Overfitting Detection and Adaptive Covariant Parsimony Pressure for Symbolic Regression[*]

Gabriel Kronberger
Upper Austria University of Applied Sciences
Research Center Hagenberg
Softwarepark 11
4232 Hagenberg, Austria
gabriel.kronberger@fh-hagenberg.at

Michael Kommenda
Upper Austria University of Applied Sciences
Research Center Hagenberg
Softwarepark 11
4232 Hagenberg, Austria
michael.kommenda@fh-hagenberg.at

Michael Affenzeller
Upper Austria University of Applied Sciences
Department of Software Engineering
Softwarepark 11
4232 Hagenberg, Austria
michael.affenzeller@fh-hagenberg.at

ABSTRACT

Covariant parsimony pressure is a theoretically motivated method primarily aimed to control bloat. In this contribution we describe an adaptive method to control covariant parsimony pressure that is aimed to reduce overfitting in symbolic regression. The method is based on the assumption that overfitting can be reduced by controlling the evolution of program length. Additionally, we propose an overfitting detection criterion that is based on the correlation of the fitness values on the training set and a validation set of all models in the population.

The proposed method uses covariant parsimony pressure to decrease the average program length when overfitting occurs and allows an increase of the average program length in the absence of overfitting. The proposed approach is applied on two real world datasets. The experimental results show that the correlation of training and validation fitness can be used as an indicator for overfitting and that the proposed method of covariant parsimony pressure adaption alleviates overfitting in symbolic regression experiments with the two datasets.

Categories and Subject Descriptors

I.2.2 [**Automatic Programming**]: Program synthesis; I.2.8 [**Problem Solving, Control Methods, and Search**]: Heuristic Methods

General Terms

Algorithms, Experimentation

[*]Paper presented at the 3rd Symbolic Regression and Modeling Workshop GECCO 2011

Keywords

Symbolic regression, overfitting, parsimony pressure

1. MOTIVATION

Overfitting is a well-known problem in data-based modeling and it has been shown that symbolic regression is also prone to overfitting [17]. One approach to detect if overfitting occurs in symbolic regression is to use an internal validation set on which all solution candidates are evaluated additionally to the training set. Instead of returning the best solution on the training set, a solution with high fitness on the training as well as the validation set is returned as the final result [5], [21]. This approach can also be extended to store a set of Pareto optimal solutions regarding validation fitness and model size in an archive [15], [11]. The effectiveness of a validation method strongly depends on the observations contained in the validation partition. Both the training and validation partition should contain a representative sample of all possible observations to make it possible to create a model that is also applicable to new observations. Thus the choice of training and validation partitions is crucial for any data-based modeling approach.

Other approaches that are often used to control overfitting and improve generalization in statistical learning methods are based either on the estimation of the expected generalization error or on penalization of overly complex models [6]. The first approach is to tune the algorithm parameters in iterative steps to find parameter settings which result in a model that generalizes well using an estimator for the expected generalization error of the model. The expected generalization error can be estimated using a hold-out set or through cross-validation. The second approach is to add a penalty term to the objective function that should be minimized. The penalty term depends on the model complexity and optionally on the number of training samples [1], [12], [10], [18]. This approach integrates directly into the training algorithm and produces a model with a good balance between training error and complexity.

A topic that has been discussed intensively and is also related to overfitting is the bloating effect of genetic programming [17]. Bloat is the growth of program length without a corresponding improvement in fitness. The most recent theory explaining the cause of bloat is the crossover bias theory of bloat [3]. A number of different methods for the prevention or reduction of bloat have been derived from the crossover bias theory of bloat [9], [13]. One example of a theoretically motivated bloat control method is covariant parsimony pressure [9]. Through covariant parsimony pressure it is possible to tightly control the dynamics of the average program length over a genetic programming (GP) run. In this contribution we take up the idea to dynamically activate and deactivate parsimony pressure depending on the state of the algorithm as has already been suggested in [9].

It is generally assumed that bloat and overfitting are related. However, it has been recently observed that overfitting can occur in absence of bloat, and vice versa. Thus, it has been suggested that overfitting and bloat are two separate phenomena in genetic programming [17], [14]. This suggests that overfitting and bloat should be controlled also by separate mechanisms. In this contribution we do not discuss bloat control methods in detail. However, we use the covariant parsimony pressure method, which was initially proposed for bloat control, to reduce overfitting, because it provides a natural method to control average model length in a GP population.

In this contribution we propose an overfitting detection criterion for symbolic regression. We use this criterion in combination with covariant parsimony pressure to control the average program length. Parsimony pressure is only applied in the overfitting state to gradually reduce model length. As long as no overfitting is detected the program length is allowed to grow.

