,720,000
13	80	No solution	No solution	16,380,000
14	67	n/a	n/a	22,572,000

Table 2 – Minimum computational efforts for the even-n-parity problem for all 3 methods.

The GP+CADF method successfully solved the even-n-parity problem up to n=14. We could not proceed further not because GP+CADF was not able to solve even-15-parity, but because increasing evaluation times made it infeasible to accumulate a sufficient number of trials to obtain a successful run.

5.2 The Sum Of Bits Problem

The n-s-sum of bits problem takes as input a bit string of length n and a desired sum s. The goal is to find all those bit strings where the number of 1's adds up exactly to the sum s. We invented this problem specifically for the purpose of comparing the different GP methods with a problem that is harder than the even-n-parity problem and that is also scalable in the same way.

Concerning scalability, the difficulty of this problem is directly proportional to n and the problem is also hardest when s is half the value of n.

The results showed that computation effort was exponentially increasing with n and that GP+CADF was the only method to solve the sum-of-bits problem for 6-s-sum of bits where s is between 2 and 4.

5.3 The Image Recognition Problem

The letter recognition problem is based on work done by Koza in chapter 15 of [1]. The inputs to the problem consist of a grid of 6x4 pixels. For the purposes of this problem, we will only use the letters I and L. The reason we only use I, L and various samples that are neither is that we are not interested in a genuine letter recognition task. Rather, we are examining a problem where we want to recognize two things perfectly among many samples. The main motivation is the discovery of solutions though subroutines that examine local pixel neighborhoods and the goal of the experiment will be to see if GP+CADF improves on GP+ADF. The inputs will include the letters I and L and 75 other images that represent a range of possibilities; some resembling the letters, and some that look very different. In other words, we will have two positive and 75 negative cases. Fig. 6 shows the set of images that will be used in this problem. The goal for the program will be to correctly identify the two letters and dismiss all other cases as negatives.

M	SP	Regular GP	GP+ADF	GP+CADF
4000	3	No Solution	8,880,000	5,760,000
3000	1	No Solution	60,888,000	40,356,000
3000	1	No Solution	40,356,000	30,780,000

Table 3 – Minimum computational efforts for the image recognition problem. SP denotes success predicate which is how many fitness cases are allowed to be wrong and still obtain perfect fitness.

The experimental results are given in Table 3 and show that GP+CADF yielded a lower computational effort in all tests.

5.4 Additional Experiments

Experiments were also performed on the lawnmower, obstacle avoiding robot, minesweeper and bumblebee problems as described by Koza in [1]. The results obtained from these experiments also confirm that GP+CADF requires less computational effort than regular GP or GP+ADF.

Problem	Regular GP	GP+ADF	GP+CADF
Lawnmower	No Solution	12,000	6,600
Obstacle Avoiding Robot	No Solution	264,600	126,000
Minesweeper	No Solution	4,176,900	2,318,400
Bumblebee	No Solution	172,800	148,800

Table 4 – Minimum computational efforts for lawn coverage (lawnmower, obstacle avoiding robot and minesweeper) and bumblebee problems.

6 Conclusion

This work began by asking whether the addition of co-evolution to genetic programming would help to overcome the current limitations of the standard GP approach as well as GP augmented with automatically defined functions. We have shown that GP+CADF is a feasible co-evolutionary approach that continues discovery of solutions where regular GP and GP+ADF reach their limits. We have shown that GP+CADF yields solutions for up to 14-even-parity and 6-3 sum of bits problems where other methods find no solutions. We have also shown that GP+CADF requires a lower computational effort to solve the parity, sum of bits, image recognition, lawn coverage and the bumblebee problems.

To further improve GP+CADF, we discovered that using elitism and a single best evaluation coupling lowers the computational effort even further. In addition, improving the initial population by regenerating individuals with no ADF calls and choosing best initial individuals after M couplings, where M is the population size, also leads to a lowered computational effort.

References:

[1] J.R. Koza, Genetic programming II: Automatic discovery of reusable programs, MIT Press. 1994.

[2] W. B. Langdon and R. Poli, Foundations Of Genetic Programming, Springer-Verlag New York, Inc., New York, NY, 2002

[3] R. Aler, Immediate transfer of global improvements to all individuals in a population compared to Automatically Defined Functions for the EVEN-5,6-PARITY problems, Springer Berlin Heidelberg, 1998.

[4] W. Daniel Hillis, Co-evolving parasites improve simulated evolution as an optimization procedure, Physica D, v.42 n.1-3, p.228-234, June 1990.

[5] N. Williams , M. Mitchell, Investigating the success of spatial coevolution, Proceedings of the 2005 conference on Genetic and evolutionary computation, June 25-29, 2005.

[6] Edwin D. De Jong, Jordan B. Pollack, Ideal Evaluation from Coevolution, Evolutionary Computation, v.12 n.2, p.159-192, June 2004.

[7] Mitchell A. Potter , Kenneth A. De Jong, A Cooperative Coevolutionary Approach to Function Optimization, Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature: Parallel Problem Solving from Nature, p.249-257, 1994.

[8] Mitchell A. Potter , Kenneth A. De Jong, Cooperative Coevolution: An Architecture for Evolving Coadapted Subcomponents, , Evolutionary Computation, v.8 n.1, p.1-29, 2000.

[9] Tulai, A. F. and Oppacher, F., Combining Competitive And Cooperative Coevolution For Training Cascade Neural Networks, In Proceedings of the Genetic and Evolutionary Computation Conference 2002.