Contents

Chapter 1

IMPROVED TIME SERIES PREDICTION AND SYMBOLIC REGRESSION WITH AFFINE ARITHMETIC

Cassio Pennachin[1], Moshe Looks[2] and J. A. de Vasconcelos[1]
[1]*Universidade Federal de Minas Gerais, Belo Horizonte, MG Brazil;*
[2]*Google Inc., Mountain View, CA 94043 USA.*

Abstract We show how affine arithmetic can be used to improve both the performance and the robustness of genetic programming for problems such as symbolic regression and time series prediction. Affine arithmetic is used to estimate conservative bounds on the output range of expressions during evolution, which allows us to discard trees with potentially infinite bounds, as well as those whose output range lies outside the desired range implied by the training dataset. Benchmark experiments are performed on 15 symbolic regression problems as well as 2 well-known time series problems. Comparison with a baseline genetic programming system shows a reduced number of fitness evaluations during training and improved generalization on test data, completely eliminating extreme errors. We also apply this technique to the problem of forecasting wind speed on a real world dataset, and the use of affine arithmetic compares favorably with baseline genetic programming, feedforward neural networks and support vector machines.

Keywords: Symbolic regression, time series prediction, wind forecasting, affine arithmetic, robustness

1. Introduction

Symbolic regression is an early and important application of Genetic Programming (GP). However, its practical applicability depends on avoiding overfitting to training data. While completely eliminating overfitting isn't feasible in most scenarios, we should at least minimize its probability and extent. A particularly damaging kind of overfitting is the error introduced when GP induces expressions containing asymptotes on points in parameter space that are not part of the training data. Out-of-sample evaluation of points lying near an

asymptote can result in extremely high errors, which makes prevention of this form of overfitting especially important.

Numerous techniques have been proposed to make symbolic regression more reliable, including the use of ensembles, scaling program tree outputs (Keijzer, 2004), penalty functions or multi-objective optimization to reduce complexity and bloat (Smits and Kotanchek, 2004), interval methods (Keijzer, 2003), and combinations of multiple techniques (Kotanchek et al., 2007; Korns, 2009). We focus on the application of affine arithmetic, an interval method, to eliminate potentially harmful trees from the population during evolution.

Interval methods, also known as self-validated numerics, are techniques for numerical computation in which approximate results are produced with guaranteed ranges, or error bounds. They can be used to analyze expressions, such as the ones defined by program trees in a symbolic regression evolution run, and will produce output bounds for those expressions. If a program tree includes asymptotes, interval methods will detect them. This process of static analysis is used in (Keijzer, 2003) and in (Kotanchek et al., 2007) to significantly improve the generalization performance of symbolic regression, by assigning infinite error to program trees with potential asymptotes.

Since interval methods provide guaranteed bounds on the output range of a program tree, another potential use is the detection of well-formed trees which, despite having finite bounds, are guaranteed to have bad fitness. A desirable output range can be estimated from the training data, and trees whose output falls outside the desirable range can be eliminated without complete fitness evaluation. Because GP applied to symbolic regression problems typically generates many trees with very bad fitness (Langdon and Poli, 2002), this kind of elimination can significantly boost performance. Unfortunately, the intervals output by interval arithmetic, the simplest interval method and the one used in (Keijzer, 2003), tend to be too wide to be useful for this purpose.

In (Pennachin et al., 2010), we proposed a system similar to Keijzer's, but based on affine arithmetic, a more refined interval method that generates tighter, although still conservative, bounds for expressions. Our technique uses the affine arithmetic bounds to detect and discard program trees whose outputs lie outside a desirable range. Benchmark results for symbolic regression show that we managed to completely avoid extreme errors on previously unseen test data, and obtained much improved generalization when compared with baseline GP or an interval arithmetic-based system such as Keijzer's.

This paper is an extended version of our work in (Pennachin et al., 2010), and we apply the same system to time series prediction problems. We begin with a summary review of affine arithmetic in the next section. Section 3 outlines our system that incorporates affine arithmetic in GP evolution for fitness estimation and asymptote detection. We review results for 15 symbolic regression benchmark problems in Section 4, and present new results for time series

prediction, covering 2 well-known benchmark datasets and a real world wind speed forecasting problem in Section 5, where we compare our system with well-known machine learning algorithms as well as baseline GP.

2. Interval Methods and Affine Arithmetic

The simplest and most widely adopted interval method is interval arithmetic (IA) (Moore, 1966). In IA, a quantity x is represented by an *interval* $\bar{x} = [\bar{x}_L, \bar{x}_U]$, where $\bar{x}_L, \bar{x}_U \in \Re$ are the interval bounds, and $\bar{x}_L \leq \bar{x}_U$. Arithmetic operations are then based on the interval bounds. For instance:

$$\begin{aligned}
\bar{x} + \bar{y} &= [\bar{x}_L + \bar{y}_L, \bar{x}_U + \bar{y}_U] \\
\bar{x} - \bar{y} &= [\bar{x}_L - \bar{y}_U, \bar{x}_U - \bar{y}_L] \\
\bar{x} \times \bar{y} &= [min(\bar{x}_L\bar{y}_L, \bar{x}_L\bar{y}_U, \bar{x}_U\bar{y}_L, \bar{x}_U\bar{y}_U), \\
&\qquad max(\bar{x}_L\bar{y}_L, \bar{x}_L\bar{y}_U, \bar{x}_U\bar{y}_L, \bar{x}_U\bar{y}_U)] \\
e^{\bar{x}} &= [e^{\bar{x}_L}, e^{\bar{x}_U}]
\end{aligned}$$