The aim of the contribution is to present the idea and first experimental results of the proposed method for the reduction of overfitting. As stated in the previous sections there are a number of shortcomings of the approach, that need to be addressed in further research to make it useful in practical applications.

2. OVERFITTING DETECTION FOR SYMBOLIC REGRESSION

Overfitting and thus poor generalization ability can be detected by evaluating solution candidates on an internal validation set. For each solution candidate two fitness values are calculated, namely f_{training} and $f_{\text{validation}}$. Only the fitness on the training set is used for selection. The fitness on the validation set is used to detect the presence of overfitting. An indicator for overfitting is that the fitness on the validation set decreases while the fitness on the training set increases. An overfitting metric that is based on this approach is for example given in [17].

In this contribution a different approach to detect overfitting using a validation set is proposed. Instead of observing the absolute values or relative changes of the validation fitness we calculate the correlation of the fitness on the training set and the fitness on the validation set over all solution candidates in a population at generation g.

Intuitively, when no overfitting occurs the validation fitness of solution candidates should be strongly correlated to the training fitness. Solution candidates with higher training fitness are more likely to be selected for recombination, so ideally such solution candidates should also have high validation fitness. Solution candidates that have low training fitness should also have low validation fitness. If the correlation of training and validation fitness is high the selection pressure implicitly leads to solutions that have better training and validation fitness. In contrast if the correlation of training and validation fitness is low the selection pressure will lead to solution candidates that are only better on the training set and the chance to create solutions with lower validation fitness increases.

This leads to the overfitting detection function shown in Equation 1. The function indicates that overfitting occurs when the correlation of training and validation fitness values of the models in generation g is lower than a certain threshold. In the experiments we used the non-parametric Spearman's rank correlation coefficient $\rho(x, y)$ [16] instead of Pearson's product moment correlation coefficient because we observed that in general the training and validation fitness values do not follow a normal distribution.

$$\text{Overfitting}(g)_\alpha = \begin{cases} \text{true} & \text{if } \rho(f_{\text{training}}(g), f_{\text{validation}}(g)) < \alpha \\ \text{false} & \text{otherwise} \end{cases} \tag{1}$$

It must be noted that the fitness correlation is also small when the algorithm is in an under-fitting stage, for instance at the beginning of a symbolic regression run when both the training and validation fitnesses are low. It is expected that the algorithm starts initially with poorly fit models and a low correlation coefficient which should increase over the first generations. After the algorithm has reached a maximal correlation coefficient after some generations it is then possible to detect overfitting by a decrease in the correlation coefficient.

Instead of calculating the correlation on the whole population it is also feasible to calculate it only for a subset of the population including the models with best training fitness. While models with large training fitness should also have a large validation fitness value, it can be argued that in practice it is not necessary that models with small training fitness should also have a small validation fitness. Additionally, if a lower limit for the fitness value is enforced, many models have a small training and validation fitness at the beginning of a run. The fitness values of these models should not be included in the calculation of the correlation coefficient to prevent biasing the result. In this work we use Pearson's R^2 of the estimated and target values as fitness function, and subsequently we calculate the correlation coefficient over all fitness values larger than zero (i.e. models producing constant output are excluded).

The correlation-based overfitting detection function indicates already very early when the GP process begins to produce over-fitted models. So it is possible to react to overfitting in a timely manner for instance by stopping the run or reducing the model length.

3. OVERFITTING, PROGRAM SIZE, AND COMPLEXITY

Overfitting can be reduced by reducing the complexity of solution candidates in the population. In the following we assume that the complexity of solution candidates can be simultaneously reduced by reducing the average program length in the population. This is a notable shortcoming of the proposed approach, because program length and model complexity cannot be regarded as equivalent in GP [19], [2], [17]. Especially, if the function set includes transcendental functions complexity and program length cannot be related directly (i.e. $\sin(x)$ is more complex and more compact than $x * x * ... * x$). Therefore, the choice of the function set has a strong impact on the tendency for overfitting and also on the applicability of methods controlling program length to reduce overfitting, including the approach described in this work. As a consequence, a very limited function set including only arithmetic functions has been used in the experiments described later in this work. However, the issue of the disconnection of program length and complexity is not completely resolved by this measure. Further research about the relation of complexity, program length and fitness is necessary for improving overfitting control strategies. An important question for further research is if complexity can also be regarded an inherited trait, and if the evolution of complexity follows a similar rule as the evolution of program length.

