Check for updates

Genetic programming as a model induction engine

Vladan Babovic and Maarten Keijzer

ABSTRACT

Present day instrumentation networks already provide immense quantities of data, very little of which provides any insights into the basic physical processes that are occurring in the measured medium. This is to say that the data by itself contributes little to the knowledge of such processes. Data mining and knowledge discovery aim to change this situation by providing technologies that will greatly facilitate the mining of data for knowledge. In this new setting the role of a human expert is to provide domain knowledge, interpret models suggested by the computer and devise further experiments that will provide even better data coverage. Clearly, there is an enormous amount of knowledge and understanding of physical processes that should not be just thrown away. Consequently, we strongly believe that the most appropriate way forward is to combine the best of the two approaches: theory-driven, understanding-rich with data-driven discovery process.

This paper describes a particular knowledge discovery algorithm—Genetic Programming (GP). Additionally, an augmented version of GP—dimensionally aware GP—which is arguably more useful in the process of scientific discovery is described in great detail. Finally, the paper concludes with an application of dimensionally aware GP to a problem of induction of an empirical relationship describing the additional resistance to flow induced by flexible vegetation.

Key words | data mining, knowledge discovery, genetic programming

Vladan Babovic
Maarten Keijzer
Danish Hydraulic Institute,
Agern Allé 5,
DK-2970 Hørsholm,
Denmark

INTRODUCTION

In making the most of a set of experimental data it is generally desirable to express the relation between the variables in the form of an equation. In view of the necessarily 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 recognised chemical, physical and biological laws have started out as empirical equations.

Sciences devote particular attention to the development of a physical symbol system, such as a scheme of notation in mathematics, together with the evolution of more refined representations of physical and conceptual processes in the form of equations in the corresponding symbols. It is a common experience that one and the same physical symbol system may serve for the expression of a great number of different equations. Since each equation can be regarded as a collection of signs that serves as a sign for a particular physical object, process or event, so it constitutes a *model* of that object, process or event (Abbott 1993). Data, on the other hand, remain as ‘mere’ data just to the extent that they remain a collection of signs that does not serve as a sign. From this point of view, the evolution of an equation within a physical symbol system as a means of better conveying the ‘meaning’ or ‘semantic content’ that is encapsulated in the data, corresponds to the evolution of another kind of sign and thereby constitutes a model. Evidently the ‘information content’ is very little changed, or even unchanged, but the ‘meaning value’ is commonly increased immensely. Since it is just this increase in ‘meaning value’ that justifies the activity of substituting equations for data, there is a natural interest in processes for further promoting such means.

The formation of modern sciences occurred approximately in the period between the late fifteenth century and the late eighteenth century. The new foundations were based on the utilisation of the concept of a physical

Downloaded from http://iwaponline.com/jh/article-pdf/2/1/35/392039/35.pdf
by guest

experiment and the applications of a mathematical apparatus in order to describe these experiments. The works of Brahe, Kepler, Newton, Leibniz, Euler and Lagrange clearly exemplify such an approach. Before these developments, scientific work primarily consisted only of collecting the observables, or recording the 'readings of the book of nature itself'.

This novel scientific approach was principally characterised by two stages: a first one in which a set of observations of the physical system are collected, and a second one in which an inductive assertion about the behaviour of the system – a hypothesis – is generated. Observations represent specific knowledge, whereas a hypothesis represents a generalisation of these data that implies and describes observations. One may argue that through this process of hypothesis generation one fundamentally economises thought, as more compact ways of describing observations are proposed.

Today, in the late 20th century, we are experiencing yet another change in the scientific process as just outlined. This latest scientific approach is one in which information technology is employed to assist the human analyst in the process of hypothesis generation. This computer-assisted analysis of large, multidimensional data sets is sometimes referred to as a process of 'data mining and knowledge discovery' (Fayyad *et al.* 1996). Data mining and knowledge discovery aims at providing tools to facilitate the conversion of data into a number of forms that convey a better understanding of the processes that generated or produced these data. These new models, combined with the already available understanding of the physical processes—the theory, can result in an improved understanding and novel formulations of physical laws and an improved predictive capability.

