Abstract-
This paper describes the initial results of a new form of evolutionary system specifically designed for time series modeling. The system combines a grammatically-based Genetic Programming system with various optimisation techniques. The system uses the evolutionary system to construct the structure of equations and optimisation techniques to essentially fill in the details. Three forms of optimisation are described: optimisation of constants in an equation; the optimisation of both the constants and variables in an equation; and the use of a hill-climbing mutation to further tune the evolved and optimised equations. Preliminary results indicate that this combination of techniques produces significant improvements in convergence based on the training data, and produces equivalent generalisation on unseen data, for a given number of population member evaluations.

1 Introduction

Evolutionary systems, such as genetic programming(GP)[1], have been successfully applied to univariate and multivariate time-series problems from a variety of problem domains (see for example [1, 2, 3, 4]). However, constructing an equation generally involves the use of constants to scale the model and to adjust the various contributions of the independent variables. Although the evolutionary process is good at finding the overall form of a solution it is often difficult to evolve appropriate constants as part of the solution.

Previous work to supplement the tuning of constants in an evolved equation have usually augmented the evolutionary process with a hill-climbing mutation [3] or other simple forms of adjustment. Although there are a number of standard mathematical optimisation techniques, such as Simplex and Powell optimisation [5], these approaches have not been integrated into the standard evolutionary approach to equation discovery.

This generation process will produce nodes that are a mix of terminals and nonterminals, except the root will always be a nonterminal, and the leaves will always be terminals. Non-

2 Grammatical GP Optimisation

Before describing how variables and constants are represented in the system, it is necessary to describe how the time-series expressions are constructed.

2.1 Structure of Expressions

The expressions used in this system are constructed only of operators and various kinds of numbers. However, as will be described, some of these numbers are interpreted as variables in the expressions. In effect, TSOGP performs a heuristic search for an optimal expression that represents a univariate or multivariate equation. However, as the search space of all possible expressions is unacceptably large, the system is restricted to only those expressions that follow a grammar. This introduces an explicit bias in our search, which can be constructed by the user using domain knowledge, based on the form of expression most likely to produce a good model.

The system described here uses a modified version of a context-free grammar, by altering the right-hand side of the rules or productions. For the type of grammar used here, \( G \) is a tree, where the nodes have variable arity. A node in the tree can be a terminal or a nonterminal. For a terminal node, the arity depends on the actual terminal: A node representing `+` will have arity 2, a node representing a constant will have arity zero, and thus be a leaf node in the tree. Creating programs based on a context-free grammar has been previously described in [6, 7]. Note that a maximum depth of derivation tree must be given to halt the generation process within a certain number of rule substitutions.

This generation process will produce nodes that are a mix of terminals and nonterminals, except the root will always be a nonterminal, and the leaves will always be terminals. Non-
terminals in the tree are kept to facilitate implementation of the mutation and crossover operators. These operators will only ever produce expressions that follow the grammar being used, by altering the trees directly below a nonterminal node, following the definition of the grammar. The approach has been demonstrated previously [8] as a method for maintaining a constrained language definition during crossover and mutation.

Shortcuts are inserted in the tree being generated to speed up the evaluation process. These shortcuts allow the evaluator to skip any nonterminal nodes internal to the tree, jumping directly from terminal to terminal. Depending on the grammar being used, there may be many consecutive internal nonterminal nodes, and, since the evaluation process will be performed many times, it is of the utmost importance that it runs as fast as possible. An example of a simple tree constructed from a grammar is shown in Figure 1. Note that the grammar has defined the expression $\text{var} + \text{const}$, however the values for the variable and constant are not yet specified. Essentially the grammar has generated a structure to hold variables and constants, that will later be mapped to values for execution using optimisation or random techniques.

