Adaptive Genetic Programming Applied to New and Existing Simple Regression Problems

Jeroen Eggermont and Jano I. van Hemert

Leiden Institute of Advanced Computer Science, Leiden University
{jeggermo,jvhemert}@cs.leidenuniv.nl

Abstract. In this paper we continue our study on adaptive genetic programming. We use Stepwise Adaptation of Weights (SAW) to boost performance of a genetic programming algorithm on simple symbolic regression problems. We measure the performance of a standard GP and two variants of SAW extensions on two different symbolic regression problems from literature. Also, we propose a model for randomly generating polynomials which we then use to further test all three GP variants.

1 Introduction

We test a technique called Stepwise Adaptation of Weights (SAW) on symbolic regression. We use a genetic programming algorithm and adapt its fitness function using the SAW technique in an attempt to improve both algorithm performance and solution quality. Also, we present a new variant of the SAW technique called Precision SAW.

Previous studies on constraint satisfaction [6] and data classification [4] have indicated that SAW is a promising technique to boost the performance of evolutionary algorithms. It uses information of the problem from the run so far to adapt the fitness function of an evolutionary algorithm.

In a regression problem we are looking for a function that closely matches an unknown function based on a finite set of sample points. Genetic programming (GP) as introduced by Koza [9] uses a tree structure to represent an executable object or model. Here we will use GP to search for functions that solve a symbolic regression problem.

The next section defines the symbolic regression problem and how this is solved using genetic programming. In Sect. 3 we provide information on the SAW technique and show how this technique can be applied to symbolic regression. In Sect. 5 we explain our experiments and we provide results. Finally we draw conclusions and we provide ideas for further research.

2 Symbolic Regression with Genetic Programming

The object of solving a symbolic regression problem is finding a function that closely matches some unknown function on a certain interval. More formally, given an unknown function $f(x)$ we want to find a function $g(x)$ such that

$f(x_i) = g(x_i)\ \forall x_i \in X$, where X is a set of values drawn from the interval we are interested in. Note that we normally do not know $f(x)$ precisely. We only know the set of sample points $\{(x, f(x)) | x \in X\}$. In this study we use predefined functions and uniformly draw a set of 50 sample points from it to test our regression algorithms equivalent to the experiments with these functions in [10].

We use a genetic programming algorithm to generate candidate solutions, i.e., $g(x)$. These functions are presented as binary trees built up using binary functions and a terminal set. The precise definition of these sets and other parameter settings varies between the two experiments presented later. Therefore, we defer the presentation of these parameters until Sect. 5 where we conduct our experiments.

The selection scheme in an evolutionary algorithm is one of its basic components. It needs a way to compare the quality of two candidate solutions. This measurement, the fitness function, is calculated using knowledge of the problem. In symbolic regression we want to minimise the total error over all samples. This is defined as the absolute error in (1), which will be the fitness function for the standard GP algorithm.

$$\epsilon = \sum_{x \in X} |f(x) - g(x)| \tag{1}$$

Other fitness functions can be used. For instance, based on the mean square error. We use this simple approach and make it adaptive in the next section.

3 Stepwise Adaptation of Weights

The Stepwise Adaptation of Weights (SAW) technique was first studied on the constraint satisfaction problem (CSP). Solving a CSP can mean different things. Here the object is to find an instantiation of a set of variables such that none of the constraints that restrict certain combinations of variable instantiations are violated. This is a NP-hard problem on which most evolutionary computation approaches will fail because they get stuck in local optima. The SAW technique is designed to overcome this deficiency.

In data classification the problem is to find a model that can classify tuples of attributes as good as possible. When we observe this problem with constraint satisfaction in mind we can draw some analogies. For instance, in CSP we have to deal with constraints. Minimising the number of violated constraints is the object and having no violated constraints at all yields a solution. Similarly, in data classification, minimising the number of wrongly classified records is the object, while correctly classifying all records yields a perfect model.

