Method for the identification of explicit polynomial formulae for the friction in turbulent pipe flow

J. W. Davidson, D. Savic and G. A. Walters

ABSTRACT

The paper describes a new regression method for creating polynomial models. The method combines numerical and symbolic regression. Genetic programming finds the form of polynomial expressions, and least squares optimization finds the values for the constants in the expressions. The incorporation of least squares optimization within symbolic regression is made possible by a rule-based component that algebraically transforms expressions to equivalent forms that are suitable for least squares optimization. The paper describes new operators of crossover and mutation that improve performance, and a new method for creating starting solutions that avoids the problem of under-determined functions. An example application demonstrates the trade-off between model complexity and accuracy of a set of approximator functions created for the Colebrook–White formula.

Key words | genetic programming, least squares, polynomial expressions, symbolic algebra, symbolic regression

J. W. Davidson
D. Savic
G. A. Walters
Department of Engineering,
School of Engineering and Computer Science,
University of Exeter,
Exeter EX4 4QF,
UK

INTRODUCTION

The method for symbolic regression proposed by Koza (1992) is an alternative approach to curve fitting. The technique creates mathematical expressions to fit a set of data points using the evolutionary process of genetic programming. Like all evolutionary computing techniques symbolic regression manipulates populations of solutions (in this case mathematical expressions) using operations analogous to the natural evolutionary processes that operate in living organisms. The genetic programming procedure mimics natural selection as the 'fitness' of the solutions in the population improves through successive generations. The term 'fitness' in this instance refers to a measure of how closely expressions fit the data points.

Symbolic regression is based on genetic programming, which is similar to the technique of genetic algorithms. Genetic algorithms evolve solutions to a given problem, while genetic programming aims at evolving computer programs that solve a given type of problem. Genetic algorithms typically organize data in linear strings of fixed length in which different genes occupy different fields. Genetic programming uses a more flexible tree structure to represent computer programs. The length or depth of the trees can vary as programs evolve. The structure of the tree is reflective of the hierarchical structure of the computer programs they represent. Symbolic regression limits the structure of evolving programs to mathematical expressions only.

The major advantage of symbolic regression over numerical regression methods is that the user does not have to specify the form of the regression model in advance. Symbolic regression finds the form of expressions as well as parameter values. However, if the optimal form of the model is known, obtaining parameter values by numerical methods is computationally more efficient and ensures optimal values. The ideal method would combine the two approaches, using genetic programming to evolve the form of the expressions while simultaneously optimizing parameter values by numerical methods. The method described in the paper combines numerical

and symbolic regression for polynomial functions only. The paper demonstrates the method by finding an accurate polynomial approximator function for the Colebrook–White formula.

Incorporating numerical regression methods within the symbolic regression algorithm is difficult. To work effectively the algorithm must be able to perform the necessary algebraic manipulation of mathematical expressions to convert expressions from the wide variety of forms that genetic programming produces to equivalent expressions that are in a form that is amenable to numerical optimization. The program uses the method of least squares to optimize constant values in the expressions. To avoid singular matrices that can be produced by the method of least squares expressions must be free of all linear dependencies. The program achieves the necessary algebraic manipulation of expressions by the inclusion of a rule-based component.

The paper assumes the reader is familiar with genetic programming and symbolic regression. A substantial body of work exists that describes these methods in detail (Koza 1992, 1994; Kinnear 1994). A variety of applications have been presented since these early works and hydroinformatics has started to benefit from the use of genetic programming (Babovic 1996; Babovic & Abbott 1997*a*,*b*; Babovic & Keijzer 1999). The method presented here incorporates substantial modification to the original algorithm. The paper describes only those elements of the technique that are new and not part of the original algorithm.

The development of the new algorithm occurred in two phases with the incorporation of the rule-based and least squares optimization components in the first phase. The second phase involved the introduction of a new method to generate the starting population of solutions and new operators of crossover and mutation designed specifically for use with polynomial expressions. The new method for creating the starting population was necessary due to the problem of under-determined functions explained later in the paper. The new operators improve performance by exploiting characteristic features of polynomial functions and making use of an established statistical regression method, backward elimination. Appendix A describes the new operators and the backward elimination procedure.

METHOD

Improving on ephemeral random constants

The original symbolic regression method makes use of 'ephemeral random constants' (Koza 1992) to create the constant values in expressions. The procedure generates the values for constants during the creation of the starting population. Once the procedure has created the starting population, the values of constants in expressions do not change as expressions evolve unless mutation acts directly on a constant.

Instead of optimizing constant values, symbolic regression evolves parameter values for expressions by evolving expressions within the main expression. These 'expressions within expressions' create the required values from the available constants in the population. The use of ephemeral random constants rather than adjustable parameters results in complex expressions, which is undesirable for two reasons:

1. The regression problem is more difficult to solve. Rather than finding the form of the expression alone, the method must also find expressions that produce the required parameter values from a limited set of available constants.
2. The resulting expressions are more complex and therefore more difficult to interpret and use. The complexity introduced by 'expressions within expressions' often obscures the relation between the dependent variable and the independent variables.