2.2 Description of Approach

As hinted at above, the shape of the optimal tree is found using an evolutionary approach. A population of such trees is generated, the score of each population member is determined, then some hillclimbing mutations may be applied to a number of the best scoring members, and finally a new population may be generated through some combination of straight copying, mutation and crossover. Note that the new generation is created from the previous one, and the previous generation is discarded. Then the process goes back to scoring the population members. This basic structure is shown in Figure 2. Note that the scoring of population members has two distinct meanings: when no optimisation occurs this is just the error measurement based on the training data and selected error function; when optimisation is allowed this means a search, limited by a set number of evaluations, that attempts to discover the minimum error by altering the numbers representing both real values (i.e. numbers in the equation) and the variables. These details will be explained in more detail in §2.2.1.

Both the hillclimbing, (adaptive training), and generating the new population, (evolution), are optional. In fact various strategies (one of which is 'None') can be chosen for both these processes, all potentially with a number of options. Of course, no training combined with no evolution results in a static system.

Each member of the population is evaluated once for each of the data points in the training data, and the results are compared to the measured values in the training period. This comparison is done by an error function. Currently the error functions available include the standard root mean square error (RMSE), coefficient of efficiency (COE) and a combined RMSE and fourier transform measure. These error functions represent the surface that the optimisation techniques will use to explore the best values for each of the constants and variables in the equation. The main criterion for these functions is that the error surface is continuous, since this is a basic requirement for the optimisation techniques.

2.2.1 How Optimisation is used

The main difference between TSOGP and previous systems is the way constants and variables are expressed in the tree, and how optimisation can be applied to this representation. Referring back to Figure 1, there are two kinds of nodes in the expression tree that represent numbers. The first kind is the node called 'number'. This node contains a number which is the value of the node, which will not be optimised. A 'number' can be set for either a value in an equation (such as $\text{var} + 2.35$) or to represent a particular independent variable. The second kind is the node called 'constant'. This node also contains a number, but this number is used as an index to an external array of numbers to get the value of the node, which represents an independent variable or real value in an equation.

This structure allows the use of a multi-dimensional optimisation routine to find the specific values for these 'constant' nodes for which the value of the error function for this expression is minimal. Again, various different optimization routines may be chosen. The standard Simplex and Powell Optimisation [5] techniques have been implemented. These optimisation techniques were selected since they are not con-
strained by conditions like monotonicity, convexity or differentiability of the function being optimised. Both are capable of optimising a function with a single output value, based on an arbitrary number of input variables. The main drawback of these techniques is that they generally require many evaluations to find the global minima (maxima).

One of the available optimizers is called the ‘evolutionary’ optimizer. When this optimizer is selected, all ‘constant’ nodes in the expression tree (which would normally be optimised) are replaced with ‘number’ nodes containing a random floating point number. This happens when such a tree is constructed. This basically means that the search for the best expression is fully evolutionary, since these values (be they variables or numbers) are fixed.

### 2.2.2 Mapping and Optimisation of Variables

One requirement that the selected optimizers have is that the function being optimised is continuous. A second requirement is that the range of the parameters being optimized should be unlimited. This has consequences for several of the built-in functions (terminals).

![Figure 3: Mapping Variables to a Continuous Domain](image)

The `var` function takes one argument. (In other words, in the expression tree it has one direct child.) It retrieves the value of one of the independent variables for the current data point. Which independent variable is selected depends on the value of the argument. Since this function will have to be continuous, the mapping between the argument and the result is as follows.

Integer values of the argument correspond directly to values of the independent variables, in such a way that any possible pair of variables corresponds to some two sequential integers. Since, for a given number of independent variables, there is only a limited number of pairs, only a limited number of sequential integers is needed. For arguments outside that range, the selected variables wrap around. For values of the argument that are non-integer, instead lying between two integers, the result is smoothly interpolated between the results for those two integers. This mapping of variables to a continuous space is shown in Figure 3. This simple example shows how 3 variables are mapped into a continuous linear space. The contribution of each variable is determined by the placement on this line. For example, if the value is exactly 2, then the variable represented is ‘variable 2’. If the value is 1.5, then the variable represented is a linear combination ‘(0.5*variable 1) + (0.5*variable 2)’. The mapping is similar to a fuzzy membership function between 2 variables, although at present there is no way to alter the membership contributions of each variable along this continuous space. Figure 3 shows the simple case of 3 variables, which is easily represented since each variable in the mapping is the neighbour of both other variables. When the number of variables is increased this mapping become more complex, since it is necessary to iterate all possible combinations of variable pairs to avoid any bias in the mapping. For example, with 4 variables the mapping is extended so that each variable is a neighbour to all other variables, giving a mapping order of 1,2,3,4,1,3,2,4. There are many possible mappings that allow this to occur, and for TSOGP a simple iteration of all possible combinations is used. The main criterion is that each possible association between any pair of variables is represented as part of the continuous space, so that no bias occurs.