The major downside of IA is that sequences of operations tend to accumulate and magnify errors. Since IA interval ranges are guaranteed to contain the actual value of an expression over the given input ranges, they can never be too narrow. As they ignore dependencies between variables, they often tend to be too wide. As a simple, but pathological case, consider $\bar{x} - \bar{x}$, where $\bar{x} = [-1, 1]$. The resulting interval is $[-2, 2]$, while clearly the correct value is $[0, 0]$. Repeated application of error-inducing operations during expression analysis tends to result in error propagation and magnification.

Several extensions and improvements to interval arithmetic have been proposed. Affine arithmetic (AA) is an interval method originally developed for computer graphics (Comba and Stolfi, 1993), which has found diverse applications, including robust optimization (de Figueiredo and Stolfi, 1997; de Figueiredo and Stolfi, 2004). Affine arithmetic keeps track of correlations between quantities, in addition to the ranges of each quantity. These correlations lead to significantly reduced approximation errors, especially over long computation chains that would lead to error explosion under IA. We now provide a brief overview of affine arithmetic, recommending the above references for further details.

In AA, a quantity x is represented by an *affine form* $\hat{x}$, which is a first-degree polynomial:

$$\hat{x} = x_0 + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n$$

where x_0 is called the central value of $\hat{x}$, the x_i are finite numbers, called partial deviations, and the ε_i are called *noise symbols*, whose values are unknown but assuredly lie in the interval $\mathbb{U} = [-1, 1]$. Each noise symbol corresponds to one component of the overall uncertainty as to the actual value of x. This uncertainty

may be intrinsic to the original data, or introduced by approximations in the computation of $\hat{x}$.

The fundamental invariant of affine arithmetic ensures that there is always a single assignment of values to each ε_i that makes the value of every affine form $\hat{x}$ equal to the true value of the corresponding x. All of these values assigned to ε_is are of course constrained to lie in the interval $\mathbb{U}$.

Correlation is accounted for in AA by the fact that the same symbol ε_i can be part of multiple affine forms created by the evaluation of an expression. Shared noise symbols indicate dependencies between the underlying quantities. The presence of such shared symbols allows one to determine joint ranges for affine forms that are tighter than the corresponding intervals of interval arithmetic.

Computations with AA involve defining, for each operation, a corresponding procedure that operates on affine forms and produces a resulting affine form. Therefore, an elementary function $z \leftarrow f(x, y)$ is implemented by a procedure $\hat{z} \leftarrow \hat{f}(\hat{x}, \hat{y})$, such that given

$$\hat{x} = x_0 + x_1\varepsilon_1 + \cdots + x_n\varepsilon_n$$
$$\hat{y} = y_0 + y_1\varepsilon_1 + \cdots + y_n\varepsilon_n$$

$\hat{f}$ produces some

$$\hat{z} = z_0 + z_1\varepsilon_1 + \cdots + z_m\varepsilon_m$$

for $m \geq n$, preserving as much information as possible about the constraints embedded in the noise symbols that have non-zero coefficients.

When f is itself an affine function of its arguments, $\hat{z}$ can be obtained by simple rearrangement of the noise symbols. This gives us addition, subtraction, and negation. Formally, for any given $\alpha, \beta, \zeta \in \Re$, $z \leftarrow \alpha x + \beta y + \zeta$:

$$\hat{z} = (\alpha x_0 + \beta y_0 + \zeta) + (\alpha x_1 + \beta y_1)\varepsilon_1 + \cdots + (\alpha x_n + \beta y_n)\varepsilon_n$$

With the exception of rounding errors, the above formula captures all the information available about x, y, and z, so it will introduce almost no error in the uncertainty estimates. Accordingly, the above formula will produce tight bounds for operations such as $\hat{x} - \hat{x}$, and identities such as $(\hat{x} + \hat{y}) - \hat{y}$ hold according to this formula, unlike in interval arithmetic. This rule of combination can be trivially extended to any number of inputs.

In the case of $z \leftarrow f(x, y)$ when f is not an affine function, z is given by $f^*(\varepsilon_1, \ldots, \varepsilon_n)$, where f^* is a non-affine function from $\mathbb{U}^n$ to $\Re$. This is the case for important functions such as multiplication, division, exponentiation, logarithm, and others.