The idea of data classification can further be extended to symbolic regression. Here we want to find a model, i.e., a function, that correctly predicts values of the unknown function on the sampled points. The object is minimising the error of prediction, consequently having no error means having found a perfect fit.

This is where SAW steps into the picture by influencing the fitness function of an evolutionary algorithm. Note that in all the problems mentioned above the fitness has to be minimised to reach the object. The idea behind SAW is to adapt the fitness function in an evolutionary algorithm by using knowledge of the problem in the run so far.

Algorithm 1 Stepwise adaptation of weights (SAW)

```
set initial weights (thus fitness function $f$)
set $G = 0$
while not termination do
  $G = G + 1$
  run one generation of GP with $f$
  if $G \equiv 0 \pmod{\Delta T}$ then
    redefine $f$ and recalculate fitness of individuals
  fi
end while
```

The general mechanism for SAW is presented in Figure 1. The knowledge of the problem is represented in the form of weights. We add a weight w_i to every sample point $x_i \in X$. These weights are initially set to one. During the run of the genetic programming algorithm we periodically stop the main evolutionary loop every ΔT generations and adjust the weights by increasing them. Afterwards, we continue the evolutionary loop using the new weights incorporated into the fitness function as shown in (2).

$$\textit{saw fitness}(g, X) = \sum_{x_i \in X} w_i |f(x_i) - g(x_i)| \tag{2}$$

The adaptation of weights process takes the best individual from the current population and determines the error it makes on each sample point. Each of the weights w_i corresponding to the error made on point x_i is updated using the error value $|f(x_i) - g(x_i)|$. We try two variants for altering weights.

– *Classic* SAW (CSAW) adds a constant value $\Delta w_i = 1$ to each w_i if the error on sample point x_i is not zero. This is based on the approach of violated constraints [6].
– *Precision* SAW (PSAW) takes $\Delta w_i = |f(x_i) - g(x_i)|$ and adds Δw_i to the corresponding weight w_i.

4 Problem generator

Besides testing our two GP variants on two problems taken from literature we present here a method for generating symbolic regression problems. This enables us to study the performance of these techniques on a large set of polynomials of a higher degree. The regression of higher degree polynomials finds its use in different application areas such as computer vision and economics [2].

We generate the polynomials using a model of two integer parameters $\langle a, b \rangle$, where a stands for the highest possible degree and b determines the domain size from which every e_i is chosen. When we have set these parameters we are able to generate polynomials in the form as shown in (3). The values e_i are drawn uniform random from the integer domain bounded by $-b$ and b.

$$\sum_{i=0}^{a} \left(e_i x^i\right), \text{ where } e_i \in \{w | w \in \mathbb{Z} \wedge -b \leq w \leq b\} \tag{3}$$

The function $f(x)$ is presented to the regression algorithms by generating 50 points $(x, f(x))$ uniformly from the domain $[-1, 1]$. This method for generating X will also be used for the two Koza functions [10].

5 Experiments and Results

5.1 Koza functions

To test the performance of the two variants of SAW we do a number of experiments using three algorithms. First, the genetic programming algorithm without any additional aids (GP). Second, the variant where we add SAW with a constant Δw (GP-CSAW). Last, the variant where we add SAW with $\Delta w_i = |f(x_i) - g(x_i)|$ (GP-PSAW).

We measure performance of the algorithms on two simple symbolic regression problems. Each algorithm is tested with two different population sizes as shown in Table 1. We use 99 independent runs for each setting in which we measure mean, median, standard deviation, minimum and maximum absolute error (ϵ). Furthermore we count the number of successful runs. Where we define a run successful if the algorithm finds a function that has an absolute error below 10^{-6}.

Besides comparing the three genetic programming variants we also provide results obtained by using splines on the same problems. This places our three algorithms in a larger perspective.

