Stepwise Adaptation of Weights for Symbolic Regression with Genetic Programming

J. Eggermont J.I. van Hemert
{jeggermo,jvhemert}@cs.leidenuniv.nl

Leiden University, P.O.Box 9512, 2300 RA, Leiden, The Netherlands

Abstract

In this paper we continue study on the Stepwise Adaptation of Weights (SAW) technique. Previous studies on constraint satisfaction and data classification have indicated that SAW is a promising technique to boost the performance of evolutionary algorithms. Here we use 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.

1 Introduction

In this study we test a technique called Stepwise Adaptation of Weights (SAW) on symbolic regression. We use a genetic programming algorithm and adapts its fitness function using the SAW technique in an attempt to improve both algorithm performance and solution quality.

The SAW technique is studied on many constraint satisfaction problems [3] and on binary data classification [1]. 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 [4] 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 Section 3 we provide information on the SAW technique and show how this technique can be applied to symbolic regression. In Section 4 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.

We use a genetic programming algorithm to generate candidate solutions, i.e., $g(x)$. These functions are presented as binary trees build up using multiplication, addition, subtraction, protected divide and the variable x. The genetic algorithm will use standard crossover and mutation operators as defined by Koza [4]. Furthermore its selection model is deterministic and generational. The parameters and characteristics of the algorithm are in Table 1.

Table 1: Parameters and characteristics of the genetic programming algorithm

parameter	value
evolutionary model $(\mu + \lambda)$	$(\mu + 7\mu)$
fitness standard GP	see Equation 2
fitness SAW variants	see Equation 3
stop criterion	maximum generations or perfect fit
functions set	{*, %, -, +}
terminal set	{x}
populations size	see Table 2
maximum generations	see Table 2
survivors selection	keep best populations size individuals
parent selection	random

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 error ϵ as defined in Equation 1.

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

Thus the fitness function becomes:

$$\textit{standard fitness}(g, X) = \sum_{x \in X} |f(x) - g(x)| \tag{2}$$

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

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, having no error is 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.

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 five generations[1] and adjust the weights. Afterwards we continue the evolutionary loop using the new weights incorporated into the fitness function as shown in Equation 3.

$$\textit{saw fitness}(g, X) = \sum_{x_i \in X} w_i |f(x_i) - g(x_i)| \tag{3}$$

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. The first variant *classic* SAW (CSAW) adds a constant value $\Delta w = 1$ to each w_i if the error on sample point x_i is not zero. This is based on the approach of violated constraints [3]. The second variant *precision* SAW (PSAW) takes $\Delta w = |f(x_i) - g(x_i)|$ and adds Δw to the corresponding w_i. We hope this variant is better in dealing with the real valued domain of the error.

[1]This value is based on previous studies on SAW

4 Experiments and Results

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 $\Delta w = |f(x) - g(x)|$ (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 2. We use 99 independent runs for each setting in which we measure mean, median, standard deviation, minimum and maximum standard fitness. Furthermore we count the number of successful runs. Where we define a run successful if the algorithm finds a function that has a standard fitness below 10^{-6}.

Table 2: 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

4.1 Quintic polynomial

This quintic polynomial is taken from [5] and is defined by Equation 4. The function is shown in Figure 1.

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

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 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 Figure 3 together with their uncertainty interval [7]. 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

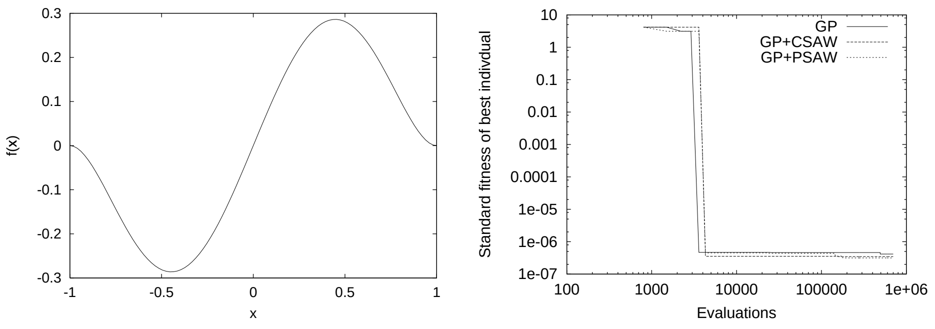

Figure 1: The quintic polynomial on the interval $[-1, 1]$ (left). Standard fitness function over evaluations for one run of the three algorithms for the quintic polynomial (right)

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.