In this case, z cannot be expressed exactly as an affine combination of the noise symbols. Instead, we need to find some affine function

$$f^a(\varepsilon_1, \ldots, \varepsilon_n) = z_0 + z_1\varepsilon_1 + \cdots + z_n\varepsilon_n$$

which is a reasonable approximation of f^* over $\mathbb{U}^n$, and then introduce an extra term $z_k\varepsilon_k$ that represents the approximation error incurred when using f^a. The noise symbol ε_k has to be exclusive to this operation; it cannot be shared with other affine forms. The magnitude of its coefficient z_k must be an upper bound on the approximation error, i.e.,

$$|z_k| \geq max\{|f^*(\varepsilon_1, \ldots, \varepsilon_n) - f^a(\varepsilon_1, \ldots, \varepsilon_n)|, \varepsilon_1, \ldots, \varepsilon_n \in \mathbb{U}\}$$

The choice of the affine approximation f^a is an implementation decision, and there are many possibilities. For purposes of simplicity and efficiency, one can consider approximations that are affine combinations of the inputs $\hat{x}$ and $\hat{y}$, so $f^a(\varepsilon_1, \ldots, \varepsilon_n) = \alpha\hat{x} + \beta\hat{y} + \zeta$. Using affine approximations gives us only $k + 1$ parameters to estimate for functions of k inputs.

These parameters have to be chosen in order to minimize the approximation error term $|z_k|$. The best choice, therefore, is arguably the one that minimizes the maximum value of $|\alpha\hat{x} + \beta\hat{y} + \zeta - f(x, y)|$ over the joint range $\langle\hat{x}, \hat{y}\rangle$. This is called the *Chebychev or minmax affine approximation* of f over $\langle\hat{x}, \hat{y}\rangle$; it guarantees that the volume of the joint range is minimized. This, however, does not always minimize the range for $\hat{z}$ alone. A minimum range approximation will do that. Also, the Chebychev approximation may introduce undesirable values in the range of $\hat{z}$, such as negative values for the exponential function, which are avoided by the minimum range approximation. The choice between the Chebychev approximation and the minimum range approximation is application-dependent, although the latter is usually easier to compute.

One does not necessarily have or want to use the best affine approximation, however. A well-known example is the case of multiplication, for which computing the best approximation is expensive, but a simple and efficient heuristic is often used in its place, with an error at most four times greater than the best affine approximation. Detailed presentations of the approximations for a number of non-affine operations are provided in (de Figueiredo and Stolfi, 1997), including pseudo-code and implementation discussions.

Affine arithmetic typically gives tighter bounds than interval arithmetic, in some cases dramatically so, as we have shown by example. However, it can also sometimes lead to *worse* bounds. For example, the product of two non-negative forms, when computed with the usual heuristic, may need to have a negative lower bound in order to preserve the (linear) dependency information while respecting the constrained (i.e., center-symmetric and convex) affine form representation. To see how this can be problematic for our GP application, consider an expression such as $log(x * y)$; we need to be able to accept such expression as valid when x and y are known to be strictly positive.

In order to handle these cases, our implementation of AA is hybridized with standard IA, as suggested in (de Figueiredo and Stolfi, 1997). Affine forms are extended by the presence of independent minimal and maximal values, which

by default are set to their AA minima and maxima. When a non-affine operation is computed, its minima and maxima according to IA are computed as well, and the tightest possible bounds are kept and exploited by AA to allow expressions that would otherwise be disallowed. This is achieved by maintaining a residual error term, which is adjusted after each operation to be the smallest needed to cover the combined (minimal) interval - see (Pennachin et al., 2010) for full details. The hybrid model produces tighter bounds than either AA or IA could attain independently, and is not significantly costlier than plain AA to compute.

Extensions to standard affine arithmetic have been proposed to make it even less conservative (Messine, 2002; Shou et al., 2003), including the use of quadratic or matricial forms to supplement the original affine forms and provide non-linear approximations for functions such as multiplication. However, these extensions introduce significant computational overhead and implementation complexity in return for incremental improvements on accuracy, and have not been explored in our work.

3. Genetic Programming with Affine Arithmetic

Static Analysis of Program Trees

When a program tree is generated, it can be analyzed by recursively applying the elementary operations of affine arithmetic, given interval representations of its inputs. For symbolic regression and time series prediction, these intervals are estimated from the training data, and this analysis will generate conservative bounds for the program output over the region of the input space covered by the training data. These bounds can be infinite, in the case of trees containing possible asymptotes. Such trees are then eliminated from the population. This is the approach used in (Keijzer, 2003), in combination with linear scaling of program tree outputs, to reduce generalization error in symbolic regression. The same technique was subsequently applied to the estimation of phytoplankton concentration in oceans from satellite spectrometer measurements (Valigiani et al., 2004).

Note, however, that due to the conservative nature of IA, some harmless expressions that do not compute functions with asymptotes may nonetheless be eliminated. For example, given $\bar{x} = [-2, 2]$, IA gives an infinite interval for the harmless expression $1/(1 + x^2)$ because the interval computed for the denominator straddles zero. Affine arithmetic is less conservative, and correctly handles the above example and many others.

Proposed System Overview

We utilize a simple, generational GP system in our experiments, based on Koza's classic configuration (Koza, 1992). This baseline GP system has pro-

tected operators for division and logarithm, as well as random ephemeral constants. Parents are chosen for reproduction via tournament selection. New expressions are created via subtree crossover (probability 0.9) and mutation (probability 0.1). Elitism preserves the top 10% of each generation without modification. The initial population is created via the ramped half-and-half method with a maximal depth of 6.

This baseline system can be extended by the integration of a number of enhancements:

Interval Arithmetic As in (Keijzer, 2003), we perform static analysis of expressions through interval arithmetic to detected and reject trees containing asymptotes.