The problem of determining optimal parameter values for an expression of a given form is one that is essentially solved. Methods such as least squares regression for polynomial expressions and Gauss–Newton iteration or the Marquardt method for nonlinear problems find the optimal parameter values for mathematical expressions. Incorporating a method to obtain the optimal parameters through numerical regression techniques may be an obvious approach to symbolic regression. However, the complex forms of expressions that genetic programming generates makes this approach difficult. As mentioned previously, the method makes use of a rule-based component to perform the necessary algebraic manipulation.

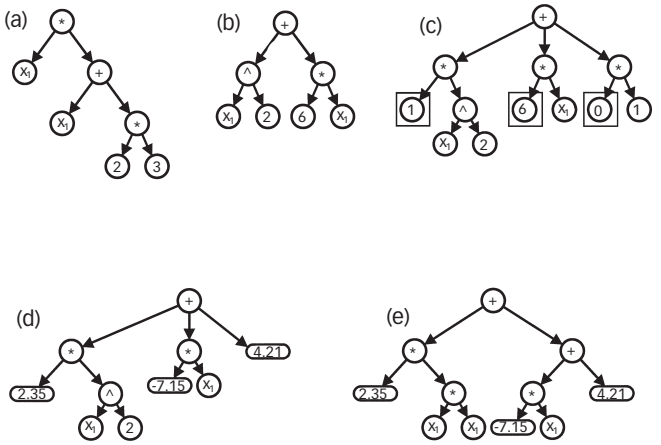

Figure 1 | Transformation and optimization of an expression.

The rule-based component converts expressions from the variety of forms produced by genetic programming to a 'standard' form.

Figure 1 illustrates how the rule base transforms expressions. Mathematical expressions appear in Figure 1 represented schematically as parse trees. A parse tree is a directed graph in which the nodes represent the elements in the expression, which consist of functions and terminals (variables and constants). In a parse tree representation function nodes have links directed downwards to point to each of the function's arguments. Evaluation of the expression proceeds from the bottom of the tree upward to the root node at the top of the tree. Figure 1a shows the parse tree representation of the expression x_1 $(x_1 + 2 \times 3)$.

The rule-based component for algebraic manipulation of expressions finds non-standard patterns in the parse tree and replaces them with their standard form equivalents. The rule bases consist of 53 rules. The technique involves the pair-wise comparison of terms in the expression and the transformation of expressions on the basis on five criteria:

1. Serialization of addition and multiplication operations;
2. Reduction of similar terms and constant expressions;
3. Sorting of terms and coefficients;
4. Distribution of multiplication into addition; and
5. Removal of identity expressions such as substituting x for $(0 + x)$.

One of the major benefits of using the rule-based component is that it provides the ability to determine the number of regression parameters implicitly represented in factored expressions produced through symbolic regression. The number of regression parameters, or degrees of freedom, is a more accurate representation of the complexity of regression models than the measures that symbolic regression procedures currently use.

Extensive testing of the rule-based component demonstrated that the simple strategy of repeated pair-wise comparison always succeeds in transforming any polynomial expression generated by genetic programming to the standard form. The transformation includes the removal of all redundant terms and coefficients and the removal of all linear dependencies between terms in the expression that would cause the production of a singular matrix in the least squares optimization stage.

Figure 1b shows the expression in Figure 1a transformed to standard form. The transformation process primarily consists of expanding the polynomial and reducing any expressions where possible. The intermediate steps in the transformation have been eliminated from the figure for brevity. The process requires the application of four rules. The first rule replaces the product of 2 and 3 with a single constant equal to 6 producing the expression $x_1(x_1 + 6)$. The next two rules perform distribution of multiplication in to addition $(x_1x_1 + 6x_1)$ and the final rule replaces the x_1x_1 term with x_1^2.

In the standard form the parse tree of a polynomial has a single addition operator at the root node. Each of the arguments to this addition operator represents a term in the polynomial expression, which should have a unique adjustable coefficient (a constant node). The expression in standard form in Figure 1b is not ready for parameter optimization since only one of the two terms $6x_1$ has an adjustable coefficient (6). The expression in Figure 1b does not have an intercept term as well (commonly referred to as b_0 in linear regression literature). A separate procedure (referred to here as 'Procedure 1') transforms the standard form expression to the form of the expression in Figure 1c in which all the required adjustable parameters have been included. A procedure separate from the rule base is required to perform this transformation because many of the terms and coefficients

introduced by the second transformation would be identified as redundant and removed by the rule base. Each of the three adjustable parameters is enclosed in a square in Figure 1c. The expression is now in the form required by linear regression shown below:

$$\hat{y}_i = \sum_{j=1}^{M} a_j z_{ji} \qquad \forall i : i = 1 \ldots n \,, \tag{1}$$

where

$\hat{y}$ is the least squares estimate of the target value;

M is the number of parameters;

n is the number of data points;

a_j is the jth adjustable parameter; and

z_{ji} is an expression based on the independent variables (x) evaluated at the ith data point. The expressions include no constants with the exception of integer powers of the independent variables. Alternatively, z_{ji} can consist of a single constant equal to 1 in the case of the intercept term, b_0.

