



Title	Tackling overfitting in evolutionary-driven financial model induction
Authors(s)	Tuite, Clíodhna, Agapitos, Alexandros, O'Neill, Michael, Brabazon, Anthony
Publication date	2011
Publication information	Tuite, Clíodhna, Alexandros Agapitos, Michael O'Neill, and Anthony Brabazon. "Tackling Overfitting in Evolutionary-Driven Financial Model Induction." Springer, 2011. https://doi.org/10.1007/978-3-642-23336-4_8 .
Publisher	Springer
Item record/more information	http://hdl.handle.net/10197/3494
Publisher's statement	The final publication is available at springerlink.com
Publisher's version (DOI)	10.1007/978-3-642-23336-4_8

Downloaded 2026-05-01 23:36:12

The UCD community has made this article openly available. Please share how this access benefits you. Your story matters! (@ucd_oa)



© Some rights reserved. For more information

Tackling Overfitting in Evolutionary-Driven Financial Model Induction

Clíodhna Tuite^{1,2}, Alexandros Agapitos^{1,2}, Michael O'Neill^{1,2}, and Anthony Brabazon^{1,3}

¹ Natural Computing Research and Applications Group & Financial Mathematics and Computation Research Cluster, Complex and Adaptive Systems Laboratory, University College Dublin, Ireland

² School of Computer Science and Informatics, University College Dublin, Ireland

³ School of Business, University College Dublin, Ireland

`cliodhna.tuite@gmail.com, alexandros.agapitos@ucd.ie, m.oneill@ucd.ie, anthony.brabazon@ucd.ie`

Summary. This chapter explores the issue of overfitting in grammar-based Genetic Programming. Tools such as Genetic Programming are well suited to problems in finance where we seek to learn or induce a model from data. Models that overfit the data upon which they are trained prevent model generalisation, which is an important goal of learning algorithms.

Early stopping is a technique that is frequently used to counteract overfitting, but this technique often fails to identify the optimal point at which to stop training. In this chapter, we implement four classes of stopping criteria, which attempt to stop training when the generalisation of the evolved model is maximised.

We show promising results using, in particular, one novel class of criteria, which measured the correlation between the training and validation fitness at each generation. These criteria determined whether or not to stop training depending on the measurement of this correlation - they had a high probability of being the best among a suite of potential criteria to be used during a run. This meant that they often found the lowest validation set error for the entire run faster than other criteria.

8.1 Introduction

Overfitting is a commonly studied problem which arises in machine learning techniques such as Genetic Programming (GP). A model is described as overfitting the training data if, despite having a high fit on the training examples, there exists another model which has better fitness on the data as a whole, despite not fitting the training data as well [15].

8.1.1 Causes of Overfitting

There are different reasons why overfitting can occur. For example, the existence of noise in training samples can cause a model to be fit to the data which is more complex than the

true underlying model [19]. For symbolic regression, an example would be fitting a high order polynomial to noisy data, which happens to pass through all training points, when the true function is in fact a lower order polynomial. Another cause of overfitting is bias in the training data. Overfitting is more likely to occur when the training sample size is small. The more training data available, the more likely we are to discover the true underlying model, and the less likely we are to settle on a spurious result. Overfitting is also more likely to occur with complex hypotheses [19]. Learning algorithms that are run for a long time are more likely to trigger overfitting, than if they had been run for a shorter time period [3].

Overfitting is a symptom of experimental setups which produce results that fail to generalise beyond the specific environment in which they were trained. While generalisation has traditionally been underexplored in the GP literature, there have been a number of recent papers examining this important issue [3, 6, 9, 11, 19, 22, 25]. Among the techniques proposed to counteract overfitting, popular examples include the use of parsimony constraints, and the use of a validation set. Parsimony constraints are inspired by Occam's Razor and the minimum description length (MDL) principle. The MDL principle claims that the most preferable solution is the one that minimizes the information required to encode it [25]. However, the direct link between parsimony and better generalisation is disputed in [8]. Domingos [8] notes that overfitting is an unwanted side-effect not of complexity itself, but of considering a large number of potential models. This results in a high chance of discovering a model with a high fitness on the training data *purely by chance*. Evaluating a smaller number of more complex models has a lower chance of causing overfitting than evaluating a larger number of simpler models [8].

Early stopping is a popular technique. When using this approach, generalisation is typically measured by observing the fitness of the evolved model on a validation data set, which is separate from the training data set. Training is stopped if the generalisation of the model on this validation set degrades. In this study, we choose to focus on early stopping in order to illustrate its usefulness as a technique to boost generalisation. Furthermore, we propose some extensions to early stopping as it has traditionally applied, via the inclusion of criteria to determine a more intelligent stopping point. These criteria are inspired by work previously carried out in the neural networks literature [21].

8.1.2 Structure of Chapter

The next section provides some background to the issues explored in this chapter, including a brief review of some of the work on generalisation to date. Section 8.3 moves on to work through and explain some examples of overfitting as observed in symbolic regression problems tackled using Grammatical Evolution, a form of grammar-based GP. Section 8.4 first introduces stopping criteria which can be used to enhance the technique of early stopping, which is used to avoid overfitting. Section 8.5 applies these criteria to trading rule problems tackled using an alternative form of grammar-based GP. Finally, Section 8.6 concludes the chapter.

8.2 Background

GP is a stochastic search and optimisation technique. It operates by first generating a "population" of random solutions, composed from elements of a function and terminal set. The terminal set is composed of a list of external inputs (typically variables), and constants that can appear in the solution. It may also contain functions with no arguments - an example

would be a function which returns a different random number with each invocation. The contents of the function set vary between problem domains - an example of such operators in a simple numeric problem are arithmetic operators. Solutions are iteratively refined over a period of some number of pre-specified generations, using the concept of a fitness function to guide search, and the operators of crossover and mutation to transition across the search space. Crossover works by combining elements of two “parent” solutions, in order to further explore the search space. Mutation randomly alters a part of a randomly selected solution among the population of solutions, in order to promote diversity in the population, and thus explore new areas of the search space. For a more detailed introduction to GP, see [2] or [20].

8.2.1 Previous Attempts to Tackle Overfitting in Genetic Programming

Two Data Sets Methodology

In a 2002 paper, [11] highlights the importance of using a separate testing set alongside a training set when evolving solutions to learning problems. The author also advocates the use of formal guidelines when selecting the training and testing cases. Such guidelines might stipulate rules for selecting training and testing instances to ensure the representativeness of the training cases, and to determine the degree of overlap between the two data sets [11]. In a case of simulated learning (the Santa Fe ant trail), it is shown that training on a given trail by no means guarantees the generalisation of the solution to other trails using the same primitives and rules for trail construction. In order to produce results that generalise to a particular class of trails, [11] conducted experiments using 30 training trails and 70 testing trails, with fitness being evaluated by counting the number of pieces of food gathered over each of the 30 training trails.