Affine Arithmetic for Asymptote Rejection Same as above, but substituting affine arithmetic for interval arithmetic.

Affine Arithmetic for Output Bounds Rejection We determine bounds for each expression, rejecting those that are outside the desirable range, defined as $r(Y^T) = [min(Y^T) - \alpha r,\ max(Y^T) + \alpha r]$, where Y^T are the target outputs for all training cases, $r = (max(Y^T) - min(Y^T))/2$, and α is a parameter.

Linear Scaling As in (Keijzer, 2003). Given an expression producing outputs Y for n training cases X, and with Y^T being the target output values, a linear regression can be performed as:

$$b = \frac{\sum_{i=1}^{n}(y_i^T - \tilde{Y}^T)(y_i - \tilde{Y})}{\sum_{i=1}^{n}(y_i - \tilde{Y})}$$
$$a = \tilde{Y}^T - b\tilde{Y}$$

where $\tilde{Y}$ and $\tilde{Y}^T$ denote the average output and average desired output, respectively. Replacing outputs by $a + bY$ minimizes mean squared error with respect to Y^T. Trees whose output variance is greater than 10^7 or lower than 10^{-7} are discarded.

We purposefully avoid introducing common improvements, such as measures to penalize bloat, to more easily observe the impact of our proposed improvements. We believe a complexity penalty would benefit both the baseline system and our extensions to an equal or similar extent.

We implement the baseline system as well as the enhancements in Common Lisp, as part of the PLOP open source project (Looks, 2010).

4. Symbolic Regression Experiments

We test our system on the same set of 15 benchmark problems used in (Keijzer, 2003), summarized in Table 1-1. Parameters are chosen to enable direct comparison with (Keijzer, 2003), and were not tuned. See (Pennachin et al., 2010) for more details on experimental setup and results. We summarize results for the following combinations of enhancements described above:

Base Baseline GP system with protected operators.

IA Linear scaling and interval arithmetic for asymptote rejection as in (Keijzer, 2003).

AA Linear scaling and affine arithmetic for asymptote rejection.

AA+ Linear scaling and affine arithmetic for asymptote rejection, plus output bounds rejection.

In terms of wall clock time, the baseline system is fastest, as expected, while **IA** introduces a modest penalty. **AA** and **AA+** are significantly more expensive, which suggests that their application is best suited for hard problems that the other systems can't solve effectively, as well as problems for which the fitness function is expensive to compute, thus justifying the overhead imposed by static analysis.

In terms of generalization ability, we use the Normalized Root Mean Square Error (NRMS) as a performance measure:

$$NRMS = \frac{\sqrt{\frac{n}{n-1}MSE}}{\sigma_{Y^T}}$$

where n is the number of training (or test) cases, σ_{Y^T} is the standard deviation of desired outputs for those cases, and MSE is the mean squared error. An expression that always produces the mean desired output will have an NRMS of 1, while a perfect expression will have an NRMS of 0.

We measure the NRMS of the best evolved expression in each run over the test set and compare these values with the training NRMS values. Figure 1-1 shows the ratio between test and train NRMS[1] for each problem, as an indication of generalization quality. The baseline system has the worst generalization performance, followed by **IA**, while **AA+** outperforms all other configurations, especially on the last five problems, which are harder.

We also look at absolute test error. As an NRMS value of 1 over the test set implies performance no better than constantly outputting the mean desired value, we use this value as a threshold for overfitting. The baseline configuration

[1]Test NRMS was capped at 1000 to minimize disruption caused by ill-formed exceptions in the baseline system.

Table 1-1. Symbolic Regression benchmarks. The last two columns describe how training and test data are generated. Datasets can be uniformly spaced (notation *[lower:step:upper]*) or a random sample of *n* uniformly distributed points (notation $r_n(lower, upper)$). For multidimensional problems, the *[lower:step:upper]* notation denotes an n-dimensional mesh, and hence $((upper - lower)/step)^n$ total data points.

Prob.	Equation	Train	Test
1		$[-1 : 0.1 : 1]$	$[-1 : 0.001 : 1]$
2	$f(x) = 0.3xsin(2\pi x)$	$[-2 : 0.1 : 2]$	$[-2 : 0.001 : 2]$
3		$[-3 : 0.1 : 3]$	$[-3 : 0.001 : 3]$
4	$f(x) = x^3e^{-x}cos(x)sin(x)(sin^2(x)cos(x) - 1)$	$[0 : 0.05 : 10]$	$[0.05 : 0.05 : 10.05]$
5	$f(x, y, z) = \frac{30xz}{(x-10)y^2}$	$x, z = r_{10^3}(-1, 1)$ $y = r_{10^3}(1, 2)$	$x, z = r_{10^4}(-1, 1)$ $y = r_{10^4}(1, 2)$
6	$f(x) = \sum_i^x 1/i$	$[1 : 1 : 50]$	$[1 : 1 : 120]$
7	$f(x) = logx$	$[1 : 1 : 100]$	$[1 : 0.1 : 100]$
8	$f(x) = \sqrt{x}$	$[0 : 1 : 100]$	$[0 : 0.1 : 100]$
9	$f(x) = \text{arcsinh}(x)$	$[0 : 1 : 100]$	$[0 : 0.1 : 100]$
10	$f(x, y) = x^y$	$x, y = r_{100}(0, 1)$	$[0 : 0.1 : 1]$
11	$f(x, y) = xy + sin((x - 1)(y - 1))$		
12	$f(x, y) = x^4 - x^3 + y^2/2 - y$		
13	$f(x, y) = 6sin(x)cos(y)$	$x, y = r_{20}(-3, 3)$	$[-3 : 0.01 : 3]$
14	$f(x, y) = 8/(2 + x^2 + y^2)$		
15	$f(x, y) = x^3/5 + y^3/2 - y - x$		