The n equations result from evaluating the expression, now in standard form, at each of the n data points. These n equations form the normal equations used in the least squares optimization procedure. The standard form of the expression does not necessarily guarantee that the expression includes all of the adjustable parameters required by the normal equations. Procedure 1 converts expressions in standard form to the form required by the normal equations including an adjustable parameter (constant) for each term in the summation and ensures that every expression includes an intercept term.

A relatively straightforward procedure constructs and solves the normal equations producing the least squares optimal parameter values for the constants. The optimal values are back-substituted into the expression in Figure 1c at the nodes shown enclosed in squares. The rule base is used again to remove any redundancies in the expression producing the expression shown in Figure 1d. Figure 1d shows parameter values found through least squares optimization as lozenge-shaped nodes.

Usually the genetic programming operators of crossover and mutation require that all multiplication and addition operators take two arguments only. Problems occur with genetic programming operators if the power operator is allowed to remain in the expressions as well. A procedure (referred to as 'Procedure 2') expands all powers to repeated multiplication operations and introduces addition and multiplication operators where required to convert the expressions to the form in which these operators take only two arguments. Figure 1e represents the final expression as it would appear in the population during symbolic regression.

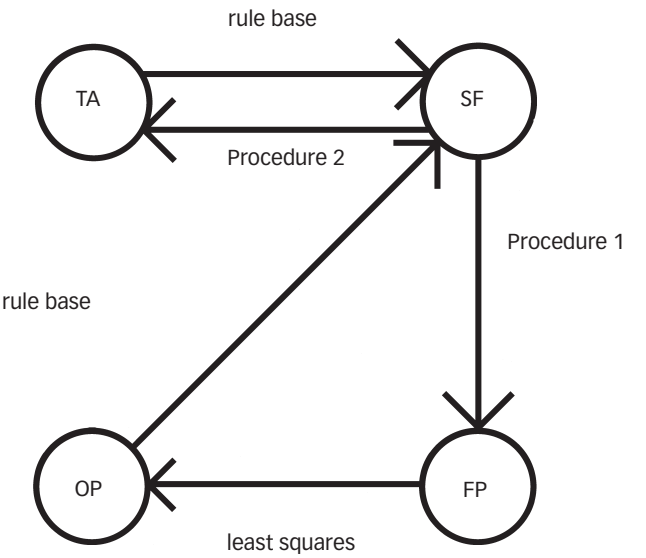

Figure 2 | State space diagram of expression transformations.

Figure 2 shows a state space diagram of all the transformation processes that polynomial expressions may undergo. The node designated TA (two argument) refers to expressions in the form in which all internal nodes (addition and multiplication operators) take two parameters only and no power operators are used. This form is compatible with conventional genetic programming operations of mutation and crossover and is consistent with the type of expressions created through genetic programming if the set of functions is limited to addition and multiplication only. Parse trees in Figure 1a and Figure 1e are in TA form.

The node designated SF (standard form) refers to the expanded polynomial form with all linear dependencies removed. Expressions in the TA form are converted to SF form by the rule base and can be converted back to TA form by Procedure 2. Figure 1b is an example of an SF

parse tree. The node FP (full parameter) designates an expression in the form in which all terms have adjustable coefficients and include an intercept term. Procedure 1 creates FP expressions from SF expressions. Figure 1c is an example of an FP expression. Least squares optimization finds the optimal values for all of the adjustable coefficients in an FP expression and converts it to an OP (optimal parameter) expression. The rule base can be used to convert an OP expression to an SF expression removing any redundant coefficients or terms thereby simplifying the expression.

The complete procedure for optimizing coefficient values takes an expression created through genetic programming and applies the rule base, Procedure 1, least squares optimization, the rule base, and finally Procedure 2. This procedure is applied to every solution in the population before evaluation to ensure that all solutions are evaluated with optimal parameter values.

The problem of under-determined functions

The incorporation of algebraic transformation and least squares parameter optimization enables the technique to identify a very important problem associated with statistical regression, namely the problem of under-determined functions. Under-determined functions result when there are more degrees of freedom (adjustable parameters) in a model than data points to fit. The result is that infinitely many sets of parameter values will fit the data exactly. A cardinal rule of regression is that methods should not create under-determined functions. Carpenter & Barthelemy (1993) describe a similar problem that can occur in neural networks.

Under-determined or exactly determined functions have an error equal to zero once the adjustable parameters are optimized. Within the framework of an evolutionary technique, populations of solutions will quickly become dominated by under-determined or exactly determined functions because these solutions are 'perfectly fit'. In the original approach to symbolic regression the parameters are not adjustable. Therefore the populations of solutions do not necessarily become dominated by under-determined and exactly determined solutions. However, the problem persists largely unacknowledged.

The use of the rule-based component revealed that symbolic regression could produce inherently factored expressions for which very many degrees of freedom exist as represented by the number of unique terms in the expanded expression. The algebraic transformation procedure that distributes multiplication over addition revealed the potential to produce a combinatorial explosion of terms in the final expanded polynomial expression prior to parameter optimization.

As a function of the length of the expression (the number of nodes in the parse tree) the number of degrees of freedom of this worst-case condition is:

$$df = 2^{(l+1)/2}, \tag{2}$$

where df is the number of degrees of freedom of the expanded polynomial; and l is the length of the expression. In the upper bound, worst-case scenario the number of unique terms in a polynomial expression increases exponentially with the length of the expression. The relation is supported by empirical evidence and has serious implications concerning the possibility of generating under-determined functions.