Validation Sets and Parsimony

In [22], Thomas and Sycara use a GP-based system to discover trading rules for the Dollar/Yen and Dollar/DM markets. They reserved a particular concern for the aspects of the system that allowed them to fight overfitting, given the inherent noise in financial data. They examined the interactions between overfitting, and both rule complexity and validation methodologies. Excess returns were used as a measure of fitness. The success of the evolved model on the test set was calculated after training had completed. The use of the test set was important given that the focus of this work was on generalization.

The authors thought that while controlling the size of trees could negatively impact on the representational capability of the generated rules, it could also reduce the potential for overfitting, and wished to test this hypothesis. Results showed that between trees with maximum tree-depths of 2, 3 and 5, trees of maximum-depth 2 produced superior excess returns, particularly in the Dollar/Yen case [22].

The authors used a validation set to determine when to cut off search in all experiments - evolution was stopped when the average rule fitness on the validation set started to decrease. In later experiments they used the validation set in two additional ways. After search had stopped, the best rule on the training set was identified and was kept if it produced positive returns on the validation set. If not, it was discarded and search resumed from scratch. A second approach examined the fitness of *every rule* from the population on the validation set after search had been stopped, keeping those that produced positive returns.

These two additional approaches did not increase the percentage of excess returns on the test set after training stopped, which the authors found surprising. They attributed this to a lack of correlation between the training and test performances. It is interesting that the authors examined the correlation between the training and test performances as an ex-post diagnostic tool. Below, we describe measuring the correlation between training and *validation* fitnesses, but we use it instead as a potential stimulus to prematurely halt our search. The authors concluded that using the validation methods they described was not providing adequate improvements to excess returns, as evaluated on the test set. They felt that additional criteria needed to be investigated in order to identify a cut-off point for search [22].

Gagné and co-authors [9] also investigate the use of three datasets (training data, validation data, and test data). They evolve solutions to binary classification problems using Genetic Programming. The inclusion of a validation set, to periodically check the evolved a model for a loss of generalisation, reduces the amount of data used to train the model. This reduction in the data reserved for training means that the training algorithm has less information with which to fit a model, and increases the possibility that the model produced will not be representative of the true underlying model. The trade-off between the inclusion of validation data on the one hand, and the reduction in training data on the other hand, is examined by the authors.

Simple solutions are sometimes postulated to both reduce the effect of bloat (an uncontrolled increase in the average size of individuals evolved by GP [13]), and produce solutions without overfitting. The direct link between parsimony on the one hand, and solutions that do not overfit the data on the other hand, has been disputed in [8], as noted in Section 8.1.1. Results that contradict this - for example those results described in [22] and summarised above, have meant that researchers have continued to pursue parsimonious solutions, albeit with an awareness of the challenge made against their effectiveness (for example, in [3]).

Gagné and co-authors [9] also investigate the use of lexicographic parsimony pressure to reduce the complexity of the evolved models. Lexicographic parsimony pressure [12] involves minimizing the error rate on the entire training set, and using the size as a second measure to compare when the error rates are exactly the same. The best individual of a run is the individual who exhibits the lowest error rate on the training set, and the smallest individual is selected in the case of a tie. They establish that while there is no clear advantage in terms of test set accuracy from using a validation set and parsimony pressure, the inclusion of a validation set and parsimony pressure does *lower the variance* of the test set error on the evolved model across 100 runs. They express a desire to develop new stopping criteria in future work, which would be based on the difference between training and validation fitness.

8.2.2 Early Stopping

Early stopping is a method used to counteract overfitting, whereby training is stopped when overfitting begins to take place. In order to enable early stopping, the data is split into three subsections: training data, validation data, and testing data. Initially, a model is fitted to the training data. At regular intervals as the model fitting proceeds, the fitness of the model on the validation data (which has not been used to train the model) is examined for disimprovement. The ability of the evolved model to generalise beyond the training examples is therefore measured while the run is in progress, the assumption being that if the learned model is generalising well, it should exhibit high fitness on both the training data, and the unseen validation data. When a disimprovement is observed in the fitness of the evolved model on the validation data (or is observed for a specified number of consecutive generations or iterations) training is stopped. The model with the lowest validation error prior to training having been stopped, is used as the output of the run [10].

A More Robust Way to Determine When to Stop Training

This simple early stopping technique has been criticised by Prechelt in [21], where he examines the use of early stopping when training a neural network. Prechelt states that the validation set error, in most cases, fluctuates throughout the course of the run. He describes how the validation set error rarely monotonically improves during the early stage of the run, before monotonically disimproving after overfitting has begun to take place. He states that real validation error curves almost always have more than one local minimum. The question then becomes, when should early stopping take place? He goes on to propose three classes of stopping criteria, with the aim of developing criteria which lead to both lower generalisation error, and exhibit a reasonable trade-off between training time and improved generalisation.

8.2.3 Grammatical Evolution: A Brief Introduction

Grammatical evolution (GE) [7, 17] is a popular form of grammar-based GP [14]. A particular strength of GE is the use of a grammar to incorporate domain knowledge about the problem we are attempting to solve. In GE, the process of evolution first involves the generation of a population of randomly-generated binary (or integer) strings, the genotype. In the case of binary genomes, each set of B bits (where traditionally B=8) is converted into its equivalent integer representation. These integer strings are then mapped to a phenotype, or high-level program or solution, using a grammar, which encompasses domain knowledge about the nature of the solution. Therefore, a GE genome effectively contains the instructions of how to build a sentence in the language specified by the input grammar. Grammatical Evolution has been applied to a broad range of problems, including many successful examples in financial modelling [5].

The grammar used in the first set of experiments we performed can be found in Fig. 8.1. The grammar is composed of non-terminal and terminal symbols. Terminals (for example, arithmetic operators) appear in the solution, whereas non-terminals can be further expanded into terminals and non-terminals. Here we can see the syntax of the solution (that will be constructed from arithmetic operators, mathematical functions, variables and constants) is encoded in the grammar.

The mapping process involves the use of an integer from the genotype to choose a production rule from the choices available to the non-terminal currently being mapped. This process proceeds as follows. The first integer from the integer representation of the genotype is divided by the number of rules in the start symbol (<expr> in our example). The remainder from this division is used to select a rule from the grammar:

$$rule = \frac{\text{(Codon integer value)}}{\text{MOD}} \\ \text{(Number of rules for the current non – terminal)}$$

Groups of production rules are indexed from zero, so if the result of division leaves a remainder of 0, the production rule with an index value of zero will be chosen. For example, given there are 5 choices of production rule available to map from <expr>, if the first integer in the integer-representation of the genotype was 8, then

$$8 \text{ MOD } 5 = 3$$

```

<prog> ::= <expr>
<expr> ::= <expr> <op> <expr> (0)
          | ( <expr> <op> <expr> ) (1)
          | <pre-op> ( <expr> ) (2)
          | <protected-op> (3)
          | <var> (4)
<op> ::= + (0)
        | * (1)
        | - (2)
<protected-op> ::= div( <expr>, <expr>)
<pre-op> ::= sin (0)
           | cos (1)
           | exp (2)
           | inv (3)
           | log (4)
<var> ::= X (0)
        | 1.0 (1)

```

Fig. 8.1 Grammar used in Symbolic Regressions

and so the third rule (indexing from zero), which is `<protected-op>`, would be selected. `<protected-op>` only contains one possible mapping - `div(<expr>, <expr>)`. This means there is no need to read an integer from the genotype to determine which rule will be chosen to map from `<protected-op>`, since there is no choice to be made. The next integer in the genotype would then need to be read in order to map between the leftmost `<expr>` and one of its constituent rules. Let's assume this next integer had value 39. Once again, there are five choices available to map from `<expr>`, so

$$39 \text{ MOD } 5 = 4$$

would select the fourth production rule for `<expr>`, which counting from 0, is `<var>`. The third integer in the genotype would next be used to map between `<var>` and one of its constituent rules. This process continues until either all integers in the genotype have been used up, or our mapping process has resulted in the production of a phenotype (that is a structure comprised only of terminal symbols) [17].