4. ADAPTIVE CONTROL OF COVARIANT PARSIMONY PRESSURE

One approach for the control of program length in GP is the parsimony pressure method. In this approach the selection probability of smaller individuals is increased by adjusting the fitness value of solutions to integrate the solution length weighted with a static parsimony pressure coefficient c as shown in Equation 2.

$$f_{\text{adjusted}}(x) = f(x) - c\ \ell(x) \tag{2}$$

An extension of this method, namely covariant parsimony pressure (CPP) [9], dynamically adapts the parsimony pressure coefficient $c(t)$ at iteration t as shown in Equation 3 where ℓ is the program length and f is the raw fitness. The advantage of this approach is that it is not necessary to tune the parsimony pressure coefficient manually and it is automatically adapted as the distribution of fitness values and problem lengths in the population evolves. Through CPP it is possible to tightly control the average program length over the whole GP run. Additionally, CPP has strong theoretic fundamentals, because it is derived from GP schema theory.

$$\begin{aligned} f_{\text{adjusted}}(x) &= f(x) - c(t)\ell(x) \\ c(t) &= \frac{\text{Cov}(\ell, f)}{\text{Var}(\ell)}, \end{aligned} \tag{3}$$

It has already been suggested in [9] to dynamically turn CPP on and off at specific stages of the GP run. This suggestion is taken up in this work to introduce a novel approach to reduce overfitting. In combination with the overfitting detection function described in the previous section, it is possible to use CPP to gradually decrease the average program length to a point where either no overfitting occurs. This strongly depends on the observations in the training and validation dataset and it is certainly possible that overfitting occurs even for very small models for instance when the dataset contains many more variables than observations ($k >> N$). In such cases the algorithm converges to a population of very small models. When the algorithm is back in a non-overfitting stage parsimony pressure is turned off again. In this way the evolutionary process can freely increase the solution complexity to a level that is necessary to solve the problem and limit the complexity from above as soon as the process starts to overfit.

Parsimony pressure should be configured in such a way that the average program length is reduced slowly. If parsimony pressure is applied too strongly there is the risk that selection probability is solely determined by program length. This can happen for instance when the average program size is forced to a level which is a lot smaller than the current average program size in one generation. This leads to a situation where the adjusted fitness values are strongly correlated to the program length instead of the raw fitness and the evolutionary process will be mainly driven by the program length.

5. EXPERIMENTS

In this section we describe the experiments we have run to test the effectiveness of the adaptive CPP approach and the overfitting detection function.

5.1 Datasets

For the experiments we selected two publicly available regression datasets.

The first dataset (Chemical-I) has been prepared and published by Arthur Kordon, research leader at the Dow Chemical company for the EvoCompetitions side event of the EvoStar conference 2010. The dataset stems from a real industrial process and contains 1066 observations of 58 variables. The first 747 observations are used for training and validation and the remaining 319 observations are used as a test partition. The values of the target variable are noisy lab measurements of the chemical composition of the product which are expensive to measure. The remaining 57 variables are material flows, pressures, temperatures collected from the process which can be measured easily. The dataset can be downloaded from [1].

The second dataset used in our experiments is the Boston housing dataset from the UCI machine learning repository [4]. The dataset contains 506 observations of 14 variables concerning the housing values in the suburbs around Boston. The first 253 observations are used for training and validation and the remaining 253 observations are used as a test partition. The dataset can be downloaded from [2].

5.2 Algorithm Configuration

To test our approach we have executed experiments with four different variants of tree-based genetic programming for symbolic regression using both datasets. The first variant, that we also used as a starting point for the modified variants, is a rather standard GP configuration (SGP) with tree-based solution encoding, tournament selection, sub-tree swapping crossover, and two mutation operators. The function set contains only arithmetic operators with two operands (+,-,*,/) and three operands (average). The terminal set includes all variables and random constants. Trees are initialized randomly using the probabilistic tree creator (PTC2) [8] where the terminal type (variable or constant) is chosen randomly with equal probability. Constants are initialized to a normally distributed random value with

[1] `http://casnew.iti.upv.es/index.php/evocompetitions/105-symregcompetition`
[2] `http://archive.ics.uci.edu/ml/machine-learning-databases/housing/`

$\mu = 1, \sigma = 1$. One-point mutation sets the function symbol of internal nodes randomly, choosing from all symbols from the function set with equal probability. If one-point mutation is applied to a terminal node representing a variable, a new variable symbol is chosen randomly from all possible symbols with equal probability. If one-point mutation is applied to a terminal node representing a constant, a normally distributed random value ($\mu = 0, sigma = 0.1$) is added to the constant value. The fitness function is the squared correlation coefficient of the model output and the actual values of target variables. Only the final model is linearly scaled to match the location and scale of the target variable [7]. The parameter settings for the SGP algorithm are specified in Table 1.