The solution to the problem of under-determined functions is to regard these solutions as infeasible. The best approach is to avoid producing such solutions in the starting population and through the operations of cross-over and mutation. Appendix A describes new operators and a new method for generating starting solutions that does not use expressions in the two argument (TA) form in Figure 2. Using these methods the algorithm can restrict the number of terms in expressions thereby eliminating the possibility of creating under-determined functions.

RESULTS

The Colebrook–White formula calculates the friction factor f on the basis of the Reynolds number, Re, and the relative roughness, K/D (the ratio of the equivalent sand roughness, K, to the diameter of the pipe, D):

$$\frac{1}{\sqrt{f}} = -2\log_{10}\left\{\frac{2.51}{Re\sqrt{f}} + \frac{K}{3.7D}\right\}. \tag{3}$$

The Colebrook–White formula is an implicit equation in which f appears on both sides of the equation. Finding the value of f is awkward, requiring the use of iteration or alternatively by the use of the Moody diagram.

The Moody diagram includes Reynolds numbers for turbulent flow that range from 4,000 to 10^8 and relative roughness values from 0.05 to 10^{-5}. The wide range in the order of magnitude of these values introduces the problems associated with ill-conditioned matrices for regression methods that use matrix inversion. To avoid ill-conditioning the data set used in the example consists of a subset of values in the Moody diagram. The objective of the example application is to find an explicit polynomial function for the friction constant for Reynolds numbers ranging from 100,000 to 1,000,000 and relative roughness values from 0.001 to 0.01.

The data set consists of a two-dimensional grid of 100 data points, created from 10 Reynolds values selected in equal increments of 100,000 on the interval of 100,000 to 1,000,000, and 10 relative roughness values selected in equal increments of 0.001 on the interval of 0.001 to 0.01. The target friction values for the 100 points are values obtained using the Colebook–White formula (Equation 3). The friction values, calculated iteratively using Equation 3, were considered to have converged when the ratio of the difference in values obtained between two iterations and the value obtained from the most recent iteration was less than 10^{-8}. The friction values on the selected interval range from a maximum value of 0.0385035 to a minimum value of 0.0199435.

To further reduce ill-conditioning the independent and target values were transformed to fit on a scale ranging from 0 to 10. Equations 4 through 6 transform the two independent variables, x_1 and x_2, and the value, y:

$$x_1 = 1000k/D \tag{4}$$

$$x_2 = Re/10^5 \tag{5}$$

$$y = 10\,\frac{f - 0.0199435}{0.0385035 - 0.0199435}\,. \tag{6}$$

The method requires the user to specify two parameter values that limit the size of the search space. The two parameters are the maximum power value and the maximum number of terms in an expression. For this example the maximum power value was 6 and the maximum number of terms in an expression was 14. These parameter values create a total number of possible expression forms of $\binom{48}{13}$ or 1.929×10^{11}.

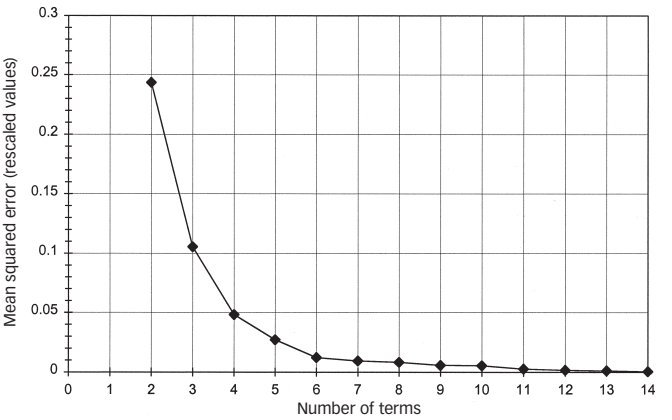

Figure 3 | Trade-off between accuracy and complexity of best solutions.

The program maintains copies of the best solutions encountered for each expression length from 2 to the maximum of 14 terms. Figure 3 shows the best MSE (mean squared error) values obtained after 100 generations. Table 1 lists the solution expressions and their corresponding MSE values. The 14-term polynomial in Table 1 is the best approximation. Haaland (1983) provides Equation 7 as an explicit approximation to the Colebrook–White formula below. In comparison to Equation 7 the 14-term polynomial is 33% more accurate over the selected region when the predicted values are transformed back to the original scale (using the inverse of Equation 6 to obtain f from $\hat{y}$). The largest absolute value of error of the 14-term polynomial over 100 selected data points was 0.000194 while the absolute value of error for Equation 7 was 0.000208. The sum of absolute values of errors for the 100 data points was 0.002967 for the 14-term polynomial and 0.00455 for Equation 7.

$$\frac{1}{\sqrt{f}} = -1.8\log_{10}\left[\frac{6.9}{Re} + \left(\frac{K}{3.7D}\right)^{1.11}\right]. \tag{7}$$

Table 1 | Best polynomial expressions.

