Introducing knowledge into learning based on genetic programming

Vladan Babovic

ABSTRACT

This work examines various methods for creating empirical equations on the basis of data while taking advantage of knowledge about the problem domain. It is demonstrated that the use of high level concepts aid in evolving equations that are easier to interpret by domain specialists. The application of the approach to real-world problems reveals that the utilization of such concepts results in equations with performance equal or superior to that of human experts. Finally, it is argued that the algorithm is best used as a hypothesis generator assisting scientists in the discovery process.

Key words | empirical equations, genetic programming, hydraulics, sediment transport, strong typing, symbolic regression, units of measurement

Vladan Babovic (corresponding author)
Faculty of Engineering,
National University of Singapore,
1 Engineering Drive 2,
117 576 Singapore
E-mail: *vladan@nus.edu.sg*

INTRODUCTION

Automatic Programming is one of the most illusive goals in Artificial Intelligence. We are hoping to program computers by telling them what we want to achieve without having to explicitly instruct them *how* to achieve such goals.

What is it that we want from these programs anyway? Do they just need to be accurate or should we also be able to interpret them? From the perspective of scientific discovery, the ambition is clear: we are looking for a learning machine capable of finding an accurate approximation of a natural phenomenon, as well as expressing it in the form of an interpretable equation. However, this bias towards interpretability creates several new issues. The computer-generated hypotheses should take advantage of the already existing body of knowledge about the domain in question. However, the method by which we express our knowledge and make it available to a learning machine remains rather unclear. Or might it be better simply to ignore the knowledge altogether and simply fit the data? These investigations are the main objective of the paper.

doi: 10.2166/hydro.2009.041

DATA AS A SOURCE OF INFORMATION

Suppose we are confronted with the task of modelling an unknown or poorly understood system. In such situations, a logical starting point is the design of measurement campaigns and the collection of data. We usually measure forcing variables (those outside the system) and simultaneously the response of the system in view of the change of the state of the system (state- or internal variables), and the change in corresponding output of the system (resulting functions). After enough data of sufficient quality are collected, we can attempt to identify the system. Then, three possible scenarios can occur (Kompare 1995), described as follows.

1. Nothing useful can be concluded from the observations. This can happen if the measuring campaign was poorly designed, carried out over an insufficiently long period of time or if relationships among variables simply do not exist. More elaborate observations are needed to improve the situation.

2. Sometimes we may end up with a statistical, black box model. With this category of models we will be able to predict the proper behaviour of the system, although we will not be able to characterize its intrinsic structure and behaviour. In other words, we will be able to say what the model does, but not how. In addition to this, we will not be able to guarantee the behaviour of such a model in regions not covered by the data from which the model was constructed. This is due to the fact that the model covers only the relationships found within the given data.
3. In some cases, we may be able to recognize patterns within the data and infer from these patterns information about basic processes in the observed system. After repeated measurements we should be able to develop a conceptual (mechanistic) model. Such a model is a so-called white box, or transparent model and we should be able to say what and how the model does. Due to the conceptual background of the model, we are much more certain that the model will represent reality. This also helps when using the data out of the range in which the model was constructed.

In making the most of experimental data it is generally desirable to express the relation between the variables in a symbolic form: an equation. Each equation can be regarded as a collection of signs, which constitutes a model of an object, process or event. Data, on the other hand, remain as mere data to the extent that they remain a collection of signs that does not serve as a model. In a view of the approximate nature of the functional relation, such an equation is described as empirical. No particular stigma should be attached to the name since many ultimately recognized chemical, physical and biological laws have began as empirical equations.

Common methods for finding white box models on the basis of data usually involve a dimensional analysis and subsequent curve-fitting by hand or automatic means. Genetic programming (GP) (Koza 1992) is a technique that can be utilized to find an approximation of data in the symbolic form of an equation. One of the advantages of genetic programming over more standard methods for regression is that an overall functional form need not be explicitly assumed. The technique simultaneously searches for both the functional form of an equation and fits coefficients. The symbolic nature of the generated solutions is another great advantage. This is particularly pronounced in the natural sciences, where a symbolic answer in a language of mathematics provides a great benefit over methods that, as a result of fitting, produce only coefficients.

Unfortunately, the solutions produced by genetic programming are not always easily interpreted. The size of the solutions produced can hinder interpretation. Setting the size to low limits hinders the search efficiency.

This paper suggests an approach in which high level concepts, such as the dimensions (physical units of measurement) of the data, are used as an additional source of information in order to help create as well as check the validity and usefulness of expressions created on the basis of data. Rather than ignoring dimensions altogether, or proposing dimensionless formulae (i.e. based exclusively on dimensionless numbers), the objective is to create fully dimensioned formulae. It is postulated that such formulae can be easier to interpret by domain specialists in the *physical symbol system*. In this sense, the GP-produced equations form a set of hypotheses in and about the domain, stated in the symbolic language (of equations). Rather than producing a black-box approximation of problems, the aim is to provide interpretable statements that can be used to understand the problem better.

The structure of the paper is as follows. First, we introduce units of measurement as a type system. An approach to typing is then presented, followed by background knowledge about the problem and the data collected. Sections are devoted to various methods of fusion of the data and knowledge and to quantitative analysis of experimental results. Finally, the paper concludes with a discussion and conclusions.