![Figure 4: Derivation Tree for non-continuous var + const](image)

This mapping makes the expressions used somewhat more expressive than they would be without it. In some cases, especially when comparing with other systems, this may not be desirable. Therefore, an option exists that, when checked, will disable this mapping. The way this works is through replacing all ‘constant’ nodes that represent variable numbers (two more will be described below) with ‘number’ nodes containing random integer numbers, as shown in Figure 4. This of course also means that, in case an optimizer has been selected, the optimizer will not optimize variable numbers since they are no longer continuous. This happens before the replacement caused by selecting the ‘evolutionary’ optimizer.

### 2.2.3 A Simple Example

To clarify the approach, it is worth working through an example showing how the different options affect the representation of an equation and how the optimiser and evolution work together. Assume that there are 3 independent variables: `x`, `y` and `z`, and that the mapping of these variables to the continuous domain is as shown in Figure 5. If no optimisation or adaption is selected, the system is a standard evolutionary system, as shown in Figure 6. If the independent variables are mapped, then the `var` function contains a random real number, otherwise a random integer. Figure 7 shows an example equation, based on the grammar.
$G_{Exp}$ described in §3.2. This tree represents the equation 
\((0.75z + 0.25y) + ((0.81x + 0.19z) \times 10.84)\). If no variable mapping is allowed, then the variable indexes become random integer values, as shown in Figure 8. This tree represents the equation \(y + (z \times 10.84)\). For either case, with the standard evolutionary system, a population of such trees are initially created and the training scores are evaluated on the training data. Evolving the new population is then just applying crossover and mutation to the current population, and recalculating the training scores for each new generation. If crossover is applied, the variable mappings are maintained and passed between the two parents. If mutation occurs, a new subtree is randomly generated and, if the mapping is allowed, the \(\text{var}\) node is given a random real number, otherwise a random integer value.

The situation when optimisation is allowed is shown in Figure 9. Two possible optimisations are possible: if no variable optimisation is allowed then the optimiser just searches for better values of the real numbered constants in each equation during the training of scores. Hence, in Figure 7 and 8 the only value altered during the optimisation is the number 10.84. The value of the variables remains the same during this process, and the optimisation only occurs over the constant numbers in the equation. After the new population is evolved, the training is recommenced which involves once again reoptimisting the constants in each equation, based on the training data.

**Figure 5:** Continuous Mapping for variables \(x, y, z\)

**Figure 6:** Basic Evolutionary System without Optimisation

When optimisation is allowed over the variables, only the situation of Figure 7 is allowed, since the variable space must be continuous. Here calculating the training scores involves searching for better values of the \(\text{var}\) value and the constant numbers, therefore optimising the variables and constants within the equation. Note that this means for the equation of Figure 7 that the optimiser is performing a 3-dimensional search for a global minima defined by the error surface based on the training data. Each member of the population is optimised and, once this has occurred, the new population is evolved using crossover and mutation. Note that, although the optimised values of the variables/ constants are passed between population members during the evolution, they are discarded when optimisation of the training scores is commenced. This makes sense since the new structures that have been evolved have not yet been optimised, and so the assumption is that training using the optimiser will find better values of the current variable and constant mappings. Hence the title of the paper: "Evolving Structure - Optimising Content", since when variable mapping and optimisation is used the system evolves new structures, however the terminal values of the structures are determined by the optimisation routine. Although it may appear sensible to use the current values of the new population member (after evolution) as the starting point for optimisation this is not done. Since Powell and Simplex routines use several starting points in the search space to maximise the chance of finding the global optima, using the current values as a starting point does not necessarily reduce the cost of evaluations. Further, since constants and variables in an equation are highly context-sensitive there appears to be no reason to assume that after evolution the new positioning of the constants/variables is optimal.