As we enter the true digital information era, one of the greatest challenges facing organisations and individuals is how to turn their rapidly expanding data stores into accessible, and actionable, knowledge (Fayyad *et al.* 1996). Means for data collection and distribution have never been so advanced as they are today. While advances in data storage and retrieval continue at an extraordinary rate, the same cannot be asserted about advances in information and knowledge extraction from data. Without such developments, however, we risk missing most of what the data have to offer. Only a very small percentage of the captured data is ever converted to actionable knowledge. Owing to the data – and information – overflow, the traditional approach of a human analyst, intimately familiar with a data set, serving as a conduit between raw data and synthesised knowledge by producing useful analyses and reports, is breaking down.

What to do with all this data? Ignoring whatever we cannot analyse would be wasteful and unwise. This is particularly pronounced in scientific endeavours, where data represent carefully collected observations about particular phenomena that are under study.

However, mining the data *alone* is not the entire story. At least not in scientific domains! Scientific theories encourage the acquisition of new data and these data in turn lead to the generation of new theories. Traditionally, the emphasis is on a theory, which demands that appropriate data be obtained through observation or experiment. In such an approach, the discovery process is what we may refer to as *theory-driven*. Especially when a theory is expressed in mathematical form, theory-driven discovery may make extensive use of strong methods associated with mathematics and with the subject matter of the theory itself. The converse view takes a body of data as its starting point and searches for a set of generalisations, or a theory, to describe the data parsimoniously or even to explain it. Usually such a theory takes the form of a precise mathematical statement of the relations existing among the data. This is the *data-driven* discovery process.

Most of the applications of data mining technology are currently in the financial sector. There is a very strong economic incentive to apply state-of-the-art technology for commercial benefits. Additionally, this domain is, relatively speaking, theory-poor, and the generation of new 'black-box' tools based solely on observations is accepted with little scepticism. In scientific applications, the situation is quite different. Clearly, there is an enormous amount of knowledge and understanding of physical processes that should not just be thrown away. We strongly believe that the most appropriate way forward is to combine the best of the two approaches: theory-driven, understanding-rich with data-driven discovery processes.

Downloaded from http://iwaponline.com/jh/article-pdf/2/1/35/392039/35.pdf
by guest

SYMBOLIC REGRESSION

Regression – linear or nonlinear – plays a central role in the process of finding empirical equations. In its most general form, regression techniques proceed by selecting a particular model structure and then estimating the accompanied coefficients based on the available data. The model structure can be linear, polynomial, hyperbolic, logarithmic, etc. The only requirement in such an approach is that the coefficients in the model can be estimated using an optimisation technique. In generalised linear regression for instance, the only requirement is that the model is *linear in the coefficients*. The model itself can consist of any functional form. Another technique may be a *nonlinear regression* where the only requirement is that the model is differentiable both in the inputs and the coefficients. Supervised artificial neural networks belong to this class of regression techniques.

In this paper a relatively novel regression technique, symbolic regression (Koza 1992) is described. The specific model structure is not chosen in advance, but is part of the search process. In this algorithm, both model structure and coefficients are simultaneously being searched for. The user has to define some basic *building blocks* (function and variables to be used); the algorithm tries to build a model using only those specified blocks. As the space of model structures is in general not smooth, not differentiable nor linear in any useful sense (it is in fact highly discontinuous), standard optimisation techniques fail when trying to find both the model structure and the coefficients.

The only information available for the symbolic regression problem is the error that a particular model makes. No auxiliary information about gradients is available. The class of evolutionary algorithms, therefore, seems to be one of the few methods able to perform an effective search in this domain.