8.3 Some Experiments to Illustrate Overfitting in Symbolic Regression Problems

In order to clearly illustrate the problem of overfitting in model induction using GP, we show in this section what happens when three symbolic regression functions are fit using Grammatical Evolution [17] in a range that biases the output towards an overfit model. This exposition is based on work we carried out in [23].

8.3.1 Setup

Equations 1 through 3 show the target functions. The training dataset, validation data set, and test datasets were all comprised of 10 randomly generated points. The test dataset was not used to train the model, and was comprised of points solely outside the training range in order to specifically focus on the extrapolation capabilities of the evolved model.

Target Function 1:

$$Y = 0.6X^3 + 5X^2 - 10X - 25 \quad (8.1)$$

Training dataset range: [-5, 5].
 Validation data set range: [-5, 5].
 Test dataset ranges: [-10, -5] and [5, 10].

Target Function 2:

$$Y = 0.3X \times \sin 2X \quad (8.2)$$

Training dataset range: [-1, 1].
 Validation dataset range: [-1, 1].
 Test dataset ranges: [-2, -1] and [1, 2].

Target Function 3:

$$Y = \exp X - 2X \quad (8.3)$$

Training dataset range: [-2, 2].
 Validation dataset range: [-2, 2].
 Test dataset ranges: [-4, -2] and [2, 4].

These functions and ranges were chosen so that the target function would be trained using a biased data sample. The bias resulted from training in a range in which the target function closely resembled an alternative function. Over a wider range than that from which the training data was drawn, the target function looked quite different from this alternative (for example, function 2 is a Sine function as we can see from the target function given above. However, if examined only in the training range of between minus one and one, it resembled a quadratic function, (see Fig. 8.6)). In this way, we engineered a situation in which overfitting was likely to take place. In each case, Grammatical Evolution was run on a population size of 100 individuals, for 51 generations, using Grammatical Evolution in Java [16]. The grammar used is shown in Fig. 8.1.

Fitness was evaluated by computing the mean squared error of the training points when evaluated on each individual (therefore the lower the fitness value, the better the evolved function fitted the training data).

$$MSE = \frac{\sum_{i=1}^n |targetY - phenotypeY|^2}{n} \tag{8.4}$$

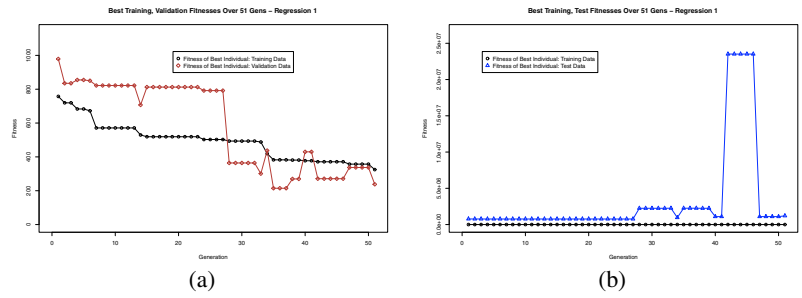


Fig. 8.2 Target Function 1

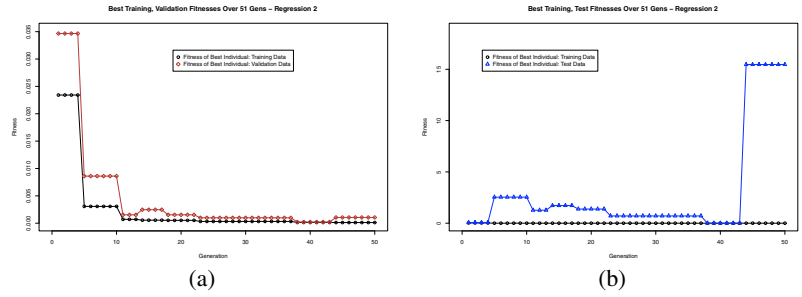


Fig. 8.3 Target Function 2, Example 1

8.3.2 Results

Figs. 8.2 through 8.5 are plots of the fitness of the best individual at each generation as evaluated on the training data, against the fitness of the best individual at each generation as evaluated on the validation and test datasets, for four illustrative runs - one run each of target functions 1 and 3, and two runs of target function 2. Table 8.1 shows results of interest with respect to the fitness as evaluated on the validation and test dataset, for 9 runs. It shows that stopping evolution before the specified number of generations had elapsed, in the majority of cases would have led to the model extrapolating better beyond the range in which it was trained [23].

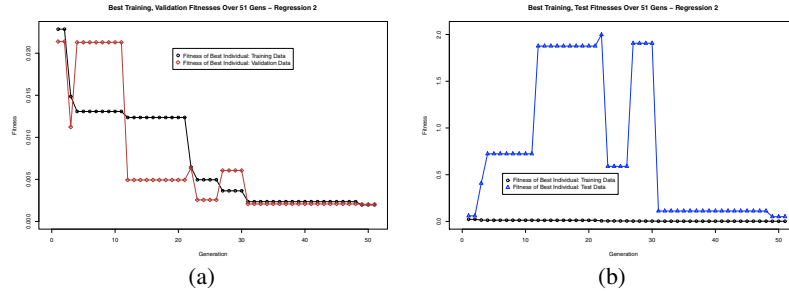


Fig. 8.4 Target Function 2, Example 2

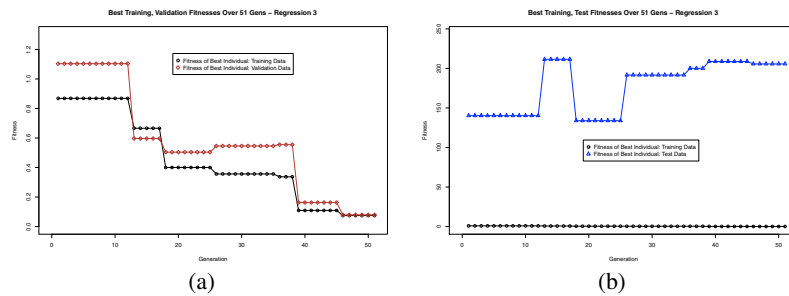


Fig. 8.5 Target Function 3

Table 8.1 Results of Interest: Validation, Test fitnesses

Target Function	Would Traditional Early Stopping Have Been Useful? - Validation Fit.	Would Traditional Early Stopping Have Been Useful? - Test Fitness	Generation of Best Result (Test Fitness) before or after the Gen. of Result of Training, as per Traditional Early Stopping?
1	No	Yes	After
1	No	Yes	Before or Same Time
1	Yes	Yes	After
2	No	Yes	After
2	No	No	After
2	No	Yes	After
3	No	Yes	Before or Same Time
3	No	Yes	After
3	Yes	Yes	Before or Same Time