The full version of the system has been shown previously in Figure 2. The addition of the adaptive training, with optional optimisation, occurs in the following manner. Adaptive training randomly selects a node in a equation derivation tree, deletes the subtree below the node, and randomly generates a new subtree (basically a mutation). If the new equation is an improved predictor with the training data then the new equation is kept, otherwise it is discarded. This process continues until no improvement can been found after a set limit of evaluations. If optimisation is not allowed, then the variable and constant values of each mutation are randomly filled.
If, however, optimisation is being used then the entire equation is optimised, without regard for the current values of the variable mappings within the old components of the equation.

\[
\text{Exp} + \text{Exp} \text{Exp} \text{var} 2 \text{Exp} \text{Exp} \text{var} 10.8 \text{Exp} 3
\]

Figure 8: A tree representing the equation \(2 + (3 \times 10.84)\)

\[
\text{Optimise Vars/Consts} \quad \text{Generate Population} \quad \text{Calculate Training Scores} \quad \text{Evolve New Population}
\]

Figure 9: The Evolutionary System with Training Optimisation

### 2.2.4 Time-Series Specific Functions

A number of functions are included with TSOGP to allow the temporal nature of time-series models to be explored. Although they are not used in the later experiments it is worth mentioning them to show how the continuous domain is handled with these functions.

The \(\text{hist}(v,t)\) function takes two arguments. The first argument decides which variable(s) to get historic values of, using the mapping described above. The second argument gives the number of steps back in time from the current time evaluation to obtain the value for this variable(s). The program allows the user to give a maximum allowed for this number of steps, and if the second argument is larger than this maximum, it wraps around. Noninteger numbers of steps are, again, interpolated smoothly.

The \(\text{avg}(v,t)\) function also takes two arguments, and is identical to \(\text{hist}(v,t)\) in the way it treats its arguments. It calculates the average of one (two) variable(s) over a number of timesteps back in history.

The system described can also be used for rule discovery. This has been implemented through an \(\text{if-then-else}(v,t,e)\) function. Here, the first argument is evaluated. If this argument is greater than zero, the result is the second argument. Otherwise, it is the third argument.

The requirement of a continuous function for optimisation is violated with this function. The problem lies, of course, in the first argument. Small changes in this argument may result in its status changing for just one of the datapoints in the training set. This can, however, result in large changes in the value of the function, as the values of the second and third argument may be very different. This means that the function is discontinuous, and that the error function is therefore also discontinuous.

This lack of continuity in the function is solved by using a \(\text{smoothif-then-else}(v,t,e)\) function. Again, the first argument is evaluated. If it is greater than one, the result is the second argument. If it is smaller than zero, the result is the third argument. Between those two values, the result is smoothly interpolated between the second and third arguments. This produces a continuous function between the two possible resulting values from the rule and therefore is appropriate for use with either selected optimiser.

### 3 Experiments

This section describes some preliminary experiments to demonstrate the performance of optimisation techniques in cooperation with an evolutionary system for timeseries model development.

#### 3.1 Lake Kasumigaura Data

Lake Kasumigaura is situated in the south-eastern part of Japan. It is a large, shallow water body where no thermal stratification occurs. Water temperatures vary widely, ranging from \(4^\circ\) in winter to over \(30^\circ\) in summer. The lake has high nutrient loadings and therefore phytoplankton abundance is high for the majority of the year. Given the reliance on light and temperature for growth there are clear seasonal patterns in the data.

A description of the variables used for this study are shown in Table 1. The data set is composed of ten years of daily data from 1989-1993 inclusive, and has been previously described in [9].