Table 1. Experiment parameters: six different experiments where each experiment consists of 99 independent runs

experiment	populations size	number of generations
1	100	100
2	100	200
3	100	500
4	100	1000
5	500	100
6	500	200

Here we present two experiments with functions taken from [10]. All genetic programming systems used the parameters shown in Table 2.

Table 2. Parameters and characteristics of the genetic programming algorithms for experiments with the Koza functions

parameter	value
evolutionary model	$(\mu, 7\mu)$
fitness standard GP	see (1)
fitness SAW variants	see (2)
stop criterion	maximum generations or perfect fit
functions set	$\{*, \text{pdiv}, -, +\}$
terminal set	$\{x\}$
populations size	see Table 1
initial depth	3
maximum depth	5
maximum generations	see Table 1
survivors selection	keep best populations size offspring
parent selection	random
ΔT (for SAW)	5

Quintic polynomial This quintic polynomial is taken from [10] and is defined by (4). The function is shown in Fig. 1. When we try to regress this function using standard cubic regression we get a total error (as measured with ϵ) of 1.8345.

$$f(x) = x^5 - 2x^3 + x,\ x \in [-1, 1] \tag{4}$$

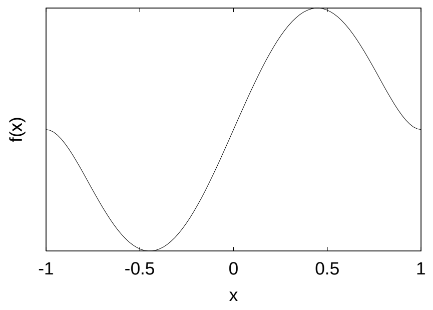

Fig. 1. The quintic polynomial on the interval $[-1, 1]$ (left)

Table 3 shows the results of the experiments on the quintic polynomial. Looking at the mean and the median for all experiments we conjecture that GP-PSAW produces the best solutions. GP-CSAW is not always better than GP, but it has the best results when we observe the maximum error, i.e., the worst result found. The best solution (1.704×10^{-7}) is found twice by GP-CSAW.

We look at the individual runs of experiment 4 (population size of 100 and 1000 generations) and determine all the successful runs and see that GP has 76

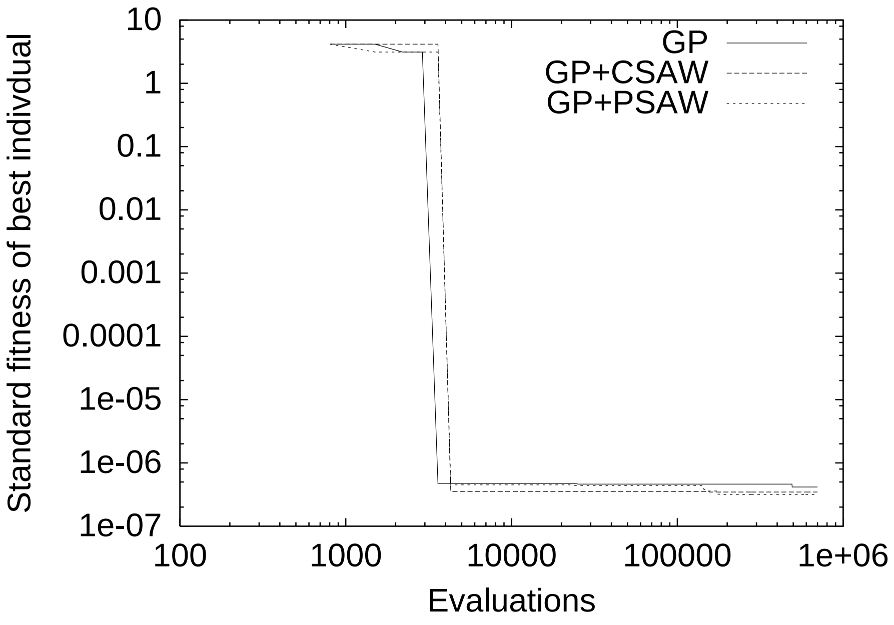

Fig. 2. Absolute error (ϵ) over evaluations for one run of the three algorithms for the quintic polynomial (right)