produces overfit expressions in 13 problems, while **IA** and **AA** suffer from overfitting in 5 and 4 problems, respectively. **AA+** outperforms all alternatives, producing overfit trees in 1% of the runs for problem 15 and 4% of the runs for problem 13, for which the baseline system is overfit in 98% of the runs - again, see (Pennachin et al., 2010) for full details.

5. Time Series Prediction

Benchmark Problems

Our initial experiments with time series prediction use 2 well-known datasets. The sunspots time series (Weigend et al., 1992) measures the yearly count of sunspots since 1700. It is typically split into a training set covering the period from 1700 to 1920, and 2 separate test sets, the first one from 1921 to 1955, and the second one from 1956 to 1979. We repeat this split. The time series is linearly normalized to fit in the interval $[0, 1]$ before learning. The Mackey-Glass dataset (Glass and Mackey, 2010) is derived from a time-delay ordinary differential equation, and one typically discards the first 3500 points in the time

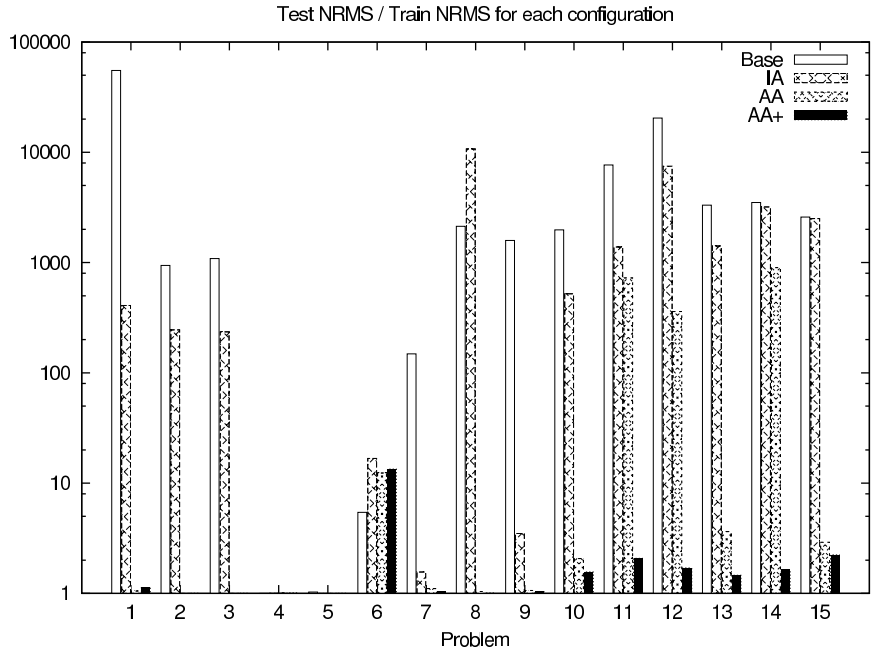

Figure 1-1. Test NRMS divided by train NRMS for each problem (log scale).

series, using the following 1000 points for training and the final 500 points for testing. No normalization is performed.

In our experiments we use a population size of 2000 individuals, allow up to 500,000 fitness evaluations, set the maximal program depth at 12, and use the following function vocabulary, with protected operators where appropriate: $+, -, \times, \div, exp, log, sin, cos, sqrt, 1/x, -x$. Inputs include the past L values of the time series up to x_t, where the desired output is the one-step lookahead value x_{t+1} and L is a lookback parameter, set to 5 for the sunspot series and to 8 for the Mackey-Glass series, according to existing literature. Other parameters are as described in Section 3.

Results are averaged over 50 runs and we compare the baseline GP system with a configuration that uses linear scaling and affine arithmetic for asymptote rejection, plus output bounds rejection (the equivalent of **AA+** in the previous Section). Results using only linear scaling (not shown) are not statistically significant.

Table 1-2 presents results for the sunspot series. We show the typical performance measure often reported for this time series, defined as $E = MSE/\sigma^2$, where MSE is the mean squared error and $\sigma^2 \sim 0.041$ is the variance over the entire dataset. Typically, generalization is easy in the first test set of the sunspot series, while it is hard on the second set. We see that in our results, where the

baseline system does well in the first subset, but is outperformed in the second one, while also presenting much higher variance which is a consequence of the strong overfitting observed in some runs. The **AA+** configuration, while displaying a modest degree of overfitting, is much more robust. Also, this configuration rejects approximately 13% of expressions created during each run.