The second variant is SGP with static constraints for the program length and depth (SGP+static). The same parameter settings specified in Table 1 are used and additionally the static limit for program length is set to 250 nodes and the static limit for the program depth is set to 17 levels.

The third variant of the standard algorithm uses covariant parsimony pressure (SGP+CPP). Covariant parsimony pressure is activated after 10 generations and is configured to keep the average program length at a constant value in expectation as shown in Equation 3. No static size constraints are used in combination with CPP.

The fourth variation of the standard algorithm uses the training- and validation fitness correlation to detect overfitting and adapts covariant parsimony pressure accordingly (SGP+AdaptiveCPP). A boolean variable *is-overfitting* is introduced which is initially set to false. After each iteration the training- and validation fitness correlation is calculated and the *is-overfitting* flag is updated accordingly. To prevent unstable behavior two threshold values $\text{thresh}_{\text{low}}$, $\text{thresh}_{\text{high}}$ are used to toggle the value of the *is-overfitting* variable as shown in Algorithm 1. In the experiments we used $\text{thresh}_{\text{low}} = 0.5$, $\text{thresh}_{\text{high}} = 0.75$.

```
$r \leftarrow \rho(f_{\text{training}}(g), f_{\text{validation}}(g))$;
if is-overfitting $= \textit{false} \wedge r < \text{thresh}_{\text{low}}$ then
    is-overfitting $\leftarrow$ true;
if is-overfitting $= \textit{true} \wedge r > \text{thresh}_{\text{high}}$ then
    is-overfitting $\leftarrow$ false;
```

Algorithm 1: Algorithm for overfitting detection with lower and upper threshold for the training and validation fitness correlation.

In SGP+AdaptiveCPP the covariant parsimony pressure is adapted based on the *is-overfitting* flag as shown in Equation 4. When the algorithm is in an overfitting state the covariant parsimony pressure is adapted to reduce the average program length by five percent each iteration. In contrast, no parsimony pressure is applied in a non-overfitting state. The evolutionary process can evolve the program length without constraints in non-overfitting phases.

Parameter	Value
Population size	2000
Max. generations	100
Parent selection	Tournament, group size = 6
Replacement	generational
	no elitism
Initialization	PTC2 [8]
Max. initial tree size	100
Crossover	Sub-tree swapping
Crossover rate	100%
Mutation	7% One-point
	7% sub-tree replacement
Model selection	Best on validation
Fitness function	R^2 (maximization)
Function set	+, -, *, / (binary)
	average (ternary)
Terminal set	constants, variables

Table 1: Genetic programming parameters for the experiments.

$$\begin{aligned} f_{\text{adjusted}}(x) &= \begin{cases} f(x) & \text{if is-overfitting} = \text{false} \\ f(x) - c(t)\ell(x) & \text{if is-overfitting} = \text{true} \end{cases} \\ c(t) &= \frac{\text{Cov}(\ell, f) - \delta_\mu \overline{f}}{\text{Var}(\ell) - \delta_\mu \overline{\ell}} \\ \delta_\mu &= 0.05\ \overline{\ell} \end{aligned} \tag{4}$$

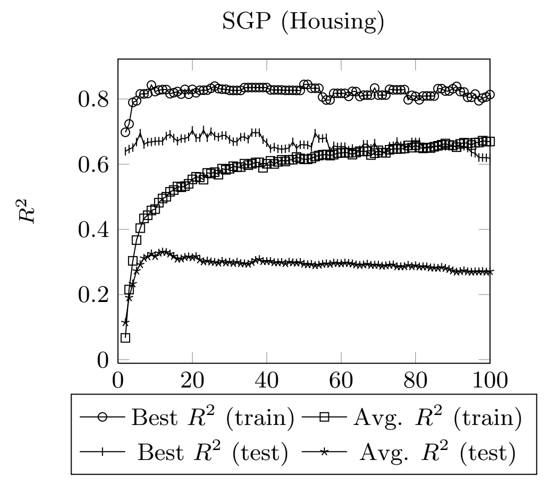
We decided to use tournament selection in all experiments. The selection pressure affects the amount of overfitting and bloat so the results would not be comparable if different selection operators are used. This is slightly problematic because the theory from which CPP is derived assumes proportional selection. However, it has been shown for simple benchmark problems that CPP can be combined with tournament selection [9].