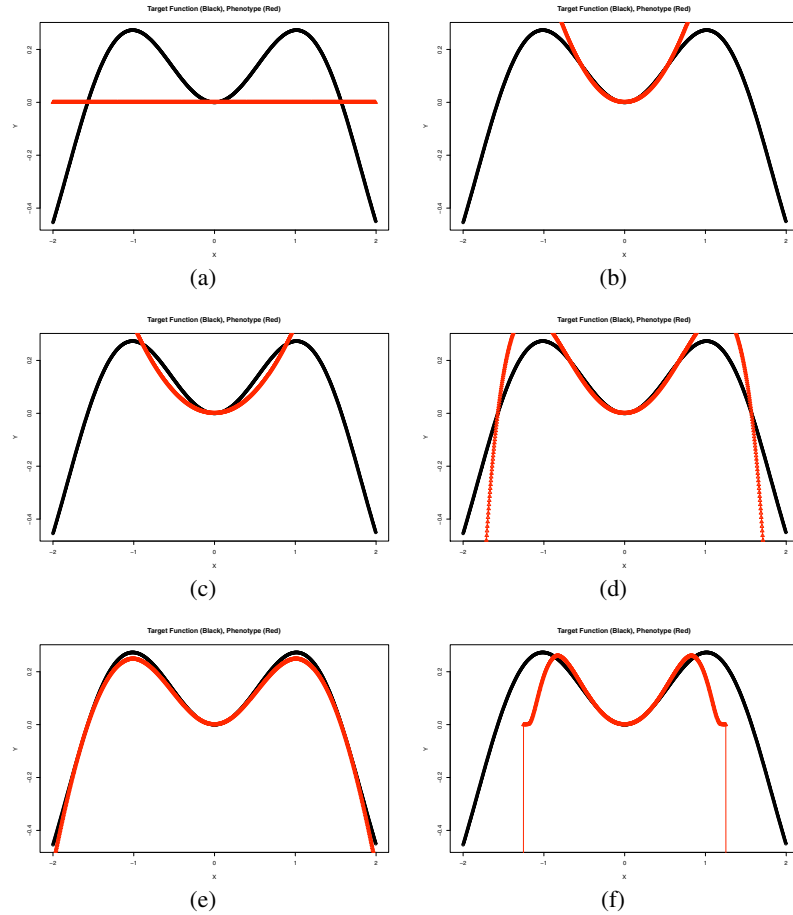


Fig. 8.6 (a) Generation 1 (b) Generation 5 (c) Generation 11 (d) Generation 23 (e) Generation 38 (f) Generation 44

Early stopping has been described in Section 8.2.2. The validation dataset is not used to train the model, but instead is used to test the fitness of the model every once in a while (for example each generation, or at five generation intervals). If the fitness of the best individual as evaluated on the validation dataset disimproves, this is taken as an indication that the evolved model is overfitting the data, and evolution is stopped. Test data is used as before to evaluate the fitness of the evolved model on out-of-sample data, after evolution has terminated, either prematurely (if early stopping has been deemed necessary), or after the specified number of generations has elapsed.

Since we explicitly chose target functions and ranges with an inherent bias, these symbolic regressions triggered overfitting, as expected. Using traditional early stopping, training is stopped the first time validation fitness disimproves. The result of training is taken to be the model produced at the generation immediately before training is stopped. In the second

and third columns of Table 8.1, we look at the validation and test fitnesses (respectively), and determine whether or not performing traditional early stopping would have produced a model with lower validation (column 2) and test set fitnesses (column 3), than the model produced at the end of the run. In eight of the nine runs described, the test fitness was better the generation immediately before the validation fitness first disimproved, than at the end of the run (column 3). Had we stopped evolution the first time the validation fitness disimproved, we would have produced a model that extrapolated better beyond the training range, than that produced at the end of the run (remember that the test set is comprised of points solely outside the training range). We see that in only two of the nine runs was the validation fitness better the generation before it first disimproved than at the end of the run. This is not that surprising. The validation data points are drawn from the same range as the training points. Therefore, overfitting is less likely to occur in the training/validation range than outside of this range.

Prechelt [21] shows that when training artificial neural networks, the first time the error on the validation set increases is not necessarily the best time to stop training, as the error on the validation set may increase and decrease after this first disimprovement. The last column of Table 8.1 shows for each run, if the best stopping point came before or after the generation of the result of training as dictated by traditional early stopping. The best stopping point here refers to the earliest generation of lowest *test* error. We examine the eight runs where the model that we would have evolved using traditional early stopping had better test fitness than the model that would have been evolved at the end of the run. In five out of these eight runs, the optimal generation at which to stop (as measured by test fitness) came later than the generation of the result of training using traditional early stopping [23].

In order to give further insight into the evolutionary process that underlie the changes in fitness observed for the training and test data sets, the phenotype was plotted against the target function in the entire data set range (that is, throughout the range of training, validation and test data), at each generation. Fig. 8.6 shows a selection of these generational graphs for the first run of function 2.

Comparing Figs. 8.6 and 8.3(b), we can clearly see the correspondences between changes in the graphed phenotype over the generations and changes in the fitness as evaluated on the test data. Between generations 1 and 22, the test fitness is either disimproving, or not improving by much. At generation 23 (Fig. 8.6(d)) fitness improves significantly, and at generation 38 (Fig. 8.6(e)), an extremely fit individual has been evolved, both with respect to the training and test set. In Fig. 8.3(b) we see that the error on the test data is extremely low at generation 38. The model extrapolates well. However, come generation 44 (Fig. 8.6(f)), a much less fit function has been evolved. It's fitness on the training data has improved, but it's fitness on the test data has drastically disimproved. In Fig. 8.3(b) we can see an explosion in the test error towards the end of the run [23], which contrasts with the low value of the training error.