successful runs out of 99, GP-CSAW has 78 successful runs out of 99 and GP-PSAW has 85 successful runs of 99. These result are set out in Fig. 3 together with their uncertainty interval [13]. We use the usual rule of thumb where we need at least a difference two and a half times the overlap of the uncertainty interval before we can claim a significant difference. This is clearly not the case here. In experiment 6 (population size 500 and 200 generations) GP fails 2 times out of 99. The other two algorithms never fail. This is a significant difference, albeit a small one as the uncertainty intervals still overlap.

Sextic polynomial This sextic polynomial is taken from [10] and is defined in (5). Figure 4 shows this function on the interval $[-1, 1]$. When we do a cubic regression on the function we get a total error (measured with ϵ) of 2.4210.

$$f(x) = x^6 - 2x^4 + x^2,\ x \in [-1, 1] \tag{5}$$

Table 4 shows the results of the experiments on the sextic polynomial. GP-PSAW has the best median in one experiment and best mean in three experiments. GP-CSAW never has the best mean but has the best median three times. If we are only interested in the best solution over all runs we have an easier job comparing. The best solution (1.013×10^{-7}) is found in experiment 2 by GP-PSAW in just 200 generations and by GP-CSAW in experiment 5 within 100 generations.

Table 3. Experiment results for the quintic polynomial, all measurements taken with absolute error (ϵ)

experiment	median ($\times 10^{-7}$)	mean ($\times 10^{-1}$)	stddev ($\times 10^{-1}$)	minimum ($\times 10^{-7}$)	maximum
1. GP	4.610	1.351	2.679	2.640	1.050
GP-CSAW	4.605	1.339	2.599	2.445	1.102
GP-PSAW	4.391	1.286	2.972	2.598	1.559
2. GP	4.354	1.274	2.610	2.640	1.034
GP-CSAW	4.303	1.226	2.376	2.445	0.8525
GP-PSAW	4.200	1.049	2.254	2.543	1.317
3. GP	3.972	1.120	2.571	2.640	1.034
GP-CSAW	4.019	1.107	2.204	1.704	0.8525
GP-PSAW	3.855	0.7785	2.049	2.449	1.111
4. GP	3.763	1.161	2.547	2.324	1.034
GP-CSAW	3.693	0.8323	1.803	1.704	0.6969
GP-PSAW	3.669	0.6513	1.856	2.114	1.111
5. GP	3.465	2.045×10^{-1}	1.544×10^{-2}	1.965	1.433×10^{-1}
GP-CSAW	3.465	3.570×10^{-5}	8.463×10^{-8}	2.412	9.965×10^{-7}
GP-PSAW	3.395	3.382×10^{-5}	4.384×10^{-8}	1.974	5.071×10^{-7}
6. GP	3.343	2.045×10^{-1}	1.544×10^{-2}	1.965	1.433×10^{-1}
GP-CSAW	3.446	3.512×10^{-5}	8.337×10^{-8}	2.412	9.965×10^{-7}
GP-PSAW	3.325	3.331×10^{-5}	4.533×10^{-8}	1.937	5.071×10^{-7}

Table 4. Experiment results for the sextic polynomial, all measurements taken with absolute error (ϵ)