Table 1-3 presents results for the Mackey-Glass series. We show RMS (root mean squared error) and NRMS (as defined in the previous Section), as well as *p*-values for a two sample Kolmogorov-Smirnov test over the test set. While both configurations show little evidence of overfitting, the **AA+** version has much better performance on both training and test data, and much lower variance (of results); it also rejects approximately 17.5% of the expressions generated during each run (around 7.3% being rejected for containing potential asymptotes, and the remaining 10.2% for output bounds outside the desirable range). Figure 1-2 shows error over the test set for both configurations.

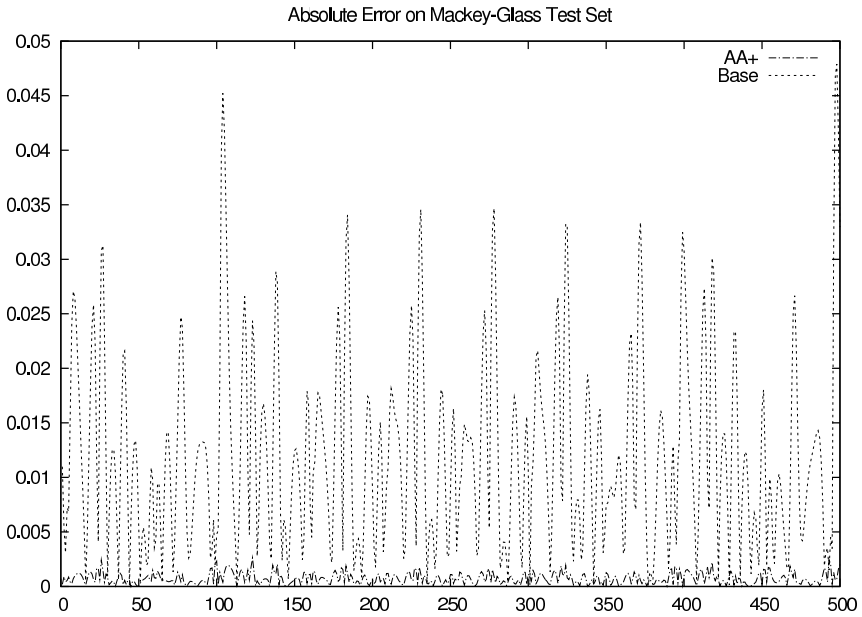

Figure 1-2. Mackey-Glass test set error for both configurations.

Table 1-2. Sunspot prediction results and Kolmogorov-Smirnov two sample test *p*-values.

Cfg	**Train**	**Test 1**	**Test 2**
Base	0.102 ± 0.014	0.115 ± 0.020	1.011 ± 1.483
AA+	0.097 ± 0.008	0.113 ± 0.013	0.427 ± 0.144
p-value		0.095	$\mathbf{0.001}$

Table 1-3. Mackey-Glass prediction results and Kolmogorov-Smirnov two sample test *p*-values.

Cfg	**Train**		**Test**	
	RMS	NRMS	RMS	NRMS
Base	0.016 ± 0.013	0.068 ± 0.053	0.016 ± 0.013	0.070 ± 0.055
AA+	0.002 ± 0.001	0.012 ± 0.005	0.002 ± 0.001	0.012 ± 0.005
p-value			$\mathbf{1.087 \times 10^{-13}}$	

Wind Speed Forecasting

Wind provides a non-polluting renewable energy source, and it has been growing in importance over the past several years as countries attempt to minimize their dependency on fossil fuels and reduce their emissions. However, wind power production is highly volatile, as it depends on wind speed, which is determined by a number of weather factors. Therefore, wind speed prediction is an important problem in the clean energy world, and it also has applications for the wider energy production and market sectors (Lei et al., 2009).

One can perform wind speed forecasts on multiple time scales, with different application purposes (Soman et al., 2010). Long-term prediction can be used as an aid in wind turbine location decisions. Short term prediction on daily data can be used to estimate wind power generation over the next 24 to 48 hours, and those estimates are used to determine how much power will be needed from traditional sources. Finally, very short term prediction, over minutes up to a couple of hours, is useful for energy auctions as well as electrical grid operation, as wind speed is a major factor in overhead transmission line ampacity,[2] and forecasts can be used to guide dynamic rating systems (Michiorri and Taylor, 2009). A number of statistical and machine learning techniques have been applied to this problem, such as neural networks and autoregressive integrated moving average (ARIMA) models.

Herein we focus on short term, daily prediction. We run experiments on two wind speed time series, corresponding to the weather stations in the southern Brazilian cities of Curitiba (station code 838400) and Florianopolis (station code 838990), respectively.[3] We use data from the years 2008 and 2009 for training, and test on data covering the first 11 months of 2010. Both series contain missing entries, and we handle that by replicating the previous observation.

We note that wind speed forecasting is a very complex problem, and one shouldn't hope to build accurate models using only past wind speed as inputs. Several climatological factors have as much or more influence on one-step looka-

[2]Ampacity is the maximal current a transmission line can carry without being damaged.