8.4 Solutions to Prevent Overfitting - Stopping Criteria

In [21], Prechelt implements three classes of stopping criteria, which determine when to stop training in order to preserve the generalisability of the evolved model. Each class of criteria assumes that we are evaluating the fitness of a model by examining the error of the model on the training and validation datasets. Validation set error is used to measure how well the model is generalising beyond the training examples. *Generalisation loss* is measured by dividing the average validation set error per example measured after the current epoch (E_{val}), by the minimum validation set error observed up until the current epoch (E_{opt}). In our implementation of these criteria, we instead use the validation set error of the best individual

at the current generation, for E_{va} . In percentage terms, the generalisation loss (GL) at epoch t is given by:

$$GL(t) = 100 \times \left(\frac{E_{va}}{E_{opt}} - 1 \right) \quad (8.5)$$

First Class of Stopping Criteria - Stop when the Generalisation Loss Exceeds a Threshold

The first class of stopping criteria measures the loss of generalisation of the trained model at each epoch, and stops training if the generalisation loss is observed to have crossed a predefined threshold (denoted here by α). In [21] the class GL_α is defined as follows:

$$GL_\alpha : \text{stop after first epoch } t \text{ with } GL(t) > \alpha \quad (8.6)$$

While stopping once the generalisation loss passes a certain threshold appears to be a good rule of thumb, it may be useful to add a caveat to that. Before applying early training stopping, we may wish to check whether or not training is still progressing rapidly. The *training progress* is examined over k generations. It measures how much the average training error during the strip of length k , was larger than the minimum training error during the strip. It is given (in per thousand) by:

$$P_k(t) = 1000 \times \left(\frac{\sum_{t'=t-k+1}^t E_{tr}(t')}{k \times \min_{t'=t-k+1}^t E_{tr}(t')} - 1 \right) \quad (8.7)$$

where $E_{tr}(t)$ is the average training error per example measured after time t . In our experiments using these criteria, $E_{tr}(t)$ measured the training error of the best individual at the current generation.

Second Class of Stopping Criteria - Quotient of Generalisation Loss and Progress

If training is progressing well, then it may be more sensible to wait until both generalisation loss is high and training progress is low, before stopping. The intuition behind this lies in the assumption that while training is still rapidly progressing, generalisation losses have a higher chance to be 'repaired'.

A second class of stopping criteria was defined in [21] to use the quotient of generalisation loss and progress:

$$PQ_\alpha : \text{stop at first end-of-strip epoch } t \text{ with } \frac{GL(t)}{P_k(t)} > \alpha \quad (8.8)$$

Third Class of Stopping Criteria - Stop Training when the Generalisation Error Increases in s Successive Strips

The third and final class of stopping criteria identified in [21] approached the problem from a different angle than the first two. It recorded the sign of the changes in the generalisation error, and stopped when the generalisation error had increased in a predefined number of successive strips. The magnitude of the changes was not important, only a persistent trend in the direction of those changes. The third class of stopping criteria is defined as:

$$\begin{aligned}
& \text{UP}_s : \text{stop at epoch } t \text{ iff } \text{UP}_{s-1} \text{ stopped at epoch } t - k \\
& \text{and } E_{va} > E_{va}(t - k) \\
& \text{UP}_1 : \text{stop at first end - of - strip epoch } t \\
& \text{with } E_{va}(t) > E_{va}(t - k)
\end{aligned} \tag{8.9}$$

where s is the number of successive strips.

Stopping criteria decide to stop at some time t during training, and the result of the training is then the set of weights (in the case of neural networks), or the evolved model (in the case of GE) that exhibited the lowest validation error prior to the time at which training was stopped. The criteria can also be used if we are trying to maximise a fitness metric, rather than minimise an error function. In this case the formulae used need to be adjusted to reflect the fact that we are maximising fitness, not minimising error.

8.5 Investigations: Stopping Criteria applied to a Financial Dataset

In Section 8.2 we outlined the contribution of various authors investigating overfitting and generalisation in GP and neural networks. Some of these [9, 21, 22] point to the need for additional criteria to determine the point at which training is stopped to avoid overfitting. Thus inspired, we present an implementation of the stopping criteria detailed in Section 8.4 using an alternative form of Grammar-based GP to GE.

These criteria have not, to the best of our knowledge, been applied in a GP setting before now. This presents a valuable opportunity to apply approaches implemented in the neural networks literature, to the field GP. This work is related to work we carried out using early stopping criteria with symbolic regression in [24]. We also implement an additional stopping criteria not found in the work carried out in [21].

Here, technical trading rules are evolved in the form of decision-trees (DTs) for the trading rule induction problem, in order to generate signals to take a short or long position. We emphasise that our focus is not on the construction of high-quality trading rules, instead we are focussing on demonstrating the practical application and potential usefulness of stopping criteria used to enhance the generalisation of solutions evolved using GP.

8.5.1 Grammar-based GP Experimental Setup

The grammar in Fig. 8.8 presents the grammar adopted for program representation. Each expression-tree is a collection of `if-then-else` rules that are represented as a disjunction of conjunctions of constraints on the values of technical indicators.

Technical Analysis (TA) has been widely applied to analyse financial data and inform trading decisions. It attempts to identify regularities in the time-series of price and volume information, by extracting patterns from noisy data [5]. The technical indicators we used for these experiments are: (a) **simple moving average** (MA), (b) **trade break out** (TBO), (c) **filter** (FIL), (d) **volatility** (VOL), and **momentum** (MOM). For a good introduction to these, please refer to [5]. TA indicators are parameterised with lag periods, which is the number of past time-steps that each operator is looking at. Currently, we allow periods from 5 to 200 closing days, with a step of 5 days. We also include the closing price of the asset at each time-step.

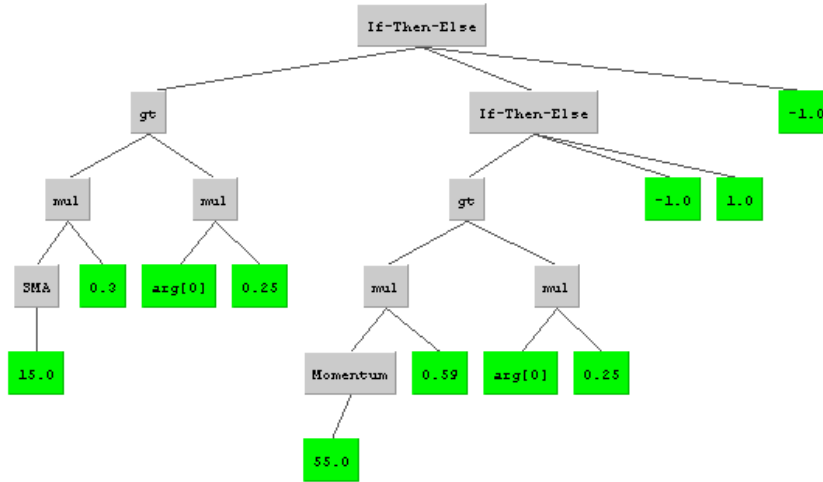


Fig. 8.7 A Sample Decision Tree

A sample decision tree is given in Fig. 8.7. Here, $\text{arg}[0]$ represents the closing price. This rule tells us to examine if three-tenths of the Simple Moving Average over the past fifteen time-steps is greater than one quarter of the closing price (in essence, a form of momentum indicator). If it is, we next execute the middle branch of the tree and act accordingly, otherwise we take a short position. In order to execute the middle branch of the tree, we evaluate whether or not the momentum over the last 55 time-steps, multiplied by 0.59, is greater than one quarter of the closing price. If it is, we take a short position - otherwise we take a long position.