4.2 Sextic polynomial

This sextic polynomial is taken from [5] and is defined in Equation 5. Figure 2 shows this function on the interval $[-1, 1]$.

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

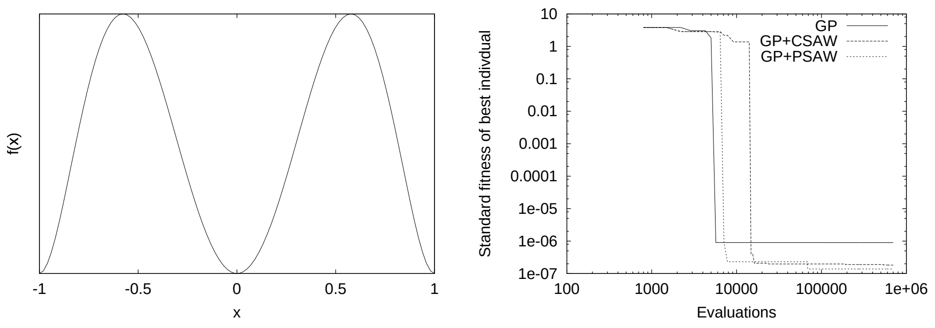

Figure 2: The sextic polynomial on the interval $[-1, 1]$ (left). Standard fitness function over evaluations for one run of the three algorithms for the sextic polynomial (right)

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 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.

Table 3: Experiment results for the quintic polynomial (all measurements with standard fitness function)

experiment		median	mean	st. deviation	minimum	maximum
1.	GP	4.610×10^{-7}	1.351×10^{-1}	2.679×10^{-1}	2.640×10^{-7}	1.050
	GP-CSAW	4.605×10^{-7}	1.339×10^{-1}	2.599×10^{-1}	2.445×10^{-7}	1.102
	GP-PSAW	4.391×10^{-7}	1.286×10^{-1}	2.972×10^{-1}	2.598×10^{-7}	1.559
2.	GP	4.354×10^{-7}	1.274×10^{-1}	2.610×10^{-1}	2.640×10^{-7}	1.034
	GP-CSAW	4.303×10^{-7}	1.226×10^{-1}	2.376×10^{-1}	2.445×10^{-7}	8.525×10^{-1}
	GP-PSAW	4.200×10^{-7}	1.049×10^{-1}	2.254×10^{-1}	2.543×10^{-7}	1.317
3.	GP	3.972×10^{-7}	1.120×10^{-1}	2.571×10^{-1}	2.640×10^{-7}	1.034
	GP-CSAW	4.019×10^{-7}	1.107×10^{-1}	2.204×10^{-1}	1.704×10^{-7}	8.525×10^{-1}
	GP-PSAW	3.855×10^{-7}	7.785×10^{-2}	2.049×10^{-1}	2.449×10^{-7}	1.111
4.	GP	3.763×10^{-7}	1.161×10^{-1}	2.547×10^{-1}	2.324×10^{-7}	1.034
	GP-CSAW	3.693×10^{-7}	8.323×10^{-2}	1.803×10^{-1}	1.704×10^{-7}	6.969×10^{-1}
	GP-PSAW	3.669×10^{-7}	6.513×10^{-2}	1.856×10^{-1}	2.114×10^{-7}	1.111
5.	GP	3.465×10^{-7}	2.045×10^{-3}	1.544×10^{-2}	1.965×10^{-7}	1.433×10^{-1}
	GP-CSAW	3.465×10^{-7}	3.570×10^{-7}	8.463×10^{-8}	2.412×10^{-7}	9.965×10^{-7}
	GP-PSAW	3.395×10^{-7}	3.382×10^{-7}	4.384×10^{-8}	1.974×10^{-7}	5.071×10^{-7}
6.	GP	3.343×10^{-7}	2.045×10^{-3}	1.544×10^{-2}	1.965×10^{-7}	1.433×10^{-1}
	GP-CSAW	3.446×10^{-7}	3.512×10^{-7}	8.337×10^{-8}	2.412×10^{-7}	9.965×10^{-7}
	GP-PSAW	3.325×10^{-7}	3.331×10^{-7}	4.533×10^{-8}	1.937×10^{-7}	5.071×10^{-7}