UNITS OF MEASUREMENT AS A TYPE SYSTEM

One of the main instruments for interpreting equations is the system of physical units of measurements (uom). This system is utilized in engineering and sciences to make a connection between the symbols in the mathematical formulae and the physical world they describe. The uom system can be viewed as a type system, where the exponents of the

physical dimensions form real-valued types. (Although integer exponents are most easily interpreted, fractional exponents are often used, especially for problems where solutions based on first principles are unattainable. A well-known example from the field of hydraulics is empirical roughness expressions. These are stated in units of square root of length per second squared. This is necessary for any roughness equation to be able to be used in combination with an overall hydraulic model of flow that *is* based on first principles.)

Consider a variable v measured in units $L^x T^y M^z$ where L, T and M are the dimensions of length, time and mass respectively and x, y and z the corresponding exponents. When one of the exponents is unity and the other exponents zero, the unit of v is referred to as a *base unit*. When all exponents are zero the unit is referred to *dimensionless*. In all other cases we speak of *derived units*. Furthermore, vector notation for the units such that $\boldsymbol{u} = [x,y,z]$ is used to denote the vector of exponents. The vector of exponents contains all information necessary to make statements about the units of measurement of variable v.

For example: $\boldsymbol{u} = [1, -2,1]$ defines a derived unit of force, whether it is measured in $\mathrm{kg\,m\,sec^{-2}}$ or in $\mathrm{lbs\,ft\,sec^{-2}}$. Although in this paper mainly SI units of measurement are used, other units e.g. *income per capita* can also be defined.

For notational convenience a smaller system consisting of two physical dimensions is used below (Table 1). This generalizes trivially to an arbitrary number of dimensions, physical or otherwise. The notation $[x,y]$ denotes an expression stated using two dimensions, where x and y are the exponents of these dimensions. It defines constraints in the case of addition and subtraction where the units of the operands need to be the same; in the case of trigonometric, hyperbolic, exponential and many other functions the units of the operands need to be dimensionless. Multiplication, division and the square root function are always defined, but introduce arithmetical manipulations on the types. Finally, the power function a^c is only defined when the second operand is a constant, whose value will influence the output type. The actual value of the expression influences its type.

Table 1 | The type system defined by the physical units of measurement

Operation	Type
Addition/subtraction	$([x, y] \rightarrow [x, y] \rightarrow [x, y])$
Multiplication	$([x, y] \rightarrow [v, w] \rightarrow [x+v, y+w])$
Division	$([x, y] \rightarrow [v, w] \rightarrow [x - v, y - w])$
Square root	$([x, y] \rightarrow [x/2, y/2])$
a^c	$([x, y] \rightarrow [0, 0] \rightarrow [cx, cy])$
Transcendental functions	$([0, 0] \rightarrow [0, 0])$

The term *type system* is used to refer to the combination of type specifications of variables and constants together with the type specifications of the operators. The notation for this type system is borrowed from the typed λ-calculus, in which $(T \rightarrow U \rightarrow V)$ denotes a function that requires two arguments of type T and U and returns a value of type V. The types in the uom system are real valued vectors.

The constraints on the mathematical operators involved in uom problems are specified as follows: each operator can impose constraints on its operands (for instance an equality requirement in the case of addition) or it can specify manipulations in order to produce the output type from the input types as for example in the case of multiplication. Several constraints and manipulations are defined within the uom system as specified in Table 1.

Along with the definition of the independent and dependent variables and possibly typed constants, such a type system defines an uncountably infinite number of types, any real-valued vector of the appropriate size being a data type in its own right. If all variables and constants are dimensionless, the grammar reduces to an untyped grammar. In this case, no manipulations can introduce non-zero exponents.

TYPED GENETIC PROGRAMMING

The original genetic programming system (Koza 1992) does not use data-types or, more accurately, it uses a single data-type. All operations are supposed to be closed (i.e. well-defined) under this data-type. The definition of a *language* in single typed genetic programming is customarily defined through the use of a terminal set T and a function set F. For the language of arithmetics, this could be:

$$\begin{aligned} T &= \{x,y\} \\ F &= \{\mathrm{sqrt}, \mathrm{plus}, \mathrm{times}, \mathrm{minus}, \mathrm{divide}\} \end{aligned} \tag{1}$$

where all functions are of arity 2, except the sqrt function. With such a definition, a set of parse trees is defined where the leaves consist of terminals and the internal nodes of functions. The actual types of the terminals and functions in this specification is omitted. In the typed λ-calculus introduced above, the terminal and function set would be specified through

x: double
y: double
sqrt: (double → double)
plus, times, minus, divide: (double → double → double).

Despite being confined to a single type, genetic programming has been applied to an impressive range of problems (Koza 1992; Babovic 1996; Babovic & Keijzer 2002; Babovic *et al.* 2003). However, despite this success researchers have identified the need for incorporating type information in applications.

Strongly typed genetic programming (Montana 1995) was the first of many approaches that constrain the allowable programs in genetic programming by means of a type system. In the research, the concept of *generic function* (i.e. a function that is defined over all or a well-defined subset of types) was also addressed. Table 1 defines arithmetic for a set of such generic functions over the types in the uom type system.

Strongly typed genetic programming aims to initialize and maintain a population consisting of only correctly typed programs with the goal of optimizing the programs with respect to some objective function. The particular typed genetic programming system used in this paper is described below.