6. RESULTS

We have executed 30 independent runs for each of the four algorithm configurations and both datasets resulting in a total of 240 independent GP runs. All experiments have been executed using HeuristicLab [20], an open source framework for heuristic optimization. Figure 1 shows the development of fitness values on the training and test partitions and the development of the training and validation correlation coefficient as observed in a single SGP run on the housing dataset. The line chart in the top panel shows the trajectories of the best and average fitness on the training partition and on the test partition over 100 generations. The chart shows that there is a slight decrease of the fitness on the test set at the later stages of the GP run. The gradually decreasing test fitness in combination with the increase in training fitness shows that overfitting occurs, even though the effect is not very strong.

The line chart in lower panel shows the trajectory of the correlation of training- and validation fitness for this run. The correlation decreases at a very early point in this run and it can be observed that the turning point is at around the same generation where the average test fitness also starts to decrease in the top panel in Figure 1.

The four panels in Figure 2 show scatter plots of the training and validation fitness of all models in the population at



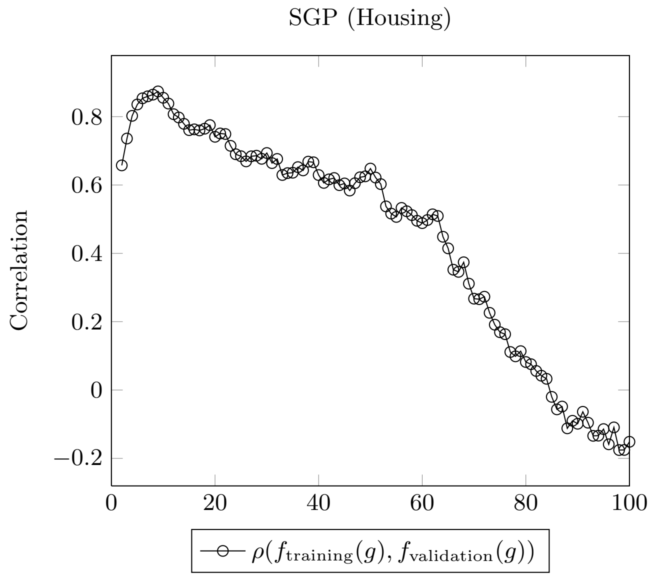


Figure 1: Trajectories of training fitness, test fitness, and $\rho(f_{\mathbf{training}}(g), f_{\mathbf{validation}}(g))$ over 100 generations of a single SGP run on the housing dataset.

specific generations of the same run shown in Figure 1. In the first generation the correlation is rather low, the maximum correlation is reached in generation eight. After generation eight the correlation decreases and it can be observed in the scatter plots that the number of individuals with high training fitness but low validation fitness increases.

Figure 3 shows trajectories of the best fitness on the training set, the test quality of the best training solution, the test quality of the best validation solution, and the training- and validation fitness correlation over 100 generations for all tested algorithms variants and both datasets. The median value over 30 independent GP runs is shown. The best training solution is the solution with the highest fitness found in the run so far. The best validation solution is the solution with the best fitness on the validation set found in the run so far. A notable difference to Figure 1, which shows the overall best test fitness, is that in Figure 3 the test fitness of the finally selected solution, which is either the best on training, or the best on validation, is reported.

The line charts in the top panels show the trajectory of the median best fitness on the training set for all algorithm configurations. It can be observed that the median best fitness on the training set increases for almost all algorithm variants; the only exception is SGP+AdaptiveCPP on the housing problem. SGP without size constraints and SGP with static constraints produce the best solutions regarding the training fitness.

In the second row of Figure 3 the trajectories of the median test fitness of the best training solution are shown. It is important to note that this information is not available to control the GP run. It can be seen first of all that the test fitness is generally lower than the median training fitness. Another observation is that there is not much differences between all algorithm variants. SGP without size limits leads to the worst median test fitness for both datasets. The median test fitness of the training best solution of SGP+AdaptiveCPP is among the best for both datasets.

In the panels in the third row the median test fitness of the validation best solution is shown. This information is also not available in the GP run. Comparing the results shown in the third row with the results shown in the second row, it can be observed that the validation best model has a better test fitness than the training best model for the housing dataset. For this dataset it is beneficial to use an internal validation set for model selection. Again, there median test fitness of the validation best solution produced by all algorithm variants is almost the same for the chemical dataset. For the housing dataset it can be observed that median test fitness of the solution produced by SGP and SGP with static size constraints is lower than the median test fitness produced by the algorithm variants using CPP.