Table 4: Experiment results for the sextic polynomial (all measurements with standard fitness function)

experiment		median	mean	st. deviation	minimum	maximum
1.	GP	2.182×10^{-7}	1.490×10^{-1}	3.987×10^{-1}	1.723×10^{-7}	2.844
	GP-CSAW	2.182×10^{-7}	1.525×10^{-1}	4.036×10^{-1}	1.353×10^{-7}	2.844
	GP-PSAW	2.182×10^{-7}	1.212×10^{-1}	2.569×10^{-1}	1.213×10^{-7}	1.720
2.	GP	2.182×10^{-7}	1.179×10^{-1}	2.882×10^{-1}	1.172×10^{-7}	1.730
	GP-CSAW	2.098×10^{-7}	1.244×10^{-1}	3.626×10^{-1}	1.244×10^{-7}	2.491
	GP-PSAW	2.115×10^{-7}	1.135×10^{-1}	2.495×10^{-1}	1.013×10^{-7}	1.720
3.	GP	1.953×10^{-7}	8.001×10^{-2}	2.318×10^{-1}	1.171×10^{-7}	1.730
	GP-CSAW	1.916×10^{-7}	8.366×10^{-2}	2.328×10^{-1}	1.172×10^{-7}	1.222
	GP-PSAW	1.984×10^{-7}	8.403×10^{-2}	2.226×10^{-1}	1.013×10^{-7}	1.720
4.	GP	1.888×10^{-7}	6.963×10^{-2}	2.258×10^{-1}	1.135×10^{-7}	1.730
	GP-CSAW	1.824×10^{-7}	6.100×10^{-2}	1.741×10^{-1}	1.048×10^{-7}	1.161
	GP-PSAW	1.899×10^{-7}	5.084×10^{-2}	1.418×10^{-1}	1.013×10^{-7}	5.467×10^{-1}
5.	GP	1.385×10^{-7}	1.507×10^{-7}	3.280×10^{-8}	1.087×10^{-7}	2.912×10^{-7}
	GP-CSAW	1.385×10^{-7}	3.390×10^{-3}	2.500×10^{-2}	1.013×10^{-7}	2.255×10^{-1}
	GP-PSAW	1.385×10^{-7}	2.485×10^{-3}	2.460×10^{-2}	1.125×10^{-7}	2.460×10^{-1}
6.	GP	1.260×10^{-7}	1.417×10^{-7}	3.029×10^{-8}	1.087×10^{-7}	2.912×10^{-7}
	GP-CSAW	1.363×10^{-7}	3.390×10^{-3}	2.500×10^{-2}	1.013×10^{-7}	2.255×10^{-1}
	GP-PSAW	1.260×10^{-7}	2.485×10^{-3}	2.460×10^{-2}	1.115×10^{-7}	2.4560×10^{-1}

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.

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 Figure 3 together with there uncertainty interval [7]. 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.

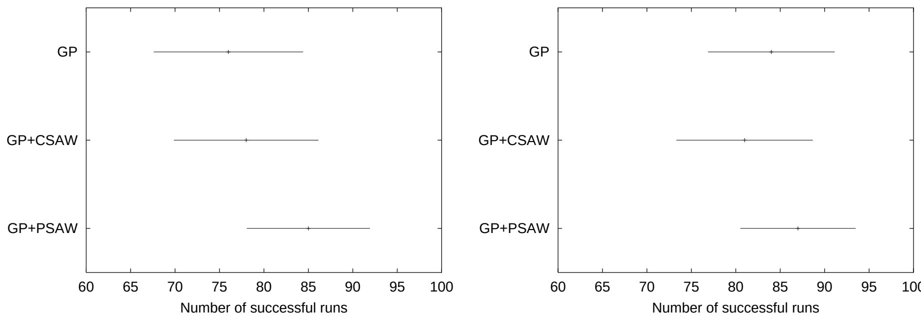

Figure 3: Number of successful runs with uncertainty intervals in experiment 4 for the quintic polynomial (left) and the sextic polynomial (right)

5 Conclusions and Future Research

We have presented 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. [2].

Although we show that in most cases our new variant PSAW provides the best solutions on average we are not able to get results that provide a significant improvement compared to standard GP. We note that this is a preliminary phase of the study on symbolic regression using the SAW technique and we hope to get more conclusive results by increasing the number of independent runs and functions.

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 [6] that adapts the set of points we need to fit. Creating an arms-race between a population of solutions and a population of sets of sample points.

Acknowledgements

We thank Maarten Keijzer for sending us his GP kernel. This kernel is part of the Data to Knowledge research project. More information on this project can be found on `http://project.dhi.dk/d2k`.

References

[1] 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.

[2] 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.

[3] 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.

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

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

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

[7] M.L. Wijvekate. *Onderzoekmethoden.* Het Spectrum, 1992.