The adaptive logic programming system

In order to implement the type system defined in Table 1, a system inspired by Grammatical Evolution (GE) (O'Neill & Ryan 2001) was used. GE is a developmental genetic programming system, where a string of integers (codons) are maintained that specify choices in a context-free grammar. For the present purpose, GE is enhanced to operate on logic programs. This approach is referred to as *Adaptive Logic Programming* (ALP) (Keijzer *et al.* 2001).

The standard engine working on such *logic programs* is Prolog which employs a depth-first selection of clauses. Executing a logic program in Prolog results in a depth-first enumeration of all possible expressions, in which the order in which the clauses are defined has a significant impact on the results. The ALP system changes the Prolog-specific depth-first behaviour to be guided by a string of choices. Evolving the optimal string of choices becomes the goal of this system.

Similarly to the grammatical evolution system, the ALP system uses a string of integers (codons) to make a choice at each choice-point in the logic program. The system makes choices between clauses belonging to the same predicate and not between the terms in the body; these are executed in order as in Prolog. The system proceeds as follows: at each non-trivial choice point in the derivation process, the number of choices r are recorded and a codon is extracted from the string. This value is then mapped into the interval $[0,r)$ using a *mapping function*, and the resulting choice is executed. In cases in which there are no choices left, the string is empty or a predetermined depth limit is exceeded, the system backtracks and tries a different derivation. (In the case of an empty string, all subsequent choices will fail and the entire mapping process is abandoned. Such an individual is marked as invalid.)

Backtracking is implemented by simply keeping a list of derivations that are tried at each choice point, and limiting subsequent choices to the untried choice points only. This ensures that the system fed with an infinite string of zeroes is equivalent to Prolog. An example of the derivation process is given in Table 2, using the logic program.

Program 1 A logic program defining arithmetic expressions.

```
expr(X)  :-  terminal(X).
expr(X)  :-  mon_op(X, A), expr(A).
expr(X)  :-  bin_op(X, A1, A2), expr(A1),
             expr(A2).
terminal(x).
terminal(y).
mon_op(sqrt(X), X).
bin_op(X + Y, X, Y).
bin_op(X * Y, X, Y).
bin_op(X/Y, X, Y).
bin_op(X - Y, X, Y).
```

The ALP system is defined in such a way that the derivation of a query with an (infinite) string of zeros is equivalent to running the query in Prolog. To make it

Table 2 | An example derivation of an expression using a string of choices to guide the derivation process

	Goals	Bindings	Choice
?-	expr(X)		
?-	$\frac{\text{bin\_op(X, Arg1, Arg2)}}{\text{expr(Arg1), expr(Arg1)}}$		2
?-	expr(Arg1), expr(Arg2)	[Arg1 * Arg2/X]	1
?-	mon_op(Arg1, Arg3), expr(Arg2)		1
?-	expr(Arg3), expr(Arg2)	[sqrt(Arg3)/Arg1]	No choice needed
?-	terminal(Arg3), expr(Arg2)		0
?-	expr(Arg2)	[y/Arg3]	1
?-	$\frac{\text{bin\_op(Arg2, Arg4, Arg5)}}{\text{expr(Arg4), expr(Arg5)}}$		2
?-	expr(Arg4), expr(Arg5)	[Arg4 − Arg5/Arg2]	3
	…	…	…
	$X = \text{sqrt}(y) * (\cdots - \cdots)$		

possible to derive constants, a real-valued array of the same size as the string of codons is maintained. Two special predicates are defined that retrieve integers and reals from this array during the derivation process.

In the present implementation, the crossover is a one-point operator in which the crossover points are chosen independently within the expressed part of the two strings. Two types of mutations are implemented:

1. Point mutation of the codons, where a single point in the codon string is chosen and mutated uniformly to a new codon value in the specified range; and
2. Gaussian mutation of the vector of reals with prespecified standard deviation.

In contrast to tree-based approaches, the crossover and mutation operation used in the ALP system are untyped, i.e. no type information is required to guide the variational operators.

Initialization is performed as a random walk through the grammar. This is followed by a uniqueness check, in order to ensure the non-existence of clones within the initial population. The vector of random numbers is initialized with normally distributed numbers.

Implementation of the uom system in a logic program

The constraints imposed by the uom system can be effectively implemented in a logic program. In order to implement the system a predicate uom/2 is defined. The first argument of the predicate provides the algebraic expression and the second argument the uomns. For addition and subtraction, the program needs to ensure that both arguments are of the same type, i.e.

```
uom(X + Y, UOM):-
uom(X, UOM),
uom(Y, UOM).
```

This simple clause constrains the expression that is induced to be dimensionally correct. The predicate uom can be invoked in four different ways, with each possible combination of the two arguments being instantiated (grounded) or not. The simplest case is where both arguments are grounded. Then the clause simply reduces to a check for dimensional correctness. For example, calling `-? uom(x + y, [1, − 1])` will check both `uom(x, [1, − 1])` and `uom(y, [1, − 1])`. A check like this is completely deterministic, and Prolog would be the optimal choice for checking dimensional correctness. However, with the ALP system we are interested in building an expression that is dimensionally correct, or at the very least dimensionally consistent. The first argument for the `uom` predicate is then never grounded; however, the second argument might be grounded in some cases and not in others.

The addition clause can then be called in two different situations: one in which the units are known and one in which the units are unknown. In the first case, the query can resemble e.g. `uom(Z,[1, − 1])`. Applying this addition clause will bind `Z` to the expression `X + Y` and two more terms are added to the list of goals: `uom(X,[1, − 1])` and `uom(Y,[1, − 1])`. Both the expressions `X` and `Y` are then recursively constrained to be stated in unit `[1, − 1]`.