experiment	median ($\times 10^{-7}$)	mean ($\times 10^{-1}$)	stddev ($\times 10^{-1}$)	minimum ($\times 10^{-7}$)	maximum
1. GP	2.182	1.490	3.987	1.723	2.844
GP-CSAW	2.182	1.525	4.036	1.353	2.844
GP-PSAW	2.182	1.212	2.569	1.213	1.720
2. GP	2.182	1.179	2.882	1.172	1.730
GP-CSAW	2.098	1.244	3.626	1.244	2.491
GP-PSAW	2.115	1.135	2.495	1.013	1.720
3. GP	1.953	0.8001	2.318	1.171	1.730
GP-CSAW	1.916	0.8366	2.328	1.172	1.222
GP-PSAW	1.984	0.8403	2.226	1.013	1.720
4. GP	1.888	0.6963	2.258	1.135	1.730
GP-CSAW	1.824	0.6100	1.741	1.048	1.161
GP-PSAW	1.899	0.5084	1.418	1.013	5.467×10^{-1}
5. GP	1.385	1.507×10^{-6}	3.280×10^{-7}	1.087	2.912×10^{-7}
GP-CSAW	1.385	3.390×10^{-2}	2.500×10^{-1}	1.013	2.255×10^{-1}
GP-PSAW	1.385	2.485×10^{-2}	2.460×10^{-1}	1.125	2.460×10^{-1}
6. GP	1.260	1.417×10^{-6}	3.029×10^{-7}	1.087	2.912×10^{-7}
GP-CSAW	1.363	3.390×10^{-2}	2.500×10^{-1}	1.013	2.255×10^{-1}
GP-PSAW	1.260	2.485×10^{-2}	2.460×10^{-1}	1.115	2.4560×10^{-1}

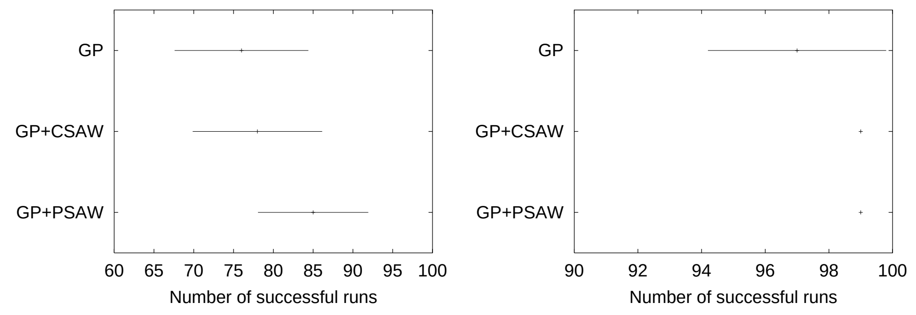

Fig. 3. Number of successful runs with uncertainty intervals for the quintic polynomial in experiment 4 (left) and experiment 6 (right)

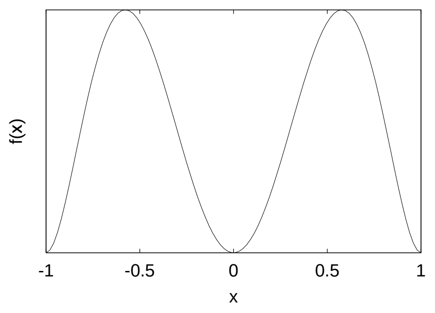

Fig. 4. The sextic polynomial on the interval $[-1, 1]$

Similar to the quintic polynomial we examine the individual runs of experiment 4 (population size of 100 and 1000 generations) and determine all the successful runs. Here GP has 84 successful runs out of 99. GP-CSAW has 81 successful runs and GP-PSAW has 87 successful runs. These result are set out in Fig. 6 together with the uncertainty interval [13]. Although differences seem larger than with the quintic polynomial we still cannot claim a significant improvement. Tides turn compared to the quintic polynomial as here GP-CSAW fails twice, GP-PSAW fails once and GP always succeeds. This leads to a significant difference with overlapping uncertainty intervals between GP and GP-CSAW.