<table>
<thead>
<tr>
<th>Variable</th>
<th>Av ± Std.Dev</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ortho Phosphate (p)</td>
<td>14.14 ± 25.71</td>
<td>Mg/l</td>
</tr>
<tr>
<td>Nitrate (n)</td>
<td>520.56 ± 503.4</td>
<td>Mg/l</td>
</tr>
<tr>
<td>Secchi Depth (s)</td>
<td>85.43 ± 44.57</td>
<td>cm</td>
</tr>
<tr>
<td>Dissolved Oxygen (o)</td>
<td>11.2 ± 2.14</td>
<td>Mg/l</td>
</tr>
<tr>
<td>pH (ph)</td>
<td>8.74 ± 0.39</td>
<td>-</td>
</tr>
<tr>
<td>Water Temperature (t)</td>
<td>16.36 ± 7.79</td>
<td>°C</td>
</tr>
<tr>
<td>Chlorophyll-a (chla)</td>
<td>74.43 ± 42.51</td>
<td>Mg/l</td>
</tr>
</tbody>
</table>

Table 1: Lake Kasumigaura Water Quality Variables
3.2 Experimental Setup

The data was divided into a training set for three years from 1984 and an independent test set from 1988-1993. Note that the year 1987 was not used for training or test so that the two data sets did not share any common boundaries of data points and could be considered independent. The goal of the system was to evolve/optimise an equation for the prediction of chlorophyll-α, using the following grammar for linear expressions.

\[ G_{Expn} = \{
\text{StartSymbol} = Expn, \\
N = \{Expn\}, \\
\sum = \{+, -, \times, /, var, const\}, \\
P = \{Expn \rightarrow Expn + Expn \\
Expn \rightarrow Expn - Expn \\
Expn \rightarrow Expn \times Expn \\
Expn \rightarrow Expn/Expn \\
Expn \rightarrow const \\
Expn \rightarrow var(const) \}
\}

Note that the production \( Expn \rightarrow const \) defines an expression as being able to contain a random number, which based on the selected options may or may not be optimised. The production \( Expn \rightarrow var(const) \) indicates that a variable may be substituted into an expression, where the \( const \) value selects the appropriate variable based on the continuous mapping for variable selection, as described in §2.2.2.

The following comparisons were made using various combinations of evolution and optimisation:

1. Evolution without variable mapping
2. Evolution with variable mapping
3. Evolution with constant optimisation
4. Evolution with variable mapping and optimisation - population 10
5. Evolution with variable mapping and optimisation - population 100
6. Evolution with variable mapping and adaption

The grammar \( G_{Expn} \) is used for each of these comparisons, however different options are selected with TSGOP based on the type of comparison being performed. Since the most important measure of a search algorithm is the amount of effort required to discover a certain quality of solution, the performance of these different approaches was compared by measuring the final training and test errors after a set number (100000) of evaluations. Essentially this gave a measure of the quality of solution for a given effort. A summary of the initial system parameters for these tests is shown in Table 2. Although ideally a comparison between methods would involve keeping all parameters constant apart from the one being measured, this is not possible with this setup due to the large number of evaluations (typically over 1000 per population member per optimisation) required when using the Powell or Simplex optimiser. Therefore the population size was adjusted for each test to allow aspects of the system to be tested. The various population sizes are given in Table 2. Note that two versions of the evolution with constant and variable optimisation are tested: one with a population of 10 and one with a population of 100. These different populations have been selected to show the affect of evolution versus straight optimisation, since a population of 100 will only allow one or two populations to be evolved before the maximum evaluations are reached.

3.3 Results

The results for the 6 different runs are shown in Table 3. Since the optimiser will search within the variable space for any good combination of variables a small population size actually represents many different possible equations. The trade-off for this situation is that the optimiser must evaluate many
combinations of variables and/or constants. For example, if you consider the equation of Figure 1, this actually represents every possible equation that has a variable (or two combined variables) added to a number. To allow the comparisons of performance to be fair, a larger population, to allow greater diversity, was deemed necessary when no optimisation was applied.

Based on an independent t-test comparison of means the training error for Evol. with Const/Var Op is significantly lower at the 95% level than all other setups. Evol. with Const/Var Op2 and Evol. with Mapping and Adap. were also significant at the 95% level from the first 3 experiments, although they were not significant from each other. No training results were significant at the 95% level except for Evol. with Const/Var Op2 versus Evol. with Const/Var Op1.