The GP algorithm employs a panmictic, generational, elitist Genetic Algorithm. The algorithm uses tournament selection. The population size is set to 500, and evolution continues for 50 generations. Ramped-half-and-half tree creation with a maximum depth of 5 is used to perform a random sampling of DTs during run initialisation. Throughout evolution, expression-trees are allowed to grow up to a depth of 10. The evolutionary search employs a variation scheme that combines mutation with standard subtree crossover. A probability governs their application, set to 0.7 in favour of mutation. No cloning was used. The maximisation problem employs a fitness function that takes the form of average daily return (ADR) generated by the rule's trading signals over a training period.

8.5.2 Financial Time-Series and Trading Methodology

The datasets used are daily prices for a number of financial assets. These are the foreign exchange rate of EUR/USD, the Nikkei 255 index, and S&P 500 index, for the period of 01/01/1990 to 31/03/2010. The first 2,500 trading days are used for training, and the remaining 2,729 are equally divided between validation and test sets, both with size of 1,364 days.

Each evolved rule outputs two values, 1 and -1, interpreted as a long and short position respectively. The average return of a rule (Average Daily Return (ADR)) is generated as follows. Let r_t be the daily return of the index at time t , calculated using $(v_t - v_{t-1})/v_{t-1}$, where v_t and v_{t-1} are the values of the time-series at time t and $t - 1$ respectively. Also, let

```

<prog> ::= <if>
<if> ::= <predicate> <expr> <expr>
<expr> ::= <if>
        | <signal>
<signal> ::= -1
           | 1
<predicate> ::= <tiexpr> <comp> <arithexpr>
<tiexpr> ::= <arithexpr> <arithop> <tiexpr>
           | <tiexpr> <arithop> <tiexpr>
           | <coeff> * <ti>
           | <coeff> * <ti> + <coeff> * <ti>
           | <ti>
<arithexpr> ::= <val> <arithop> <val>
              | <val>
<valA> ::= <coeff>
          | closingprice
<coeff> ::= <const> <arithop> <const>
          | <const>
<comp> ::= <
          | >
<arithop> ::= +
            | -
            | *
            | /

<ti> ::= MA ( 5 )
        | ...
        | MA ( 200 )
        | TBO ( 5 )
        | ...
        | TBO ( 200 )
        | FIL ( 5 )
        | ...
        | FIL ( 200 )
        | MOM ( 5 )
        | ...
        | MOM ( 200 )
        | VOL ( 5 )
        | ...
        | VOL ( 200 )
<const> ::= random constant in [-1.0, 1.0]

```

Fig. 8.8 Grammar used on Financial Datasets

s_{t-1} be the trading signal generated by the rule at time $t - 1$. Then $d_t = s_{t-1}r_t$ is the realised return at time t . Using a back-test period, an average of d_t can be induced; to annualise this we simply multiply it by 200 trading days. Trading, slippage and interest costs are not considered.

8.5.3 Results and Discussion

The thresholds used for the GL, PQ and UP criteria were closely mirrored on those used in [21], with a few minor changes. The strip length used was also taken from [21], and was set to 5, for both the UP and PQ classes of criteria.

As noted in Section 8.5.1, the evolution of trading rules is governed by a fitness function that maximises the average daily return (ADR) generated over a period. One point to note here is that the formulae have been re-written in terms of the fitness being maximised, rather than the equivalent form of the error being minimised. It is worth pointing out, in order to avoid confusion, that instead of stopping when the generalisation error increased for some number of successive strips, when dealing with fitness maximisation, the ‘UP’ criteria stopped when the generalisation error *decreased* for some number of successive strips.

Tables 8.2, 8.3 and 8.4 examine the performance of the GL, PQ and UP stopping criteria on the datasets from the EUR/USD, Nikkei 255, and S&P 500 indices. In addition, a new class of criteria were added, which depended on the correlation between the training and validation fitnesses. To the best of our knowledge, these criteria have not been implemented in the same way before. They caused training to be stopped if the Pearson’s correlation coefficient between the training set fitness and the validation set fitness, fell below a predefined threshold. The correlation criteria were evaluated at every generation during the run, until a stopping

Table 8.2 Criteria Performance on Euro/USD: Averages and (Standard Deviations) Over 50 Runs

Criterion	Slowness	Generalisability	Prob Best Crit	Prob Halting
GL_1	3.37 (2.7)	1.13 (0.78)	0	0.3
GL_2	3.39 (2.69)	1.13 (0.79)	0	0.28
GL_3	3.41 (2.68)	1.14 (0.78)	0	0.26
GL_5	3.41 (2.68)	1.14 (0.78)	0	0.26
$PQ_{0.5}$	4.08 (4.49)	1.02 (1.76)	0	0.86
$PQ_{0.75}$	4.28 (4.56)	1.03 (1.81)	0	0.86
PQ_1	4.45 (4.61)	1.05 (1.84)	0	0.82
PQ_2	4.8 (4.77)	1.01 (1.77)	0	0.8
PQ_3	5.02 (4.85)	0.94 (0.54)	0	0.8
UP_2	3.29 (4.22)	1.39 (3.6)	0.02	0.56
UP_3	3 (3.56)	1.19 (0.41)	0	0.24
UP_4	3.67 (3.51)	1.34 (0.58)	0	0.06
$COR_{0.8}$	3.62 (4.53)	0.76 (3.62)	0.4	1
$COR_{0.6}$	3.98 (4.66)	0.83 (3.18)	0.18	1
$COR_{0.4}$	4.31 (4.75)	0.88 (2.84)	0.2	0.98
$COR_{0.2}$	4.73 (4.56)	0.97 (2.28)	0.06	0.94
COR_0	5.03 (4.77)	0.93 (1.16)	0.06	0.9
$COR_{-0.2}$	5.73 (5.16)	0.94 (0.94)	0.02	0.76
$COR_{-0.4}$	5.43 (4.62)	0.95 (1.06)	0.02	0.66
$COR_{-0.8}$	5.25 (5.55)	0.58 (2.44)	0.04	0.32

generation was identified. The thresholds used for the correlation criteria were 0.8, 0.6, 0.4, 0.2, 0, -0.2, -0.4 and -0.8.

We experimented with a wide range of correlation values, from values describing strong positive correlation to strong negative correlation. It is important to keep in mind that these values represented thresholds, *which if crossed, would trigger an end to training*. As such, examining whether or not weaker (negative) as well as stronger (positive) thresholds would prove a useful stimulus to stop training merited investigation. As can be seen in Tables 8.2 to 8.4, this exploration proved worthwhile.

For each criteria, any run for which the criteria's threshold is not breached, is discarded from the analysis of the performance of the criteria. In examining the usefulness of the stopping criteria when applied to these financial data sets, we are only concerned with the performance of the criteria *when a stopping generation is identified*. We therefore don't know if this would have been the optimal stopping point in the context of the run completing its 50 generations. The final column in each of Tables 8.2, 8.3 and 8.4, entitled 'Prob Halting', displays the proportion of all 50 runs in which a stopping generation was identified by the criterion in question (the values in the 'Prob Halting' column are scaled between 0 and 1). To calculate the number of runs each criterion acted upon, multiply the probability of halting by 50.