[3]These time series were obtained from the US National Climatic Data Center (NCDC), and are part of the Global Surface Summary of Day Data (GSOD) base, avaliable via `http://www.ncdc.noaa.gov/cgi-bin/res40.pl?page=gsod.html`

head wind speed, such as temperature, pressure, humidity, as well as seasonal factors and other climate phenomena. In fact, using only past wind speed data can be actively misleading, and lead to very poor out of sample performance. Our goal, therefore, is to provide acceptable out of sample performance, determined as beating a zero-intelligence model that always outputs the mean value of the past data. Such model would have a test set NRMS very close to 1, so the same overfitting test we devised for symbolic regression in Section 4 can be used here. We note that the zero intelligence benchmark is frequently used in the wind forecasting literature (Soman et al., 2010; Lei et al., 2009).

The system configuration is the same as used in the benchmarks presented above. No normalization is performed on the data (preliminary experiments with both linear and zero-mean unit-variance normalization showed no improvements). We also present results from feed-forward neural networks trained via backpropagation and support vector machines using the Gaussian kernel for regression. Parameters for neural nets and SVMs are tuned via cross-validation on the training set. We use code from the WEKA open source project (Hall et al., 2009). We execute 10 runs for each configuration on each dataset.

Table 1-4 presents summary results for both time series. For each series, we show test RMS, as well as the percentage of runs with good generalization (defined as a test NRMS below 1). Given the difficulty of the problem, the higher RMS values are not unexpected. We see that the baseline system is competitive with SVMs, while neural networks have the worst performance. The **AA+** configuration is the only one to achieve good generalization on all runs.

Table 1-4. Wind speed prediction results on 2010 data (test set).

Cfg	**St. 838400**		**St. 838990**	
	RMS	% $NRMS < 1$	RMS	% $NRMS < 1$
Base	2.001	60	2.165	20
AA+	1.972	100	2.125	100
SVM	2.070	40	2.203	30
NN	4.023	0	5.149	0

Figure 1-3 shows the actual time series and predicted values on the test set for one of the **AA+** runs.

6. Conclusion

We have developed a GP system that utilizes affine arithmetic to detect and discard program trees that might include asymptotes, and also to estimate the output bounds of these trees and discard those that lie outside a desirable range, determined from training data. Our system greatly improves out of sample performance for symbolic regression problems, when compared with a baseline

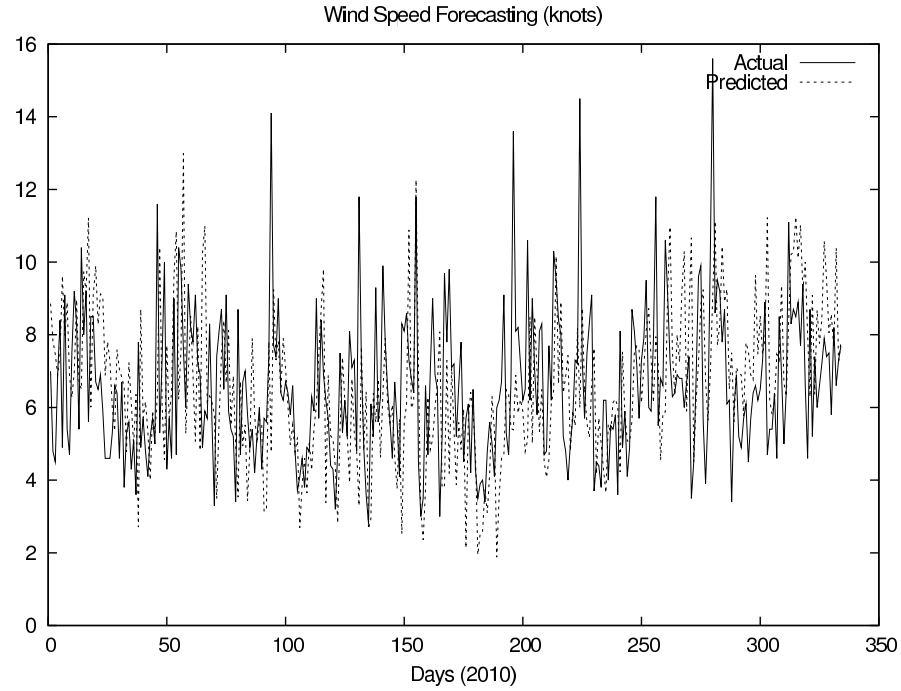

Figure 1-3. Actual and predicted wind speed values over 2010 for the Curitiba weather station.

GP system as well as a similar system that relies on interval arithmetic, a simpler but less accurate estimation method.

We have applied the same system to time series prediction problems. Experiments on the Mackey-Glass dataset show much higher accuracy and lower variance than the ones obtained with baseline GP. Similarly, experiments on the sunspot time series show better generalization and lower variance. Experiments on the much harder wind speed forecasting problem also show improved generalization in comparison with standard GP as well as leading machine learning method such as SVMs and neural networks.

We should note that our improved generalization comes at a price in runtime, as the **AA+** configuration is more computing intensive than the baseline GP system, by a factor of 3-5 on the time series experiments presented here. The extra cost of affine arithmetic evaluation can, however, be vastly reduced by caching the estimated output ranges of subtrees, so only a small fraction of each program tree needs to be re-estimated after crossover or mutation. Another alternative to reduce the computational overhead would be performing static analysis only on a subset of the population, such as individuals selected for recombination.