Terms	MSE	Expression
2	0.243627	$1.012242914x_1 + 0.2234608476$
3	0.105384	$-2.145754708(10^{-5})x_1^5 + 1.210918343x_1 - 0.3954177254$
4	0.0483078	$1.254894577(10^{-4})x_1^5 - 1.673751306(10^{-3})x_1^4 + 1.503667565x_1 - 1.0103815$
5	0.0271015	$7.257150624(10^{-3})x_1^3 - 0.1797109638x_1^2 + 2.22415984x_1 - 2.574376943(10^{-5})x_2^4 - 1.653281513$
6	0.0122998	$7.257150624(10^{-3})x_1^3 - 0.1797109638x_1^2 + 2.22415984x_1 + 5.676887479(10^{-4})x_2^3 - 0.1043049727x_2 - 1.3165467$
7	0.00943032	$7.257150624(10^{-3})x_1^3 - 1.442425901(10^{-5})x_1^2x_2^3 - 0.1753476255x_1^2 + 2.333723942(10^{-3})x_1x_2^2 + 2.134311469x_1 - 0.1271286765x_2 - 1.019290483$
8	0.00827975	$3.935181523(10^{-7})x_1^4x_2^2 + 6.923840749(10^{-3})x_1^3 - 2.362521433(10^{-6})x_1^2x_2^4 - 0.171286766x_1^2 + 2.745220178(10^{-4})x_1x_2^3 + 2.134450733x_1 - 0.1247294884x_2 - 1.027286383$
9	0.0056247	$-1.038686347(10^{-5})x_1^6 + 1.540829606(10^{-4})x_1^5 - 0.1598131752x_1^2 + 7.093578814(10^{-3})x_1x_2 + 2.210907893x_1 - 2.610219679(10^{-4})x_2^4 + 3.816806857(10^{-3})x_2^3 - 0.227939205x_2 - 1.018236261$
10	0.0053075	$1.829368793(10^{-10})x_1^4x_2^6 + 5.822540879(10^{-7})x_1^4x_2^2 - 1.099257686(10^{-3})x_1^4 + 3.015141939(10^{-2})x_1^3 - 5.99494835(10^{-6})x_1^2\,x_2^4 - 0.3320683826x_1^2 + 4.749383542(10^{-4})x_1x_2^3 + 2.538376364x_1 - 0.1484685662x_2 - 1.259071682$
11	0.00274283	$-2.347843515(10^{-9})x_1^4x_2^4 - 1.034700788(10^{-3})x_1^4 + 3.015141939(10^{-2})x_1^3 - 0.3472553852x_1^2 - 1.114753295(10^{-3})x_1x_2^2 + 2.164677064(10^{-2})x_1x_2 + 2.605905979x_1 - 2.216792999(10^{-3})x_2^3 + 5.431016405(10^{-2})x_2^2 - 0.4539718364x_2 - 0.9991651301$
12	0.00173278	$6.190147186(10^{-12})x_1^6x_2^5 - 1.106851934(10^{-3})x_1^4 + 1.407431777(10^{-6})x_1^3x_2^2 + 3.098232643(10^{-2})x_1^3 - 2.627594044(10^{-4})x_1^2x_2^2 - 0.3415729834x_1^2 + 3.225424878(10^{-2})x_1x_2 + 2.513597566x_1 - 2.749741718(10^{-3})x_2^3 + 6.273377558(10^{-2})x_2^2 - 0.5268614664x^2 - 0.7669618375$
13	0.00119686	$-4.824683142(10^{-12})x_1^6x_2^6 - 1.875226683(10^{-6})x_1^6 + 6.686137342(10^{-10})x_1^5x_2^5 + 1.42377513(10^{-2})x_1^3 - 4.880507527(10^{-4})x_1^2x_2^2 - 0.2426854662x_1^2 + 4.556722648(10^{-2})x_1x_2 + 2.247419776x_1 + 5.67249706(10^{-4})x_2^4 - 1.568498481(10^{-2})x_2^3 + 0.1628852637x_2^2 - 0.840682524x_2 - 0.2943737513$
14	0.000528698	$1.222995307(10^{-5})x_1^6 - 2.242748136(10^{-10})x_1^5x_2^5 - 2.482162347(10^{-4})x_1^5 + 9.286977109(10^{-6})x_1^3x_2^3 + 3.645038671(10^{-2})x_1^3 - 1.18044694(10^{-3})x_1^2x_2^2 - 0.3849323423x_1^2 + 6.598401765(10^{-2})x_1x_2 + 2.522401137x_1 + 6.471827292(10^{-4})x_2^4 - 1.776888826(10^{-2})x_2^3 + 0.1829816121x_2^2 - 0.9369530943x_2 - 0.3698214152$

DISCUSSION

In earlier work Watson & Parmee (1996) used a modified symbolic regression algorithm to find an explicit approximator function for the Colebrook–White formula. The modification to the algorithm was the incorporation of micro-evolution to improve on ephemeral random constants. Appendix B explains micro-evolution and disadvantages with the approach. Watson & Parmee (1996) compared their best result with Haaland's equation. However, they incorrectly report an inaccurate version of Haaland's formula with a value 3.6 in place of the value of 1.8 in Equation 7. Unfortunately Equation 8, the best solution offered by Watson & Parmee (1996), is similarly inaccurate. The sum of the absolute value of error for the 100 data points for Equation 8 is 2.296928.