5.2 Randomly generated polynomials

Here we test our three GP variants on a suite of randomly generated polynomials. These polynomials can have at most a degree of twelve. The parameters of the underlying GP system are noted in Table 5

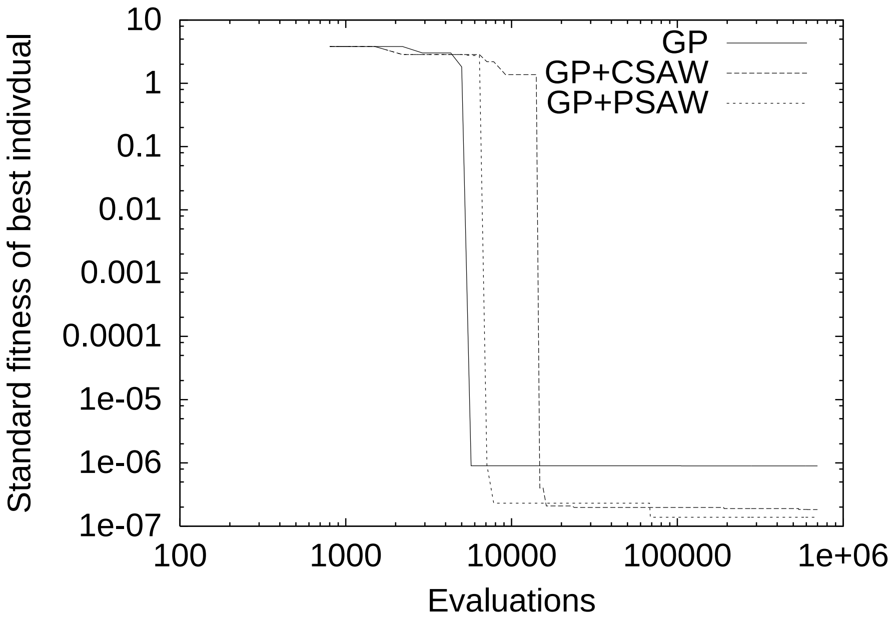

Fig. 5. Absolute error (ϵ) over evaluations for one run of the three algorithms for the sextic polynomial (right)

We generate polynomials randomly with the model from Sect. 4 with parameters $\langle 12, 5 \rangle$. We generate 100 polynomials and do 85 independent runs of each algorithm where each run is started with a unique random seed.

The previous simple functions had a smaller maximum degree and could therefore be efficiently solved with cubic splines. Here we need a technique that can handle a higher degree of polynomials. Hence, we try a curve fitting algorithm that uses splines under tension [3] on every generated polynomial. The results are very good as the average total error over the whole set of polynomials measured as the absolute error (ϵ) is 1.3498346×10^{-4}.

We present the results for the set of randomly generated polynomials in Table 6. Clearly, the new variant GP-PSAW is better than any of the others as long as we do not update the SAW weights too many times. This seems like a contradiction as we would expect the SAW mechanism to boost improvement, but we should not forget that after updating the weights the fitness function has changed so we need to re-calculate the fitness for the whole population. This re-calculateion can be performed by either re-evaluating all individuals or by using a cache-memory which contains the predicted value for each point and individual. Because we have used a general library we opted for the re-evaluation. Also, when we would extend the system in such a way that the data points would change during the run this would render the cache useless. But, in a GP system

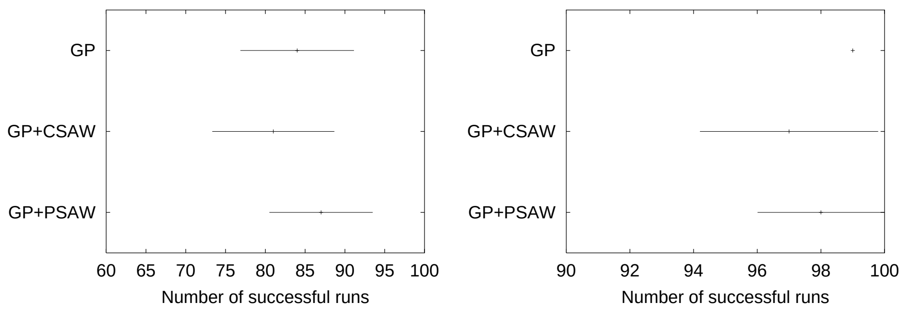

Fig. 6. Number of successful runs with uncertainty intervals for the sextic polynomial in experiment 4 (left) and experiment 6 (right)