The panels in the fourth row of Figure 3 show the trajectories of the median correlation coefficient of training and validation fitness. The correlation value is also available in the GP run and is used to control parsimony pressure in the SGP+AdaptiveCPP configuration. The charts clearly show that for the SGP runs without size constraints the median correlation coefficient quickly decreases for both datasets. This indicates that overfitting occurs in SGP without size constraints. The median correlation value of the SGP runs with static size constraints, which is the algorithm variant that is most likely used in practice, is the same as the median correlation value of the other algorithm variants for the chemical dataset and only slightly lower for the housing dataset.

In summary, our experiments show that overfitting is not a big issue for the chemical dataset, however, for the housing dataset overfitting can be observed. Additionally, the experiment results show that the algorithm variants perform almost the same. The only exception is SGP without size constraints, which also produces very large and bloated solutions (not shown in the results). The analysis of the correlation of training- and validation fitness seems to work well an indicator of overfitting for the two exemplary regression datasets used in our experiments. This is interesting as the information is available in the GP run and can be used to dynamically adapt the process similarly to the approach we proposed in this contribution. This result is probably the most convincing and encouraging to work on a better way of using this information for controlling the algorithm to reduce the tendency of overfitting.

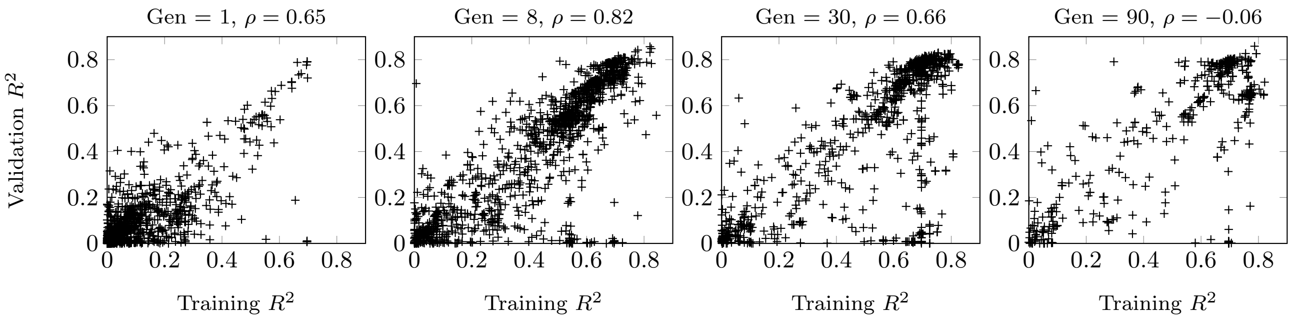


Figure 2: The scatter plots in the lower half show the training vs. validation fitness of all models in the population at four different generations of the same GP run shown in Figure 1.

7. CONCLUSION AND OPEN TOPICS

In this contribution we presented a way to detect overfitting using a validation partition as described in [5] to calculate the correlation of training and validation fitness of all models in the population. We combined this approach with covariant parsimony pressure, which makes it possible to tightly control the average program length in a GP run. While CPP is primarily a bloat control method we used it to reduce overfitting. We described an adaptive approach using CPP to gradually reduce the average program length when overfitting is detected and allow growth of average program length when no overfitting occurs. The idea is that the algorithm should automatically find the correct program length that is needed for accurate modeling without overfitting.

The aim of the contribution is to present the idea and first results of this method for the reduction of overfitting and we are aware of a number of shortcomings of the approach that need to be addressed in further research to make it useful in practical applications. The biggest issue is certainly that the proposed methods tries to control overfitting through control of program length. This is problematic as it is well known that in GP program length and complexity are not equivalent and overfitting is related foremost to program complexity and not program length. The second issue is that in this work we demonstrated the approach only on two different regression datasets. In order to draw strong conclusions it is necessary to test the approach on a larger number of different datasets. And finally a number of additional parameters for adaptive control of CPP must be tuned for the method to work correctly. In the experiments presented in this work we used reasonable default values for these parameters but for practical applications it is necessary to run many more experiments in order to find general rules for setting these parameter values, or, alternatively, robust default settings. All this issues are starting points for further research and a more exhaustive analysis and potentially improvements of the proposed idea.

In the experimental section we tested our algorithm and compared the results to the results achieved with SGP (optionally with static program length and depth constraints), and with SGP with covariant parsimony pressure to hold the program length at a constant level. In our experiments we analyzed the behavior of the different algorithmic variants on two real world regression datasets.