In the second case, the UOM argument can be ungrounded i.e. a variable. In that case, two different terms are added to the list of goals. If the query was uom(Z, UOM), then Z will be bound to X + Y but the queries added to the list of goals are uom(X,UOM) and uom(Y,UOM). As the derivation of elements on this list of goals is made in a depth-first manner, the first term to be derived will be the term resolving the sub expression X. To be able to derive this expression, it will need to instantiate the UOM variable. Subsequently, the term containing the Y variable will be derived. But at that point in time, the UOM variable will have been grounded by the value obtained by deriving X. The expression that will be induced for the Y variables is then constrained to be stated in the same units as the expression denoted by X. The two elements of the addition are then always constrained to be stated in the same units.

For multiplication and division, two clauses need to be defined: one when the UOM variable is defined (grounded) and another when it is not defined. In this case, the standard Prolog predicate ground is used which checks if there are any free variables in the expression. The following set of clauses implements multiplication (implementation of division is similar):

```
uom(X * Y, UOM) :-
ground(UOM),                % is UOM set to a value?
uom(X, UOMx),               % get the units for the first
                            argument
minus(UOM, UOMx, UOMy),     % calculate the units for
                            the second argument
                            % such that y =
                            output − x
uom(Y, UOMy).               % constrain the expression
                            to be of units UOMy

uom(X * Y, UOM) :-
not(ground(UOM)),           % is UOM unknown?
uom(X, UOMx),
uom(Y, UOMy),               % Do not constrain the
                            units
plus(UOMx, UOMy, UOM).      % Calculate the output
                            units: output = x + y
```

The first clause handles the case where the UOM variable is grounded. It will first derive an arbitrary expression for X, with arbitrary exponents UOMx. From knowing the desired units UOM, and the units of one part of the multiplication UOMx, the units of the second sub-expression can be deduced. This is handled by the call to the minus function, which deterministically calculates UOMy.

In the case where the UOM variable is free, the second clause is used. Here both sub-expressions X and Y can be induced in arbitrary units. Multiplication of two arbitrary sets of units results in adding their exponents. This is handled by the deterministic plus predicate.

In the experiments described below the function sqrt was introduced, defined as:

```
uom(sqrt(X), UOM) :-
ground(UOM),                % is UOM set to a value?
mult(UOM, 2.0, UOMx),       % multply by two
uom(X, UOMx).               % constrain the operand
uom(sqrt(X), UOM) :-
not(ground(UOM)),           % is UOM unknown?
uom(X, UOMx),               % find an operand
                            (unconstrained)
mult(UOMx, 0.5, UOM).       % calculate the output UOM
```

This breaks up the derivation into two seperate cases, similarly with multiplication.

Together with clauses defining the variables and retrieving constant values (which are constrained to dimensionless in order to disallow arbitrary coercions), this logic program implements the uom system in full generality. The ALP system evolves paths through the logic program that result in correctly typed expressions, which are subsequently subject to evaluation of the observations. The procedure above implements the uom system in a form of so-called *declarative bias*, where only those expressions can be derived which are dimensionally correct. The performance of other methods that implement the uom system using *preferential bias* has been analyzed in Keijzer & Babovic (2002).

As a final example, a derivation trace in the program using the following variables and terminals is given:

```
uom(9.81, [1, − 2])  % Earth's gravity acceleration,
                     stated in metres per second
                     squared
uom(d, [1,0])        % A distance measurement,
                     stated in metres
```

Codon values 0 and 1 will be associated with the two terminals, codon values 2 and 3 with the two clauses

for multiplication and codon values 4 and 5 are associated with the two clauses for the square root function.

If the object is to derive an expression stated in velocity units, the query will be `uom(X,[1, -1])`. One possible derivation of this query is:

The selection `prolog` indicates that the result is evaluated deterministically. The expression generated is then: `sqrt(9.81*d)`. In this example, the checks for groundedness always succeeded. When it would fail, the system would backtrack and try another clause.

INTRODUCING DOMAIN-SPECIFIC KNOWLEDGE

The previous section described an approach to introducing high-level information by means of strong typing. In the present case, the approach guarantees that the resulting formulae are dimensionally correct. This, however, cannot be equated to introducing domain-specific knowledge into a data-driven learning machine. Rather, this is an example of strict adherence to a certain high-level concept. The only elements of domain specificity are the units the measurements of the data are performed in.

In the continuation, a brief overview of a case study is given together with existing knowledge about the domain. In subsequent sections, an investigation of injecting both high-level concepts and domain-specific knowledge into the learning algorithm is carried out.

The experiments take the form of a sequence of steps that increasingly constrain the search to use more domain specific knowledge. In the first experiment there is no notion of the use of prior knowledge other than the definition of a certain set of primitive functions that are used. The second experiment injects the units of the measurements into the search process. In the third experiment, the data is pre-processed using the results from a dimension analysis performed by domain experts. Finally, the strongest form of knowledge is introduced by simplifying the search process to finding the elements of an overall functional form. This functional form is also suggested from the literature. The four experiments are analyzed by focusing both on the ability to fit the data and on the interpretability of the results.

Concentration of sediment near the bed

Background

The bottom concentration of suspended sediment is a key parameter within the mechanics of sediment transport. Here the aim is to develop an empirical formulation for the bed concentration c_b, defined at an elevation of a few grain diameters from the bed.