Table 5. Parameters and characteristics of the genetic programming algorithms for experiments with the random polynomials

parameter	value
evolutionary model	steady state $(\mu + 1)$
fitness standard GP	see (1)
fitness SAW variants	see (2)
stop criterion	maximum evaluations or perfect fit
functions set	$\{*, \text{pdiv}, -, +, \neg, y^3\}$
terminal set	$\{x\} \cup \{w \mid w \in \mathbb{Z} \wedge -b \leq w \leq b\}$
populations size	100
initial maximum depth	10
maximum size	100 nodes
maximum evaluations	20,000
survivors selection	reverse 5-tournament
parent selection	5-tournament
ΔT (for SAW)	1000, 2000 and 5000 generations

this will cost us fitness evaluations, leaving less fitness evaluations for the GP. Thus less points of the problem space are visited.

The comparison with the splines on tension only remains. The error of the splines on tension is incredibly low. As such our results are significantly worse. One reason could be the number of evaluations that are performed at maximum. Here we have set this at 20,000. If we look at the right hand of Figure 4 we notice that the absolute error suddenly drops after 10,000 evaluations. If we take into account that our current polynomials have a degree that is up to twice that of the sextic polynomial we can expect that a better solution will not be found before this 20,000 mark.

Table 6. Experiment results for 100 randomly generated polynomials, results are averaged over the 85 independent runs. The error is measured as absolute error (1)

algorithm	ΔT	mean	stddev	minimum	maximum
GP		21.20	10.30	5.82	48.60
GP-CSAW	1000	21.55	10.33	5.79	48.05
GP-CSAW	2000	21.21	10.24	5.83	48.74
GP-CSAW	5000	21.25	10.47	5.75	49.31
GP-PSAW	1000	21.46	10.52	6.41	50.80
GP-PSAW	2000	20.45	10.05	5.90	48.06
GP-PSAW	5000	20.28	9.87	5.96	48.18

6 Conclusions

We have shown how the SAW technique can be used to extend genetic programming to boost performance in symbolic regression. We like to point out that the simple concept behind SAW makes it very easy to implement the technique in existing algorithms. Thereby, making it suitable for doing quick try outs to boost an evolutionary algorithms performance. As SAW solely focuses on the fitness function it can be used in virtually any evolutionary algorithm. However, it is up to the user to find a good way of updating the weights mechanism depending on the problem at hand. This paper shows two ways in which to add SAW to a genetic programming algorithm that solves symbolic regression problems. By doing this, we add another problem area to a list of problems that already contains various constraint satisfaction problems and data classification problems.

When we focus on a comparison of mean, median and minimum fitness it appears that our new variant of the SAW technique (Precision SAW) has the upper hand. In most cases it finds a better or comparable result than our standard GP, also beating the Classic SAW variant. Moreover, it seems that the SAW technique works better using smaller populations. Something that is already concluded by Eiben et al. [5].

When we restrict our comparison to the number of successes and fails we find that there is only a small significant difference when we run the algorithms with a populations size of 500. Then GP+PSAW is the winner for the quintic polynomial and GP for the sextic polynomial.

Our GP algorithms perform poorly on the suit of randomly generated polynomials compared to splines on tension. We suspect the cause to lie in the maximum number of evaluations. The Koza functions have showed us that the drop in absolute error can happen suddenly. We conjecture that our GP systems need more time on the higher degree polynomials before this drop occurs.

7 Future Research

We need to enhance our polynomial generator as we still have unanswered questions about which polynomials are easy to regress and which are not. Maybe we can alter or bias the generators parameters such that we can model polynomials of which we have prior knowledge. A technique that is used in fields such as constrained optimisation [12] and constraint satisfaction [1].