$$\frac{1}{\sqrt{f}} = -3.8364\log_{10}\left[\frac{0.2097}{D} + \frac{11.1001}{Re}\right]. \quad (8)$$

For manual calculation it is desirable to have as small and compact an expression as possible. A smaller expression reduces the number of steps in the calculation and is easier to remember. However, for computer simulation only two criteria are important: speed and accuracy. Both the Colebrook–White formula and Haaland's equation are reasonably compact. However, the many iterations required to solve the Colebrook–White formula represent substantial computational effort. Haaland's equation, while less accurate, is more efficient. However, Haaland's equation requires the use of transcendental functions, in the form of a logarithm and a non-integer power. The actual computational effort will depend on the efficiency of the computer's library functions, which are likely to calculate the values of transcendental functions to an accuracy greater than necessary by the summation of a lengthy series. By contrast, the operations in the polynomial function can be performed at high speed by the floating-point hardware and if parallel processing is available the terms of polynomial expression can be evaluated in parallel. The 14-term polynomial function, while awkward, is both more accurate and computationally more efficient than Haaland's equation. It should be stressed that accuracy of the function outside the selected range (Reynold's numbers between 100,000 and 1,000,000 and relative roughness values of 0.001 to 0.01) is unknown and likely to be unacceptable.

CONCLUSIONS

For polynomial functions the best features of symbolic regression and numerical regression have been combined into a single method requiring substantial modification to the original symbolic regression algorithm. The hybrid method works by transforming expressions to equivalent forms that eliminate any linear dependencies between parameters. The method for creating populations of starting solutions and the operators of crossover and mutation in the new algorithm are different from the original method to avoid the production of under-determined functions.

The advantage of the new method is the clear trade-off between accuracy and efficiency offered by functions of different complexity as illustrated in Figure 3 and Table 1. For the example problem the need for greater computational efficiency has increased in recent times. The use of large 'all pipe' simulation models of water distribution networks, and real-time and extended simulation models, has contributed to that need. The example demonstrates that there are instances in which relatively short polynomial functions can accurately approximate more computationally intensive functions requiring transcendental functions. The strength of the new technique presented here is its ability to find the form of these functions from within an extremely large set of possible combinations.

ACKNOWLEDGEMENTS

The U.K. Engineering and Physical Sciences Research Council supported this work, grant GR/L67189.

REFERENCES

Babovic, V. 1996 *Emergence, Evolution, Intelligence; Hydroinformatics*. Balkema, Rotterdam.

Babovic, V. & Abbott, M. B. 1997*a* The evolution of equation from hydraulic data, Part I: Theory. *J. Hydraulic Res.* **35** (3), 397–410.

Babovic, V. & Abbott, M. B. 1997*b* The evolution of equation from hydraulic data, Part II: Applications. *J. Hydraulic Res.* **35** (3), 411–430.

Babovic, V. & Keijzer, M. 2000 Genetic programming as a model induction engine. *J. Hydroinformatics* **2**(1), 35–60. (In the press.)

Carpenter, W. & Barthelemy, J.-F. 1993 Common misconceptions about neural networks as approximators. In *Neural Networks and Combinatorial Optimization in Civil and Structural Engineering* (ed. B. H. V. Topping & A. I. Khan), pp. 1–9. Edinburgh: Civil-Comp Limited.

Draper, N. & Smith, H. 1998 *Applied Regression Analysis.* New York, John Wiley and Sons.

Haaland, S. 1983 Simple and explicit formulas for the friction factor in turbulent pipe flow. *J. Fluids Engng* **105**, 89–90.

Kinnear, K. 1994 *Advances in Genetic Programming.* Cambridge, Massachusetts: MIT Press.

Koza, J. 1992 *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* Cambridge, Massachusetts: MIT Press.

Koza, J. 1994 *Genetic Programming II: Automatic Discovery of Reusable Programs.* Cambridge, Massachusetts: MIT Press.

Watson, A. & Parmee, I. 1996 Systems identification using genetic programming. In *Proceedings of ACEDC '96, PEDC*, University of Plymouth, UK.