Table 8.3 Criteria Performance on Nikkei 255: Averages and (Standard Deviations) Over 50 Runs

Criterion	Slowness	Generalisability	Prob Best Crit	Prob Halting
GL_1	0.88 (0.98)	0.77 (0.6)	0	0.12
GL_2	0.93 (0.98)	0.77 (0.6)	0	0.12
GL_3	0.97 (0.99)	0.76 (0.6)	0	0.12
GL_5	0.99 (0.99)	0.76 (0.6)	0.02	0.12
$PQ_{0.5}$	8.41 (11.03)	0.95 (0.43)	0	0.98
$PQ_{0.75}$	8.59 (11.11)	0.94 (0.43)	0	0.98
PQ_1	8.89 (11.27)	0.96 (0.39)	0.02	0.98
PQ_2	9.71 (11.74)	0.95 (0.39)	0	0.98
PQ_3	10.57 (12.17)	0.95 (0.37)	0	0.98
UP_2	7.56 (10.75)	0.96 (0.25)	0.02	0.86
UP_3	5.44 (9.71)	0.96 (0.16)	0	0.32
UP_4	4.36 (5.26)	0.99 (0.02)	0	0.18
$COR_{0.8}$	7.42 (10.44)	0.91 (0.53)	0.32	1
$COR_{0.6}$	8.05 (10.72)	0.93 (0.47)	0.12	1
$COR_{0.4}$	8.86 (11.08)	0.95 (0.39)	0.08	0.98
$COR_{0.2}$	9.84 (11.56)	0.98 (0.29)	0.04	0.92
COR_0	10.69 (12.07)	0.99 (0.2)	0.04	0.86
$COR_{-0.2}$	10.07 (10.86)	0.98 (0.2)	0.02	0.78
$COR_{-0.4}$	9.69 (10.13)	0.98 (0.23)	0.08	0.72
$COR_{-0.8}$	9.14 (10.43)	0.99 (0.36)	0.24	0.54

Analysing Performance

Some notation, which will be used below, is defined here. The time at which training was stopped (the stopping generation) is denoted by t_s . The test fitness at the generation of the result which is produced by applying criterion C , is $E_{te}(C)$.

Prechelt [21] defines a criterion as being *good*, if it is among those that find the lowest validation set error for the entire run. The *best* criterion of each run, is defined as a good criterion which has an earlier stopping generation than other good criteria. That is, among all the criteria that find the best validation fitness, the *best* criterion of all is that which stopped training quickest [21].

We analysed the performance of the stopping criteria on each of the three benchmark financial data sets[1]. Three performance metrics were defined. The ‘**slowness**’ of a criterion C in a run, relative to the best criterion x , is defined as

$$S_x(C) = t_s(C)/t_s(x).$$

The ‘**generalisability**’ of a criterion C in a run, relative to the best criterion x , is defined [21] as

$$B_x(C) = E_{te}(C)/E_{te}(x).$$

Table 8.4 Criteria Performance on S&P500: Averages and (Standard Deviations) Over 50 Runs

Criterion	Slowness	Generalisability	Prob Best Crit	Prob Halting
GL_1	6.5 (8.19)	0.74 (0.33)	0	0.08
GL_2	6.5 (8.19)	0.74 (0.33)	0	0.08
GL_3	6.5 (8.19)	0.74 (0.33)	0	0.08
GL_5	6.5 (8.19)	0.74 (0.33)	0	0.08
$PQ_{0.5}$	7.79 (11.13)	0.79 (1.12)	0.02	1
$PQ_{0.75}$	8.15 (11.35)	0.76 (0.97)	0.02	1
PQ_1	8.52 (11.59)	0.76 (0.95)	0.02	1
PQ_2	9.45 (12.26)	0.74 (0.9)	0.02	0.98
PQ_3	10.13 (12.75)	0.74 (0.86)	0	0.98
UP_2	7.07 (10.21)	0.64 (0.8)	0	0.74
UP_3	9.24 (11.47)	0.59 (0.9)	0	0.44
UP_4	7 (5.95)	0.48 (1.02)	0	0.12
$COR_{0.8}$	7.21 (10.78)	0.76 (1.28)	0.18	1
$COR_{0.6}$	8.04 (11.24)	0.8 (1.25)	0.26	0.98
$COR_{0.4}$	8.76 (11.67)	0.79 (1.3)	0.04	0.96
$COR_{0.2}$	9.56 (12.07)	0.8 (1.3)	0.1	0.9
COR_0	10.58 (12.61)	0.92 (1.22)	0.04	0.88
$COR_{-0.2}$	10.76 (12.7)	0.96 (1.28)	0.08	0.88
$COR_{-0.4}$	10.98 (12.97)	0.97 (1.49)	0.02	0.82
$COR_{-0.8}$	2.24 (3.62)	0.68 (2.26)	0.2	0.64

The ‘generalisability’ of a criterion therefore measures how well the result it produces generalises beyond the training and validation sets, relative to the best criterion. Tables 8.2, 8.3 and 8.4 show the average values for slowness and generalisability over all the runs in which halting took place, for each criterion. Standard deviations are shown in brackets.

The third and final metric measures the **probability that a particular criterion will be the best criterion for a run**. This is calculated as follows. The number of times a particular criterion was chosen as the best criterion of a run is counted. This number is divided by the total number of runs, to give the average number of times the criterion is chosen as the best criterion. This average is used to measure the probability that the criterion will be the best criterion.

The correlation criteria come out very well in terms of having a high probability of being judged the best criterion of a run in Tables 8.2, 8.3 and 8.4. This is a very interesting and potentially useful result. For the S&P 500 dataset, the correlation criteria have better generalisation performance than all other criteria (averaging performance values across all criteria in a particular class). Their generalisation performance is marginally worse than the UP criteria for the Nikkei 255 dataset, but they outperform the other classes of criteria. They perform worse than all classes of criteria for the Euro/USD dataset, in terms of generalisation. The GL criteria are the fastest criteria at identifying a stopping point in two out of the three datasets, and second best in the Euro/USD dataset. The correlation criteria are always slower than both the GL and UP criteria, across all three datasets (once again, taking the average performance across all criteria for each class). The correlation criteria may still be a superior choice than the other three classes of criteria, unless training time is a constraint.