The performance of genetic programming is seen as an interesting problem to overcome as many other techniques exist to boost performance of genetic programming [7, 15, 8]. These technique often are bias towards handling large data sets which is not the case for the problems we have described.

Looking at symbolic regression as used in this paper presents a problem that is viewed as a static set of sample points. That is, the set of sample points is drawn uniformly out of the interval of the unknown function and stays the same during the run. Therefore, the problem is not finding an unknown function, but just a function that matches the initial sample points. To circumvent this we could use a co-evolutionary approach [14] that adapts the set of points we need to fit, thereby creating an arms-race between a population of solutions and a population of sets of sample points.

Acknowledgements

The program for the the Koza functions was created with the GP kernel from the Data to Knowledge research project of the Danish Hydraulic Institute (`http://www.d2k.dk`).

The program for the randomly generated functions was created with the Evolvable Objects library [11] which can be downloaded from `http://sourceforge.net/projects/eodev`.

References

[1] Dimitris Achlioptas, Lefteris M. Kirousis, Evangelos Kranakis, Danny Krizanc, Michael S.O. Molloy, and Yannis C. Stamatiou. Random constraint satisfaction a more accurate picture. In Gert Smolka, editor, *Principles and Practice of Constraint Programming — CP97*, pages 107–120. Springer-Verlag, 1997.

[2] With S. Ar, R. Lipton, and R. Rubinfeld. Reconstructing algebraic functions from erroneous data. *SIAM Journal on Computing*, 28(2):487–510, 1999.

[3] A. K. Cline. Six subprograms for curve fitting using splines under tension. *Commun. ACM*, 17(4):220–223, April 1974.

[4] J. Eggermont, A.E. Eiben, and J.I. van Hemert. Adapting the fitness function in GP for data mining. In R. Poli, P. Nordin, W.B. Langdon, and T.C. Fogarty, editors, *Genetic Programming, Proceedings of EuroGP'99*, volume 1598 of *LNCS*, pages 195–204, Goteborg, Sweden, 26–27 May 1999. Springer-Verlag.

[5] A.E. Eiben, J.K. van der Hauw, and J.I. van Hemert. Graph coloring with adaptive evolutionary algorithms. *Journal of Heuristics*, 4(1):25–46, 1998.

[6] A.E. Eiben and J.I. van Hemert. *SAW-ing EAs: adapting the fitness function for solving constrained problems*, chapter 26, pages 389–402. McGraw-Hill, London, 1999.

[7] C. Gathercole and P. Ross. Dynamic training subset selection for supervised learning in genetic programming. In *Proceedings of the Parallel Problem Solving from Nature III Conference*, pages 312–321, 1994.

[8] Chris Gathercole and Peter Ross. Tackling the boolean even N parity problem with genetic programming and limited-error fitness. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 119–127, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.

[9] J.R. Koza. *Genetic Programming.* MIT Press, 1992.

[10] J.R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs.* MIT Press, Cambridge, MA, 1994.

[11] J. J. Merelo, J. Carpio, P. Castillo, V. M. Rivas, and G. Romero. Evolving objects, 1999. Available at `http://geneura.ugr.es/~jmerelo/EOpaper/`.

[12] Z. Michalewicz, K. Deb, M. Schmidt, , and T. Stidsen. Test-case generator for nonlinear continuous parameter optimization techniques. *IEEE Transactions on Evolutionary Computation*, 4(3), 2000.

[13] David S. Moore and George P. McCabe. *Introduction to the Practice of Statistics.* W.H. Freeman and Company, New York, 3rd edition, 1998.

[14] J. Paredis. Co-evolutionary computation. *Artificial Life*, 2(4):355–375, 1995.

[15] Byoung-Tak Zhang. Bayesian methods for efficient genetic programming. *Genetic Programming And Evolvable Machines*, 1(3):217–242, July 2000.