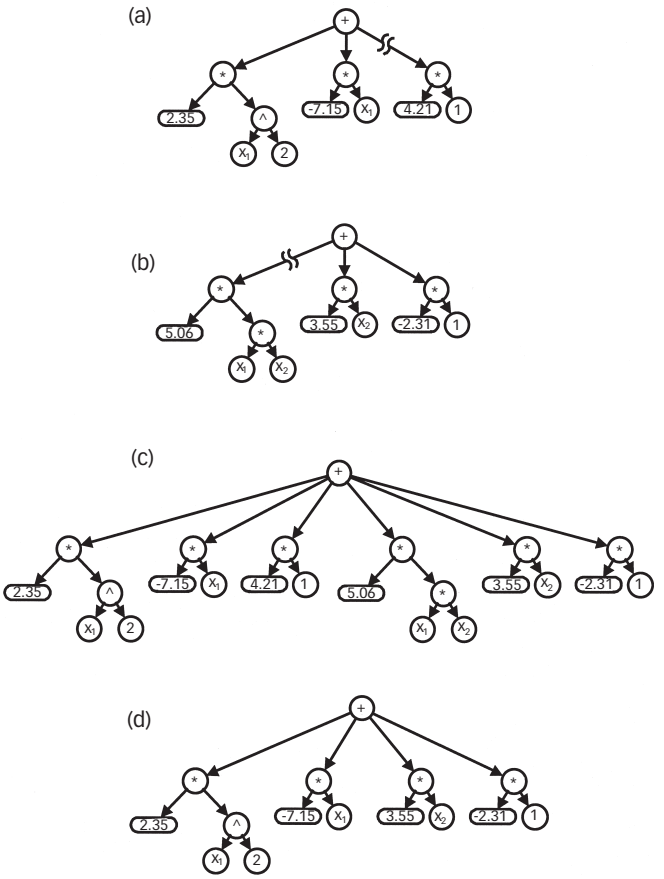

Figure 4 | New crossover operators.

APPENDIX A. OPERATORS AND PROCEDURES

New operators

In addition to the problems of exponential growth in the number of terms in polynomials and the potential for under-determined functions, there are other problems associated with the conventional genetic programming operators of crossover and mutation. The conventional genetic programming crossover operator splits a parse tree at any possible location creating a sub-tree and super-tree. Crossover randomly selects the location for splitting in two parent solutions and creates offspring by combining the super-tree from one parent with the sub-tree of the other. The method is described in greater detail in Koza (1992). The difficulty with this approach when applied to polynomial expressions is that the offspring usually bear little resemblance to the parent expressions from which they derive, particularly if the expressions then undergo algebraic manipulation. The same is true of the commonly used mutation operators of allele and shrink mutation, described in Koza (1992). The symbolic regression method described in the paper makes use of new operators of crossover and mutation specifically designed to preserve polynomial terms as they are transferred from parent to descendant solutions. These operators ensure descendant solutions consist of the same powers and cross-products of independent variables as are found in the parent solutions.

A polynomial consists of a sum of terms. Each of the terms consists of an adjustable parameter multiplied by a cross-product of powers of the independent variables. The parse tree of a polynomial in standard or full parameter form can have only one addition operator at the root of the tree, corresponding to the summation in Equation 1. Figure 4a,b shows two parse trees in full parameter (FP) form. Each of the trees has been split into a sub-tree and a

super-tree. In both cases the split occurs between the root addition operators and one of its arguments. A split that occurs at this level can add or remove terms to the root addition operation but will not alter the coefficients that make up the polynomial terms. Conventional genetic programming crossover allows the splitting to take place anywhere within the parse tree. Splitting the tree below the top (summation) level results in modification of the terms within the polynomial expression. Modifying the terms in this manner produces expressions that bear little resemblance to their parents and can even produce non-polynomial expressions that are not amenable to optimization by least squares. The new crossover operators explained in this section operate on standard form (SF) or full parameter (FP) form expressions only and split parent solutions at the top (summation) level only as shown in Figure 4a,b.

'Add' Crossover

An example of this operation is illustrated in Figure 4a–c. Figure 4a,b represents parent solutions and Figure 4c is the child solution. The parent solutions are not actually split by this operation as illustrated in Figure 4a,b. Instead the operation sums all the terms in the two parent expressions under one addition operator as in Figure 4c. Next, the rule base operates on the resulting expression to transform it to standard form (SF) removing any duplicate terms. Figure 4c shows the results of 'add' crossover before the application of the rule base. The rule base will reduce the duplicate intercept terms (4.21×1) and (-2.31×1) to a single term, 1.9. Then Procedure 1 operates on the expressions to convert them back from standard form (SF) to full parameter (FP) form. The final step applies least squares optimization to the resulting expression to obtain the best constant values.

Expressions created by 'add' crossover tend to grow in length. The user of the symbolic regression program specifies a maximum number of terms allowed in expressions before using the program. This maximum number of terms remains constant throughout the entire execution of the program. If 'add' crossover creates an expression that exceeds the number of terms, the procedure removes terms selected at random from the expressions until the expression no longer exceeds the maximum length.

'Set' Crossover

Figure 4a,b,d illustrates an example of 'set' crossover. As with 'add' crossover the operation requires the specification of a maximum number of terms. However, the maximum number of terms for this operator does not remain constant throughout the execution of the program. Normally the maximum number of terms is specified at a low number and gradually increased to the maximum specified for 'add' crossover. The next section, which describes the method for generating the starting population, explains the reason for the gradual increase in the maximum number of terms.

The 'set' crossover operation generates a random number between 1 and the maximum number of terms. This number specifies the number of terms the operation will take from the first parent. For the parent solutions in Figure 4a,b the maximum number of terms is 4. The random number of terms selected from the first parent solution, Figure 4a, in this instance was 2. The procedure retains the first two terms in the expression and removes all remaining terms as illustrated by the broken line in Figure 4a. The next step removes leading terms from the second parent solution until the number of terms in the remaining expression is equal to the difference of the maximum number and the number selected from the first parent. In the example the procedure removes leading terms from the second parent until only 2 terms remain, as illustrated in Figure 4b. The procedure adds what remains of the two expressions together by the same process as 'add' crossover. If the second parent solution does not have enough terms then the procedure simply uses all the terms in the second parent. Figure 4d shows the resulting expression. As with 'add' crossover the rule base transforms the resulting expression to standard form (SF); then Procedure 1 transforms the expression to full parameter form (FP) and finally least squares optimization finds the best constant values.