8.6 Conclusions

In this chapter, we focused on the techniques to counteract overfitting in evolutionary driven financial model induction, first providing some background to the problem. In order to clearly illustrate overfitting, we showed the existence of overfitting in evolved solutions to symbolic regression problems using grammar-based GP, using diagrams as visual aids. After noting other authors observations that traditional early stopping techniques, which usually stop training when validation fitness first disimproves, were often insufficient in increasing generalisation [21, 22], we applied four early stopping criteria to an exemplar trading rule induction problem using grammar-based GP. Three of these criteria had previously been implemented in the neural network literature [21]. The fourth class of criteria, which measured the correlation between the training and validation fitness at each generation and stopped training once the correlation dropped below a predefined threshold (such as 0.6), proved very successful. This class of criteria had a high probability of being a best criterion of a run, meaning that it would stop sooner than any other criteria after the global maximum validation set fitness had been observed. It also usually had adequate generalisation performance with respect to independent test data, that had neither been used for training, nor to determine stopping points.

As for the other classes of criteria, the UP criteria stopped after a persistent disimprovement had been observed in the validation fitness over consecutive time periods. UP criteria generalise well - they display the best results for generalisation on the Euro/USD and Nikkei 225 dataset. They are by no means slow to stop either - being fastest at identifying a stopping point on the Euro/USD dataset, and second fastest on both the Nikkei 225 and S & P 500 datasets. If a low training time is important, then the GL criteria are the best option - however, the UP criteria are a very attractive alternative, given they also display good generalisation performance.

Several areas are indicated for future work. In early stopping, the evolutionary process could be extended to include parameters such as the appropriate stopping threshold, or indeed to the evolution of high-quality stopping rules from low-level stopping criteria primitives (a meta-heuristic approach). We also intend to extend our work to embrace other generalisation methodologies from the statistical literature.

Acknowledgements. All of the authors acknowledge the financial support of Science Foundation Ireland (Grant No. 08/SRC/FMC1389) under which this research was conducted.

References

1. Agapitos, A., O'Neill, M., Brabazon, A.: Evolutionary Learning of Technical Trading Rules without Data-Mining Bias. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6238, pp. 294–303. Springer, Heidelberg (2010)
2. Banzhaf, W., Nordin, P., Keller, R., Francone, F.: Genetic programming: An introduction. Morgan Kaufmann, San Francisco (1999)
3. Becker, L.A., Seshadri, M.: Comprehensibility and overfitting avoidance in genetic programming for technical trading rules. Worcester Polytechnic Institute, Computer Science Technical Report (2003)
4. Brabazon, A., Dang, J., Dempsey, I., O'Neill, M., Edelman, D.: Natural computing in finance: a review. In: Handbook of Natural Computing: Theory, Springer, Heidelberg (2010)
5. Brabazon, A., O'Neill, M.: Biologically inspired algorithms for financial modelling. Springer, Heidelberg (2006)
6. Costelloe, D., Ryan, C.: On improving generalisation in genetic programming. In: Vanneschi, L., Gustafson, S., Moraglio, A., De Falco, I., Ebner, M. (eds.) EuroGP 2009. LNCS, vol. 5481, pp. 61–72. Springer, Heidelberg (2009)
7. Dempsey, I., O'Neill, M., Brabazon, A.: Foundations in Grammatical Evolution for Dynamic Environments. Springer, Heidelberg (2009)
8. Domingos, P.: The role of Occam's razor in knowledge discovery. *Data Mining and Knowledge Discovery* 3(4), 409–425 (1999)
9. Gagné, C., Schoenauer, M., Parizeau, M., Tomassini, M.: Genetic programming, validation sets, and parsimony pressure. In: Collet, P., Tomassini, M., Ebner, M., Gustafson, S., Ekárt, A. (eds.) EuroGP 2006. LNCS, vol. 3905, pp. 109–120. Springer, Heidelberg (2006)
10. Gencay, R., Qi, M.: Pricing and hedging derivative securities with neural networks: Bayesian regularization, early stopping, and bagging. *IEEE Transactions on Neural Networks* 12(4), 726–734 (2002)
11. Kushchu, I.: Genetic programming and evolutionary generalization. *IEEE Transactions on Evolutionary Computation* 6(5), 431–442 (2002)
12. Luke, S., Panait, L.: Lexicographic parsimony pressure. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 829–836. Morgan Kaufmann, San Francisco (2002)
13. Luke, S., Panait, L.: A comparison of bloat control methods for genetic programming. *IEEE Transactions on Evolutionary Computation* 14(3), 309–344 (2006)
14. McKay, R.I., Hoai, N.X., Whigham, P.A., Shan, Y., O'Neill, M.: Grammar-based Genetic Programming: a survey. *Genetic Programming and Evolvable Machines* 11(3–4), 365–396 (2010)

15. Mitchell, T.: *Machine Learning*. McGraw-Hill, New York (1997)
16. O'Neill, M., Hemberg, E., Gilligan, C., Bartley, E., McDermott, J., Brabazon, A.: GEVA: grammatical evolution in Java. *ACM SIGEVOLUTION* 3(2), 17–22 (2008)
17. O'Neill, M., Ryan, C.: *Grammatical Evolution: Evolutionary automatic programming in an arbitrary language*. Kluwer, Dordrecht (2003)
18. O'Neill, M., Vanneschi, L., Gustafson, S., Banzhaf, W.: Open issues in genetic programming. *Genetic Programming and Evolvable Machines* 11(3-4), 339–363 (2010)
19. Paris, G., Robilliard, D., Fonlupt, C.: Exploring overfitting in genetic programming. In: Liardet, P., Collet, P., Fonlupt, C., Lutton, E., Schoenauer, M. (eds.) *EA 2003*. LNCS, vol. 2936, pp. 267–277. Springer, Heidelberg (2004)
20. Poli, R., Langdon, W.B., McPhee, N.F.: *A field guide to genetic programming*. Lulu Enterprises, UK (2008)
21. Prechelt, L.: Early stopping - but when? In: Orr, G.B., Müller, K.-R. (eds.) *NIPS-WS 1996*. LNCS, vol. 1524, p. 55. Springer, Heidelberg (1998)
22. Thomas, J.D., Sycara, K.: The importance of simplicity and validation in genetic programming for data mining in financial data. In: *Proceedings of the Joint GECCO 1999 and AAAI 1999 Workshop on Data Mining with Evolutionary Algorithms: Research Directions*, pp. 7–11 (1999)
23. Tuite, C., Agapitos, A., O'Neill, M., Brabazon, A.: A Preliminary Investigation of Overfitting in Evolutionary Driven Model Induction: Implications for Financial Modelling. In: Di Chio, C., Brabazon, A., Di Caro, G.A., Drechsler, R., Farooq, M., Grahl, J., Greenfield, G., Prins, C., Romero, J., Squillero, G., Tarantino, E., Tettamanzi, A.G.B., Urquhart, N., Uyar, A.Ş. (eds.) *EvoApplications 2011, Part II*. LNCS, vol. 6625, pp. 120–130. Springer, Heidelberg (2011)
24. Tuite, C., Agapitos, A., O'Neill, M., Brabazon, A.: Early Stopping Criteria to Counteract Overfitting in Genetic Programming. In: *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*. ACM, New York (2011) (forthcoming)
25. Vanneschi, L., Gustafson, S.: Using crossover based similarity measure to improve genetic programming generalization ability. In: *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, pp. 1139–1146. ACM, New York (2009)