Data

A total number of 10 datasets were utilized in the determination of c_b (Guy *et al.* 1966). The experiments

Selection	Goals	Bindings
	`uom(X, [0,0]).`	
`sqrt`	`ground([1, -1]), mult([1, -1], 2.0, U1),`	
	`uom(X1, U1).`	`[sqrt(X1)/X]`
`prolog`	`mult([1, -1], 2.0, U1), uom(X1, U1).`	`[sqrt(X1)/X]`
`prolog`	`uom(X1, [2, -2])`	`[[2, -2]/U1]`
`times`	`ground([2, -2]), uom(X2, U2),`	
	`minus([2, -2], U2, U3), uom(X3, U3).`	`[X2*X3/X1]`
`prolog`	`uom(X2, U2), minus([2, -2], U2, U3), uom(X3, U3).`	
`9.81`	`minus([2, -2], [1, -2], U3), uom(X3, U3).`	`[[1, -2]/U2]`
`prolog`	`uom(X3, [1,0]).`	`[[1,0]/U3]`
`d`		`[X3/d]`

consisted of a number of alluvial channel tests with the aim of determining the effects of the grain size and of water temperature on thehydraulic and sediment transport variables.

The hydraulic conditions of the individual tests were adjusted by changing the discharge or the slope (or both). The water and sediment were re-circulated until equilibrium conditions were reached. A significant drawback of these datasets is the limited range of water depth covered (0.06–0.41 m). However, the tests comprise a wide range of situations from both the point of view of the hydraulic parameters as well as the bed materials used, the transport rates measured and the bed forms present, making them very attractive for the derivation of an expression for the near-bed concentration in pure current flow.

Table 3 summarizes the quantities used in the problem of determination of concentration of suspended sediment near the bed. As before, it is interesting to observe that only ν, w_s and d_{50} represent 'raw' observations. Shear velocities u_f and u_f' are calculated on the basis of raw observations as:

$$u_f = \sqrt{gDI} \tag{2}$$

and

$$u_f' = \sqrt{gD'I} \tag{3}$$

where I denotes water surface slope and D' denotes boundary thickness layer defined as:

$$\frac{v}{u_f'} = 6 + 2.5 \ln \frac{D'}{k_N} \tag{4}$$

where v is mean flow velocity and k_N is bed roughness $\sim 2.5d$.

Table 3 | Units of measurement of the independent and the dependent variables for the problem of determining the concentration of sediment near the bed

Variable	Description	uom
ν	Kinematic viscosity	$m^2 s^{-1}$
w_s	Settling velocity	$m s^{-1}$
d_{50}	Median grain diameter	m
g	Gravity acceleration	$9.81 m s^{-2}$
u_f	Shear velocity	$m s^{-1}$
u_f'	Shear velocity related to skin friction	$m s^{-1}$
c_b	Concentration of sediment near the bed	–

Human-proposed relationships for near-bed concentration

Generally, the near-bed concentration of suspended sediment c_b depends on: (i) the effective shear stress exerted on the bed by the flow τ'; (ii) the characteristics of the bed material (size d, density ρ_s); and (iii) the characteristics of the fluid (density ρ, kinematic viscosity ν).

Zyserman & Fredsøe (1994) followed an approach initially adopted by Garcia & Parker (1991) for the selection of an expression for c_b, namely

$$c_b = \frac{Ax^n}{1 + \frac{Ax^n}{c_m}} \tag{5}$$

where A, c_m and n are constants and x is a suitable combination of the independent dimensionless parameters. The choice of the functional form of Equation (5) is driven by the fact that c_b becomes zero when x does and c_b converges to the limiting value c_m for high values of x.

The fitting (Zyserman & Fredsøe 1994) yielded values $A = 0.331$, $c_m = 0.46$ and $n = 1.75$, resulting in

$$c_b = \frac{0.331(\theta' - 0.045)^{1.75}}{1 + \frac{0.331}{0.46}(\theta' - 0.045)^{1.75}}. \tag{6}$$

The proposed relationship compares well to values of near-bed concentration obtained from independent datasets. It also provides an improved accuracy over similar expressions and is universally regarded as the formulation describing the concentration of suspended sediment near the bed.

Machine-proposed relationships for near-bed concentration

The most useful knowledge can be condensed to form the dimensionless Shield's parameters θ and θ', as well as utilization of Equation (5) for concentration.

In order to analyze the ability of a GP to produce accurate and interpretable equations, a number of experiments corresponding to various methods of introducing knowledge have been conducted. The same computational setup was chosen for all experiments, summarized in Table 4.

In order to provide an indication of the quality of the solutions that are generated, a short analysis is provided.

Table 4 | The parameters for the experiments

Objective 1	Minimize the normalized RMS
Function set	{+, −, ×, /, sqrt}
	Either with or without type constraints
Population size	250
No. of runs	50
No. of generations	400
Crossover prob.	0.8
Mutation prob.	0.2
Tournament size	2
Strategy	Elitist (250 + 250)
Multi objective strategy	NSGA II (Deb *et al.* 2000)
Size of training set	2/3 of the full set
Size of test set	Remaining 1/3 of the full set

This is an inherently subjective process as it involves the interpretation of the equations and even more subjective reasons such as aesthetic appeal. The expressions are simplified and only the first three significant digits of constants are presented. The presented expressions are selected by taking the best-performing expressions over the entire dataset using both the training and testing sets. These expressions are also inspected with regard to their value in describing the problem itself, with the aim of learning something about the interactions occurring in the processes under study and possibly to guide further data collection campaigns.