Experience has shown that when the maximum number of terms is low (six or less) the resulting child solution usually has the maximum number of terms. As the

maximum is increased above eight, solutions rarely have the maximum number of terms. The smaller number of terms is likely due to two reasons: (1) the second parent solution may not have enough terms to make up the full complement; and/or (2) the rule base reduces common terms in the contribution of both parents.

Backward Elimination

Backward elimination is not an evolutionary procedure, but rather, it is a statistical procedure commonly used within the method of stepwise regression, described in Draper & Smith (1998), to eliminate non-significant terms. Unlike conventional symbolic regression, all the statistics normally associated with linear regression are available for use because the method uses least squares optimization to obtain parameter values for each solution. Backward elimination removes terms on the basis of the associated *t*-statistics. If any terms have *t*-statistics less than a specified level of significance (customarily 95%) the procedure eliminates one term, the least significant term, then re-evaluates the new values for the remaining parameters. The process repeats until no insignificant terms remain.

Experience with backward elimination has shown that the procedure often eliminates terms while simultaneously improving the MSE, the statistic used as the fitness function. Considered with 'add' and 'set' crossover, the three procedures exert pressure to alternatively increase the length of expressions ('add' crossover), keep the length roughly the same ('set' crossover), or decrease the length (backward elimination). The algorithm uses the three procedures in combination to ensure that each generation consists of expressions of a wide variety of lengths enabling an assessment of the trade-off between complexity and accuracy.

'Close' Mutation

This type of mutation operates in one of three modes selected at random. The operation alters a single term in an expression. In the first mode the procedure increases the power of one of the independent variables by one. The second mode decreases the power of an independent variable by one and the third mode increases the power of one independent variable while simultaneously decreasing the power of another.

'Wild' Mutation

This form of mutation creates a new term at random and adds that term to the expression. The procedure selects powers of each of the independent variables at random as integer values between zero and a specified maximum power value.

Procedure for Generating the Starting Population

The method creates the starting population by an exhaustive search of all expressions of the form below:

$$\hat{y} = C_1 x_1^i x_2^j + C_2, \tag{9}$$

where

C_1, C_2 are the parameters obtained through least squares regression;

$\hat{y}$ is the least squares estimate of the target y value; and

i, j are values between 0 and the selected maximum value for powers.

The population undergoes a series of generations using 'set' crossover with the maximum number of terms in expressions increased by 1 up to the selected maximum. Each generation produces 100 new solutions and the best 40 are chosen to form the next generation. The process of gradually increasing the lengths of expressions creates the starting population and replaces the conventional genetic programming methods, usually a combination of 'ramped', 'full' and 'grow' methods (Koza 1992). The conventional methods tend to produce a combinatorial explosion of terms resulting in under-determined functions.

Once the program has created the starting population, a combination of 'set' and 'add' crossover operators and backward elimination ensure that each generation contains expressions of a variety of lengths. These iterations make use of roulette wheel selection with fitness based on the MSE according to Equation 10.

$$F_i = \mathrm{MSE}_{\mathrm{worst}} \times 1.2 - \mathrm{MSE}_i, \tag{10}$$

where

F_i is the fitness of solution i (relative proportion of the roulette wheel);

MSE_{worst} is the mean squared error of the worst solution in the population; and

MSE_i is the mean squared error of solution i.

APPENDIX B. DISADVANTAGES OF MICRO-EVOLUTION

The approach Watson & Parmee (1996) use to optimize, or evolve, constant values in expressions incorporates genetic algorithms as a form of micro-evolution within the symbolic regression procedure. Symbolic regression creates the form of the expressions and genetic algorithms optimize the parameter values. For each generation the genetic algorithm must operate on several populations of expressions, one for each solution produced by symbolic regression.

Micro-evolution represents an improvement over ephemeral random constants. However, there are difficulties with the micro-evolutionary approach when compared with numerical parameter optimization methods.

1. *Bounds and granularity*. Crossover and mutation used by genetic algorithms normally require parameter values in a discrete form with the unavoidable upper and lower bounds that such coding schemes introduce. Optimal parameter values may exist outside the bounds or between available discrete values.
2. *The approach does not guarantee optimality*. Genetic algorithms do not guaranty the optimal solution after any given number of iterations.
3. *Computational effort*. The approach introduces a massive increase in computational effort requiring an entire genetic algorithm population for each expression in the genetic programming population. This necessarily requires the execution of many generations for each micro-population with each generation of the macro-population.
4. *Statistics*. Numerical regression methods provide statistics to assess the fitness of expressions on the basis of more sophisticated criteria than those normally employed by symbolic regression. These criteria include the number of degrees of freedom of the expression, which is a true measure of the complexity of the expression, the MSE, the t-statistics of individual parameters and the F-statistic of the entire model.