Our experiments show, that, except for the SGP configuration without size constraints, there are no strong differences regarding the overfitting behavior of the four algorithm variants. Thus, no strong conclusions can be derived from the experimental results on the two regression datasets used in our experiments. We also observed that the correlation coefficient of training- and validation fitness seems to indicate overfitting in symbolic regression runs on the two regression datasets used in our experiments. The correlation value is interesting as this information is available in the GP run and can be used to dynamically adapt the process similarly to the approach we proposed in this contribution. We suggest running further experiments with adaptive covariant parsimony pressure on more datasets with different parameter settings. In our experiments we only tested a 5% change rate for the average program length per generation. However, this might be either too drastic or not strong enough to effectively reduce the overfitting.

Another very interesting research question that is potentially fruitful to follow up is the evolution of model complexity in GP. Especially, if model complexity behaves like any other inheritable trait, and thus, can be modelled in a similar way to the evolution of program length. If the evolution of program complexity in relation to program length and fitness is better understood it can become possible to formulate much better methods to control overfitting in symbolic regression.

8. ACKNOWLEDGMENTS

This work mainly reflects research work done within the Josef Ressel-center for heuristic opimization "Heureka!" at the Upper Austria University of Applied Sciences, Campus Hagenberg. The center "Heureka!" is supported by the Austrian Research Promotion Agency (FFG) on behalf of the Austrian Federal Ministry of Economy, Family and Youth (BMWFJ). The authors would like to thank the anonymous reviewers of EuroGP 2011 for their constructive and helpful comments to improve a first version of this paper.