The problem considered in this paper is from a highly specialized subfield of hydraulics and such a discussion would fall outside the scope of the present text. The aim of the present analysis is the investigation of methods for introduction of domain knowledge into learning algorithms with emphasis on approaches which enhance interpretability of the resulting equations.

Unconstrained GP: raw values only

The first set of experiments was conducted using standard, unconstrained genetic programming on the basis of raw observations only. Consider the best expression produced in this case, i.e. Equation (7). Although this is one of the more accurate expressions found in this experimentation, it would prove difficult if not impossible to interpret this expression. The sheer size of the formulation makes the exercise almost impossible. In addition, the dimensionally inconsistent fashion in which the variables are combined provides no help in determining the physical interactions for this problem. The following formula is provided as an indication of the sort of expression unconstrained genetic programming can induce:

$$\begin{aligned} c_b \simeq 0.284 & \left((uf_p - w_s)^3 (uf_p - g)\left(g + \frac{uf_p + uf}{uf - g}\right) g^{-5} uf^{-1}\right. \\ & \times \left(g + 13.0\left(ws + g^3 uf_p ws uf^{-1}\left(g + \frac{uf_p}{g}\right)^{-1}\right) uf^{-1}\right. \\ & \times \left(uf_p - 11.3\frac{guf_p}{(uf_p - ws)^2} - g^2\right)^{-1} \\ & \times \left(g + \left(d_{50} + \sqrt{\frac{(uf_p - ws)^2 ws}{g^4}} + g\right)\right. \\ & \left.\left.\left.\left.\times \left(2g + \frac{uf_p}{ws} + uf_p - uf - ws + g^2\right)^{-1}\right)^{-1}\right)^{-1}\right)^{1/2}. \end{aligned} \tag{7}$$

Constrained GP: raw values only

By way of comparison, a constrained genetic programming environment was set up to comprehend all measured data and not the corresponding dimensionless parameters based on the measurements. The purpose for conducting this experiment was to test whether such a set-up is capable of creating a dimensionally correct and still accurate formulation. Since the pre-processing of raw observations (formation of dimensionless θ and θ') was not employed here, it can be argued that GP was confronted with a problem of trying to formulate a solution from first principles. The evolutionary processes resulted in a number of expressions, of which only the most interesting is presented here:

$$c_b = 1.121^{-5} \frac{(u_f' - w_s)\left(1 + 100\frac{u_f' w_s}{g d_{50}}\right)}{u_f + u_f'}. \tag{8}$$

The degree of accuracy of the induced expression is quite satisfactory. The total error over the dataset is reduced and all other statistical measures of accuracy such as

average deviation, coefficient of efficiency, robustness and 95% confidence disclose improvements.

At the same time, the formula is dimensionally correct and it uses the most relevant physical properties in the relevant context. For example, the dimensionless term $u_f' w_s/(gd_{50})$ is effectively a ratio of shear and gravitational forces. Shear forces are represented by u_f', responsible for elevating sediment particles into the stream, while the gravitational term gd_{50}/w_s is 'responsible' for settling the particles. It should be emphasized that this term is rather similar to θ', and that there have been attempts to formulate similar alternative terms for θ (Zyserman & Fredsøe 1994). The remaining group $(u_f' - w_s)/(u_f + u_f')$ is a ratio of resultant energy near the bed and of the total available energy in the flow transporting the particles.

Equation (8) offers a marginal improvement regarding accuracy over the formula induced through standard scientific practice. However, the simple fact that this formula was induced through automatic means based on raw data, and that it provides a competing view on the importance of the processes occurring in this phenomenon, is very exciting indeed. It can also be argued that Equation (8) can be more easily interpreted than the Zyserman–Fredsøe expression (Equation (6)).

Unconstrained GP: dimensionless values only

An alternative road to take is to perform dimensional analysis and transform dimensional variables to dimensionless groups of numbers. By utilizing this approach we avoid problems related to units of measurement, which is a strong reason for the transformation. The section about background knowledge on this sediment transport problem reveals that scientists indeed follow this approach, which is more or less standard scientific practice (Babovic & Keijzer 2000).

All directly measurable quantities (see Table 3) do not correlate as well with the concentration of sediment c_b as the derived dimensionless quantities θ and θ'. For example, the correlation coefficient between c_b and u_f' amounts to 0.784 and between c_b and u_f to 0.628. At the same time, the correlation between c_b and θ' amounts to 0.894 and between c_b and θ to 0.711.

The choice for utilization of Shield's parameters as sole inputs is not only statistically motivated, but it can also be seen as injecting knowledge into the learning algorithm. Since the transformation of the problem removed units of measurement, a standard unconstrained genetic programming environment can be used resulting in:

$$c_b = 0.175\left(\theta' + \left(6.309 + \theta + \left(\theta' - \sqrt{\theta} - \sqrt{\theta'} - \sqrt{\frac{\theta'}{\theta} - \frac{1}{\sqrt{\theta'}} - 1}\right)\right.\right.$$
$$\left.\left.\theta'^2(\theta'\theta - 2\theta')^{-1}\theta^{-2}\right)\left(93.117\frac{1}{\theta'\theta} + 2\frac{\theta}{\theta'} - \theta' + (\theta + \theta'\theta)\right.\right. \quad (9)$$
$$\left.\left.(-\theta' - \theta)\theta^2 + 74.615\theta'^{-1}\right)^{-1}\theta^{-1}\right)\sqrt[4]{\theta'}.$$

Additional 'symbolic gymnastics' could be performed in order to simplify Equation (9), but the main results would still remain. Approaches based on unconstrained genetic programming simply do not take advantage of knowledge provided in the form of pre-processed observations. The resulting models provide a good fit, but are almost impossible to interpret.