4 Discussion

Based on Table 3 it is clear that using evolution in cooperation with the optimiser for variable and constant substitutions gives a significant improvement over straight evolutionary approaches. It must be noted, however, that this is just one test, and that it in no way proves that the hybrid system will always outperform a normal evolutionary system. In fact, Darwen [10], has argued that comparing that results of even changing a single parameter (for example, population size) within a system does not always show the effect of this parameter. Unlike the traditional scientific view of altering one variable to understand its relationship to a system, evolutionary systems are often more difficult to control and analyze in a reductionist manner.

One difficulty that had to be overcome was that when using the optimiser, with a limited number of evaluations, the evolution component of the system was not evoked to a significant degree. This was partially overcome by having the 2 tests shown in Table 2, where the population for the const/var experiments were set at 10 and 100. Since the setup with a smaller population that allowed evolution to occur more often was significantly better this implies that evolution combined with optimisation has improved the search, however optimisation is fundamental to the overall search.

The question of whether this hybrid approach will in general find "better" solutions with similar effort is still open. Certainly it appears that the use of a mapping for the variable space, giving a more expressive combination of variables, had a significant effect from the evolution-no-mapping to the evolution-with-mapping approach. This was shown not only by the first two experiments, but also with the final experiment using adaption. This implies that work should continue in considering how representations can best be mapped between the genotype (in our case, a real number representing a continuous variable space) and phenotype (the value of the variables). This may not only be relevant to equation discovery, but to general evolutionary systems.

Referring to Table 3, a significant improvement at the 95% level occurred between evolution-only (technique 2) and evolution-with-adaption (technique 6). Since the adaptive process is a hill-climbing algorithm this result implies that either the problem is one best suited to hill-climbing (which would also account for the optimiser performance) or that in general local hill-climbing is a useful approach that should be incorporated in most evolutionary search strategies. Further work is required to understand why adaption gave such an improvement, although it is clear that the fitness landscape (i.e. the problem space) is the main influence over these results.

Table 3 also shows that the test error (i.e. a measure of the generalisation of the evolved equation) is not significantly different between all but two of the described methods. In fact, the only significant difference at the level of test error is Evol. with Const/Var Op2 versus Evol. with Const/Var Op1. This is probably explainable due to the difference in size of the typical solutions produced by these two experiments, where using the larger population size, which minimised the number of crossover operations, tended to produce smaller solutions. Since generalisation and size of solution tend to be related this accounts for the difference. However, the fact that each technique cannot be easily distinguished based on generalisation does not mean that they are all equivalent. The significant difference in training errors indicates a difference in solution structure, and the sum of training and test errors implies a difference between the experiments with and without optimisation.

The constant/variable optimisation has one main drawback, namely the number of evaluations required to find some near-optimal values for the variables and constants of the expression. One possible approach to reduce this cost would be to only apply the optimisation after a set number of evolutions, or as a method of fine tuning the result after evolution is complete. Although this might be satisfactory it is difficult to justify why this approach would be taken if in fact the optimisation/evolution approach is at least as effective as evolution alone. Further tests are required to determine whether this hybrid approach offers any general advantages over straight evolution and the types of problems where this would be most suitable.

5 Conclusion

This paper presents preliminary results for a hybrid system incorporating a grammatical evolutionary system combined with an optimiser for variable and constant substitution. The positive results indicated by this work are encouraging, however further work applied to a number of standard machine learning cases and comparisons to other systems is required before any significant statements can be made. The use of a mapping between the genotype and phenotype for variable selection, and the combination of optimisation and evolution, has been shown to give significant improvements for the particular problem studied, and indicate some interesting future work for general evolutionary theory.
Acknowledgments

This work was funded by a University of Otago Research Grant. Paul McKitrick must also be thanked for testing and supporting the software development. The anonymous referees must also be thanked for their valuable comments that have extended and improved this paper.

Bibliography