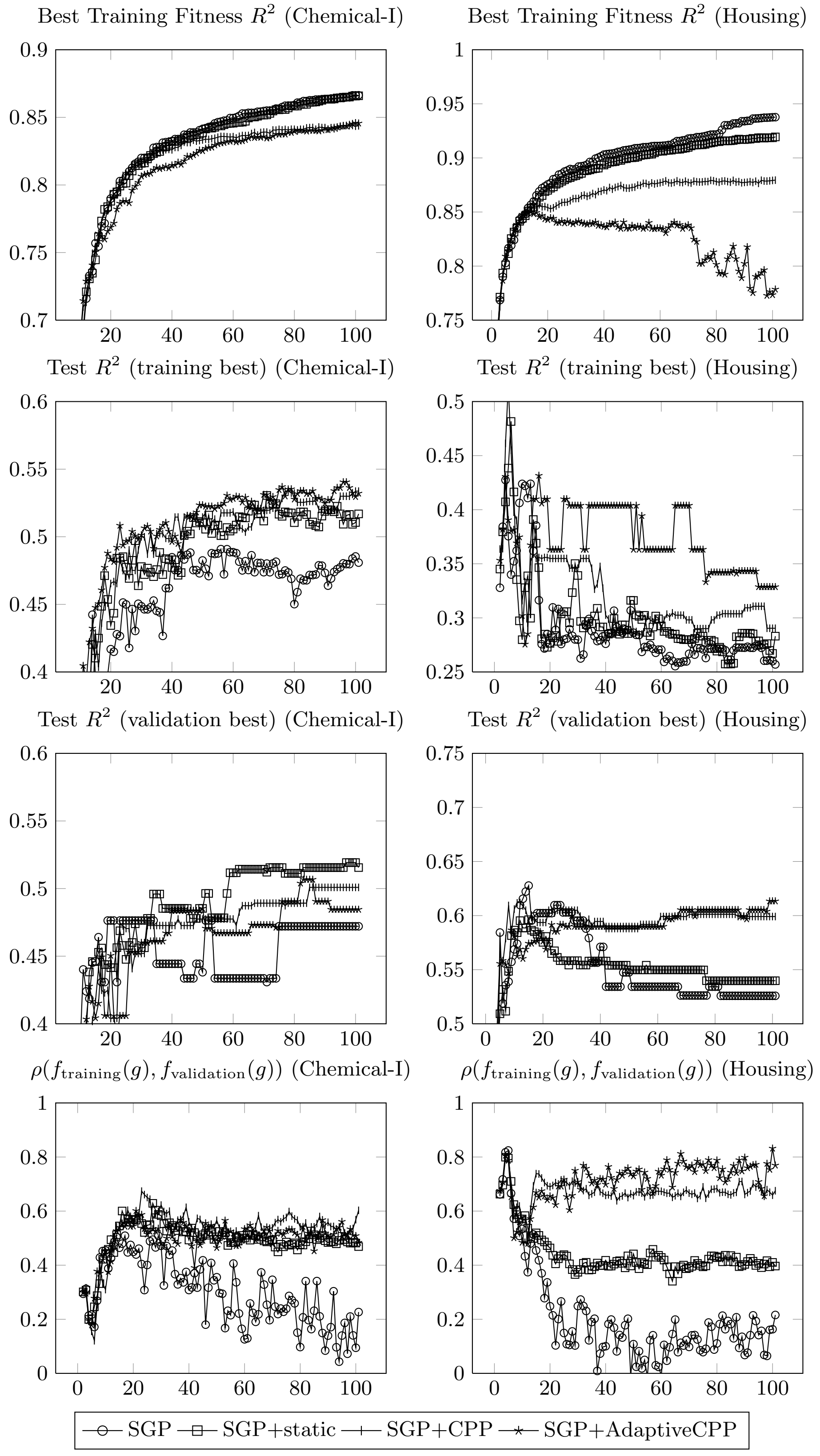


Figure 3: Line charts of best training fitness, test fitness of the best training solution, test fitness of the best validation solution, and training and validation fitness correlation of SGP, SGP with static size limits (SGP+static), SGP with covariant parsimony pressure (SGP+CPP), and SGP with adaptive covariant parsimony pressure (SGP+AdaptiveCPP). Values are median values over 30 independent runs for all 100 generations.

9. REFERENCES

[1] H. Akaike. Information theory and an extension of the maximum likelihood principle. In *Second International Symposium on Information Theory*, pages 267–281. 1973.

[2] R. M. A. Azad and C. Ryan. Abstract functions and lifetime learning in genetic programming for symbolic regression. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, GECCO '10, pages 893–900, New York, NY, USA, 2010. ACM.

[3] S. Dignum and R. Poli. Generalisation of the limiting distribution of program sizes in tree-based genetic programming and analysis of its effects on bloat. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 2, pages 1588–1595, London, 7-11 July 2007. ACM Press.

[4] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[5] C. Gagne, M. Schoenauer, M. Parizeau, and M. Tomassini. Genetic programming, validation sets, and parsimony pressure. In *Genetic Programming, 9th European Conference, EuroGP2006*, volume 3905 of *Lecture Notes in Computer Science*, pages 109–120, Berlin, Heidelberg, New York, 2006. Springer.

[6] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning - Data Mining, Inference, and Prediction*. Springer, 2009. Second Edition.

[7] M. Keijzer. Scaled symbolic regression. *Genetic Programming and Evolvable Machines*, 5(3):259–269, Sept. 2004.

[8] S. Luke. Two fast tree-creation algorithms for genetic programming. *IEEE Transactions on Evolutionary Computation*, 4(3):274–283, Sept. 2000.

[9] R. Poli and N. F. McPhee. Covariant parsimony pressure for genetic programming. Technical Report CES-480, Department of Computing and Electronic Systems, University of Essex, UK, 2008.

[10] J. Rissanen. A universal prior for integers and estimation by minimum description length. *Annals of Statistics*, 11:416–431, 1983.

[11] M. Schmidt and H. Lipson. Symbolic regression of implicit equations. In *Genetic Programming Theory and Practice VII*, Genetic and Evolutionary Computation, pages 73–85. Springer US, 2010.

[12] G. E. Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 6(2):461–464, 1978.

[13] S. Silva and S. Dignum. Extending operator equalisation: Fitness based self adaptive length distribution for bloat free GP. In *Proceedings of the 12th European Conference on Genetic Programming, EuroGP 2009*, volume 5481 of *LNCS*, pages 159–170, Tuebingen, Apr. 15-17 2009. Springer.

[14] S. Silva and L. Vanneschi. Operator equalisation, bloat and overfitting: a study on human oral bioavailability prediction. In *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1115–1122, Montreal, 8-12 July 2009. ACM.

[15] G. F. Smits and M. Kotanchek. Pareto-front exploitation in symbolic regression. In *Genetic Programming in Theory and Practice II*, pages 283–299. Springer, 2005.

[16] C. Spearman. The proof and measurement of association between two things. *The American Journal of Psychology*, 15(1):72–101, 1904.

[17] L. Vanneschi, M. Castelli, and S. Silva. Measuring bloat, overfitting and functional complexity in genetic programming. In *Proc. GECCO'10*, pages 877–884, July 7–11 2010.

[18] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, 1996.

[19] E. J. Vladislavleva, G. F. Smits, and D. den Hertog. Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming. *IEEE Transactions on Evolutionary Computation*, 13(2):333–349, 2009.

[20] S. Wagner. *Heuristic Optimization Software Systems - Modeling of Heuristic Optimization Algorithms in the HeuristicLab Software Environment*. PhD thesis, Institute for Formal Models and Verification, Johannes Kepler University, Linz, Austria, 2009.

[21] S. Winkler, M. Affenzeller, and S. Wagner. Using enhanced genetic programming techniques for evolving classifiers in the context of medical diagnosis. *Genetic Programming and Evolvable Machines*, 10(2):111–140, 2009.