The results of experiments appear to support the conclusion that it is more beneficial to introduce the domain knowledge in a form which constrains the search rather than providing elements of knowledge (in our case utilization of θ and θ') which are subsequently manipulated by a learning algorithm in an unconstrained fashion. Related work (Keijzer & Babovic 2002) is entirely devoted to the direct comparison of methods for introducing knowledge in declarative and preferential fashion.

Wrapper

The strongest form of injecting knowledge in the present case would correspond to utilizing Equation (5) and use of constrained genetic programming to evolve the functional form x as well as the constants A, c_m, n. In such a setting information is introduced from three sides: scientific knowledge about the concentration profile c_b is explicitly taken into account; dimensional consistency is assured through the utilization of the constrained genetic programming system; and data are used as a basis for creating the functional form x from Equation (5) as well as for fitting

parameters. The resulting functional form for x is:

$$x = \frac{u_f' - w_s}{u_f} + \frac{u_f' - w_s - \frac{\nu u_f}{u_f'}\sqrt{\frac{g}{u_f'\nu}}}{\sqrt{\frac{u_f'\nu}{d_{50}}}} \tag{10}$$

and the resulting constants are $A = 0.168$, $c_m = 70.28$ and $n = 2.234$. Once these constants and Equation (10) are substituted into Equation (5), the resulting NRMS on training set amounts to 0.448. The constrained nature of genetic programming provides a relatively simple relationship. However, the accuracy of the created model is disappointing (see Table 5). While functional forms such as $(u_f' - w_s)/u_f$ reveal interesting considerations regarding kinetic energy, other parameters such as c_m are well outside the range of physically sensible maximum concentration.

Quantitative results

The comparison between constrained and standard genetic programming is clear: the inclusion of the dimensional constrains does not preclude GP from searching well. More importantly, the constraints seem to have a regularizing effect and implicitly promoteparsimonious solutions.

Closer inspection of statistical measures of accuracy of equations generated by genetic programming reveals a performance which is level with human-generated Equation (6). The worst-performing is Equation (10), which corresponds to the strongest form of introduction of knowledge. The smallest normalized RMS errors are generated by the unconstrained approaches, whereas the best correlation coefficient by the constrained approach on raw data is given by Equation (8). However, it is this last equation that we were able to interpret and obtain additional insights. Equation (8) outperforms the human-generated formulation, and is at the same time rich in meaning. Even although we were able to find unconstrained expressions with a marginally better fit than Equation (8), the increased confidence that it receives by being able to explain its ability to fit the data well makes it the best candidate for use on unseen data. The equation produced using the wrapper approach has, however, such a large error difference that it will probably be rejected regardless of its interpretability value: the 'knowledge' distilled from such a poorly fitting equation is quite likely to be wrong.

Table 5 | Correlation coefficient between observed and calculated concentrations as well as normalised root mean squared error (NRMS, root mean squared error is divided by variance of measured concentrations) provided for the purposes of quantitative comparison of various expressions

Equation	NRMS	Correlation
Human proposed Equation (6)	0.4721	0.7914
Unconstrained GP, raw data Equation (7)	0.4643	0.7899
Constrained GP, raw data Equation (8)	0.4704	0.8058
Unconstrained GP, preprocessed data Equation (9)	0.4597	0.7891
Constrained GP, raw data, wrapper Equation (10)	0.6373	0.6471

At this stage we can try to draw more general conclusions. It appears that the provision of knowledge in a strong form does not necessarily help a data-driven learning algorithm. The presented results seem to favour introduction of knowledge in the form of constraints which define the space of admissible solutions. It is within this space that an algorithm can find an accurate and meaningful functional form. However, it also appears that once the imposed constraints are too narrow, the learning machine cannot generate approximations of a good quality.

DISCUSSION

Genetic programming is an opportunistic search algorithm: it provides expressions that fit the data while satisfying the constraints. Since the only feedback from the problem domain is in the form of the error functions, the algorithm produces expressions that model the relationship in whatever fashion which reduces this error. Although the results presented here are based on preliminary experiments, several main conclusions can be drawn.

Traditionally, dimensionless numbers are used as the dominant vehicle in the interpretation and modelling of experimental values. Such a choice is natural as this conveniently avoids the issues related to dimensional analysis and its correctness. It is also believed that dimensional numbers collapse the search space and that resulting formulations are more compact. This paper demonstrated that it can be advantageous to use data

together with its dimensions. The knowledge-discovery software system uses this information to guide a search for an accurate and physically sound formulation. The result is more accurate than that achieved when a more conventional approach is followed.

As the examples in the previous sections shows, genetic programming can contribute to the creation of novel knowledge. This is done by making use of higher-level elements of knowledge, an intricate part of the process of creating expressions based on data. The obvious corollary is that genetic programming is perhaps best used as an aid to scientists during the discovery process.