Future research opportunities include the incorporation of other improvements over standard GP, such as ensembles and complexity penalties, as done in (Kotanchek et al., 2007); devising improved mutation and crossover operators that take into consideration the estimated output ranges of subtrees; pruning operators that eliminate constant or low-variance subtrees, replacing them with new constant input nodes; automated building block identification and preservation through analysis of shared affine arithmetic noise symbols in subtrees; exploring potential limitations of linear scaling (such as scaling-deceptive search spaces) as well as alternative ways to scale program output; and, finally, application to other problem domains such as ranking and supervised classification.

References

Comba, J. and Stolfi, J. (1993). Affine arithmetic and its applications to computer graphics. In *Anais do VI SIBGRAPI*.

de Figueiredo, L. H. and Stolfi, J. (1997). *Self-Validated Numerical Methods and Applications*. Brazilian Mathematics Colloquium monographs. IMPA, Rio de Janeiro, Brazil.

de Figueiredo, L. H. and Stolfi, J. (2004). Affine arithmetic: Concepts and applications. *Numerical Algoritms*, 37:147–158.

Glass, L. and Mackey, M. (2010). Mackey-glass equation. *Scholarpedia*.

Hall, Mark, Frank, Eibe, Holmes, Geoffrey, Pfahringer, Bernhard, Reutemann, Peter, and Witten, Ian H. (2009). The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11:10–18.

Keijzer, Maarten (2003). Improving symbolic regression with interval arithmetic and linear scaling. In Ryan, Conor et al., editors, *Genetic Programming, Proceedings of EuroGP'2003*, volume 2610 of *LNCS*, pages 70–82, Essex. Springer-Verlag.

Keijzer, Maarten (2004). Scaled symbolic regression. *Genetic Programming and Evolvable Machines*, 5(3):259–269.

Korns, Michael F. (2009). Symbolic regression of conditional target expressions. In Riolo, Rick L., O'Reilly, Una-May, and McConaghy, Trent, editors, *Genetic Programming Theory and Practice VII*, Genetic and Evolutionary Computation, chapter 13, pages 211–228. Springer, Ann Arbor.

Kotanchek, Mark, Smits, Guido, and Vladislavleva, Ekaterina (2007). Trustable symbolic regression models: using ensembles, interval arithmetic and pareto fronts to develop robust and trust-aware models. In Riolo, Rick L., Soule, Terence, and Worzel, Bill, editors, *Genetic Programming Theory and Practice V*, Genetic and Evolutionary Computation, chapter 12, pages 201–220. Springer, Ann Arbor.

Koza, John R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

Langdon, W. B. and Poli, Riccardo (2002). *Foundations of Genetic Programming*. Springer-Verlag.

Lei, Ma, Shiyan, Luan, Chuanwen, Jiang, Hongling, Liu, and Yan, Zhang (2009). A review on the forecasting of wind speed and generated power. *Renewable and Sustainable Energy Reviews*, 13(4):915 – 920.

Looks, M. (2010). PLOP: Probabilistic learning of programs. `http://code.google.com/p/plop`.

Messine, F. (2002). Extentions of affine arithmetic: Application to unconstrained global optimization. *Journal of Universal Computer Science*, 8(11):992–1015.

Michiorri, A. and Taylor, P.C. (2009). Forecasting real-time ratings for electricity distribution networks using weather forecast data. In *20th International Conference and Exhibition on Electricity Distribution (CIRED)*, pages 854–854.

Moore, R. (1966). *Interval Analysis*. Prentice Hall.

Pennachin, Cassio L., Looks, Moshe, and de Vasconcelos, Joao A. (2010). Robust symbolic regression with affine arithmetic. In Branke, Juergen et al., editors, *GECCO '10: Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 917–924, Portland, Oregon, USA. ACM.

Shou, Huahao, Lin, Hongwei, Martin, Ralph, and Wang, Guojin (2003). Modified affine arithmetic is more accurate than centered interval arithmetic or affine arithmetic. In Wilson, Michael J. and Martin, Ralph R., editors, *10th IMA International Conference on the Mathematics of Surfaces*, volume 2768 of *Lecture Notes in Computer Science*, pages 355–365. Springer.

Smits, Guido and Kotanchek, Mark (2004). Pareto-front exploitation in symbolic regression. In O'Reilly, Una-May, Yu, Tina, Riolo, Rick L., and Worzel, Bill, editors, *Genetic Programming Theory and Practice II*, chapter 17, pages 283–299. Springer, Ann Arbor.

Soman, S.S., Zareipour, H., Malik, O., and Mandal, P. (2010). A review of wind power and wind speed forecasting methods with different time horizons. In *North American Power Symposium (NAPS)*.

Valigiani, Gregory, Fonlupt, Cyril, and Collet, Pierre (2004). Analysis of GP improvement techniques over the real-world inverse problem of ocean color. In Keijzer, Maarten et al., editors, *Genetic Programming 7th European Conference, EuroGP 2004, Proceedings*, volume 3003 of *LNCS*, pages 174–186, Coimbra, Portugal. Springer-Verlag.

Weigend, A. S., Huberman, B. A., and Rumelhart, D. E. (1992). Predicting sunspots and exchange rates with connectionist networks. In Casdagli, M. and Eubank, S., editors, *Nonlinear Modeling and Forecasting*. Addison-Wesley.

Index