The main advantage of using genetic programming in scientific discovery is its ability to generate a large number of different, yet meaningful hypotheses in a very short amount of time. The timescale of human invention runs on the scale of months, if not years. Using a hypothesis generator can considerably accelerate this process, once the scientist is able to interpret these hypotheses.

The interpretation of models should be carried out by domain specialists who can use the background knowledge and their sense of aesthetics to judge which of the proposed hypotheses is the most appropriate formulation. Such a judgment is not offered here; related work (Babovic & Keijzer 1999, 2000; Baptist *et al.* 2007) does attempt to select an appropriate formulation and set up a small theory of worth of the expressions produced by GP. These 'theories' are set up after examining the hypotheses generated by GP, and provide ground for discussion and further experimentation.

CONCLUSIONS

The work is part of a research effort aiming to provide new (and sometimes provocative) hypotheses built from data. The ultimate objective is to build models that can be interpreted by the domain experts. It was shown that relying on *prior* analysis and subsequently embedding a data-driven, black-box model in parts of the problem that are resistant to such prior analysis can easily lead to severe difficulties with interpretation. By including units of measurements in the search, the resulting expressions are however *analyzable*, i.e. the use of units of measurement can help in decomposing the overall expression into composite terms. The general concepts behind the formal system of units can then be used to identify the physical phenomena that are used by these subexpressions to achieve its quality of fit. Demonstrating this amenability to *posterior* analysis of constrained yet data-driven search is the main result of this work.

This amenability to posterior analysis lead us to postulate that the expressions induced in this way transcend the data-driven, black-box approach of model induction. It is argued that the act of interpreting the expression, and accepting or rejecting it on the basis of considerations above and beyond the domain of the measurements, makes the expression function more as a hypothesis on the problem domain than as a set of predictions on data. The use of a data-driven search system providing hypotheses on the problem, as well as predictions on data, makes it possible to use the resulting expressions with more than just statistical confidence. It is only in this way that we can take full advantage of knowledge discovery and advance our understanding of physical processes.

Several expressions have been found that provide new insights into the problem domain. This demonstrates the interpretability value of expressions utilizing elements of knowledge. It is this possibility of directly interpreting the results that distinguishes the approach from other methods. The experiments that increasingly constrain the search using increasingly more elements from prior analysis show that the use of such prior knowledge does not automatically lead to enhanced interpretability. Even worse, the increase in bias can easily lead to excluding the search method of finding well-performing expression.

Finally, we are only beginning to develop effective ways of combining the strengths of human cognition with those of computational discovery systems. However, it is fairly easy to predict a more widespread use of genetic programming in the process of scientific discovery.

ACKNOWLEDGEMENTS

The research presented in this work was carried out as part of Singapore–Delft Water Alliance (SDWA) (R-264-001-001-272). The author gratefully acknowledges support by

research grant 'Data Assimilation and Data-Driven Knowledge Discovery' R-264-000-199-133/112. For more information, visit http://www.sdwa.nus.edu.sg

REFERENCES

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

Babovic, V. & Keijzer, M. 1999 Data to knowledge–the new scientific paradigm. In *Water Industry Systems: Modelling and Optimisation Applications* (ed. in D. Savic & G. Walters), pp. 3–14. Research Studies Press, Exeter.

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

Babovic, V. & Keijzer, M. 2002 Rainfall runoff modelling based on genetic programming. *Nordic Hydrol.* **33** (5), 331–346.

Babovic, V., Harris, E. & Falconer, R. 2003 Velocity predictions in compound channels with vegetated floodplains using genetic programming. *Int. J. River Basin Manage.* **2** (1), 117–125.

Baptist, M., Babovic, V., Rodriguez, J., Keijzer, M., Mynett, A. & Verwey, A. 2007 On inducing equations for vegetation resistance. *J. Hydraulic Res.* **45** (4), 435–450.

Deb, K., Agrawal, S., Pratab, A. & Meyarivan, T. 2000 A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *Proceedings of PPSN-6*, pp. 849–858. Springer-Verlag.

Garcia, M. & Parker, G. 1991 Entrainment of bed sediment into suspension. *J. Hydraulic Eng.* **117** (4), 414–435.

Guy, H. P., Simons, D. B. & Richardson, E. V. 1966 *Summary of alluvial channel data from flume experiments, 1956–61*. Professional Paper 462-I, US Geological Survey, Washington, DC.

Keijzer, M. & Babovic, V. 2002 Declarative and preferential bias in GP-based scientific discovery. *Genet. Programming Evol. Hardw.* **3** (1), 41–79.

Keijzer, M., Babovic, V., Ryan, C., O'Neill, M. & Cattolico, M. 2001 Adaptive logic programming. In *Proceedings of GECCO 2001*, Morgan Kaufmann.

Kompare, B. 1995 *The Use of Artificial Intelligence in Ecological Modelling*. PhD Thesis, University of Ljubljana, Slovenia.

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

Montana, D. J. 1995 Strongly typed genetic programming. *Evol. Comput.* **3** (2), 199–230.

O'Neill, M. & Ryan, C. 2001 Grammatical evolution. *IEEE Trans. Evol. Comput* **5** (4), 349–358.

Zyserman, J. A. & Fredsøe, J. 1994 Data analysis of bed concentration of suspended sediment. *J. Hydraulic Eng.* **120** (9), 1021–1042.

First received 13 May 2008; accepted in revised form 26 January 2009