The evolutionary paradigm

The paradigm of evolutionary processes, as established by Darwin and Weismann in the 19th century, and provided with its information-theoretic interpretation by Crick, Watson and others in the 1950s, distinguishes between an organism's *genotype*, which is constructed of genetic material that is inherited from its parent or parents, and the organism's *phenotype*, which is the coming to full physical presence of the organism in a certain given environment and is represented by a body and its associated collection of characteristics or phenotypic traits. Within this paradigm, there are three main *criteria* for an evolutionary process to occur (Maynard Smith 1975):

- **Criterion of Heredity:** Offspring are similar to their parents: the genotype copying process maintains a high fidelity.
- **Criterion of Variability:** Offspring are not exactly the same as their parents: the genotype copying process is not perfect.
- **Criterion of Fecundity:** Variants leave different numbers of offspring: specific variations have an effect on behaviour and behaviour has an effect on reproductive success.

It can be shown that these three requirements provide necessary and sufficient conditions for an evolutionary process to occur, so that they define the grammar of the corresponding 'language', whether this be written in strings of nucleotides, or amino acid molecules, or in strings of binary digits, or in strings of symbols, or whatever. The criterion of heredity ensures that the offspring inherits information from its parents, assuring similarity. Variability is guaranteed in any entropy-producing system, whereas the criterion of fecundity provides, on average, more fit individuals with the possibility to reproduce more often and thus generate more and better-surviving offspring.

The application of these evolutionary principles results in an adaptation of a population to an environment. Adaptation can in turn be conceived as a process of accumulation of knowledge (see, for example, Margalef 1968). Since Darwinism is a theory of processes of cumulative adaptation, it can best be conceived within the present perspective as a *theory of accumulation of knowledge* about an environment.

The first proposal to apply the creative power of the evolutionary process in more general terms than the

Downloaded from http://iwaponline.com/jh/article-pdf/2/1/35/392039/35.pdf
by guest

Table 1 | Pseudo-code for evolutionary algorithm.

```
Function evolutionary_algorithm
    $P_0$=initialize()
    $P_{t+1}$=variate(select(evaluate($P_t$)))
End function
```

where
- `initialize` — produces the initial random population
- `evaluate` — performs the genotype to phenotype mapping and evaluates the phenotype
- `select` — performs selection based on observed performance
- `variate` — performs reproduction, mutation, crossover or even multiparent recombination

biological can be traced back at least as far as the work of Cannon (1932: see Harvey 1993). The first working computer algorithm that realised this approach – that of *Evolutionary Operation* (EVOP) – is attributed to Box (1957). Works of Friedberg (1958) and Bremermann (1958) provided some inspiring results but were not well accepted by the contemporary scientific community (e.g. Minsky 1963).

In time, however, research in evolutionary algorithms overcame most of the problems that both Friedberg and Bremermann had encountered. More than three decades of research have resulted in differentiation into four main streams of *Evolutionary Algorithm* (EA) development: namely those of *Evolution Strategies* (ES) (Schwefel 1981); *Evolutionary Programming* (EP) (Fogel *et al.* 1966); *Genetic Algorithms* (GA) (Holland 1975); and *Genetic Programming* (GP) (Koza 1992). However, all evolutionary algorithms share the common property of applying evolutionary processes in the form of selection, mutation and reproduction on a population of individual structures that undergo evolution. The general process is illustrated in the form of a pseudo-code in Table 1. The criterion of heredity is assured through the application of a crossover operator, whereas the criterion of variability is maintained through the application of a mutation operator. A selection mechanism then 'favours' the more fit entities so that they reproduce more often, providing the fecundity requirement necessary for an evolutionary process to proceed.

The structures undergoing adaptation

Owing to the many different types of evolutionary algorithm defined, it is difficult to develop a formal framework for describing evolving genotypes. For linear genotypes such as those used in most genetic algorithms and evolution strategies, Radcliffe & Surry (1994) showed that an individual's genotype ω can be coded as a string of l genes.

Each of these genes can take on values from some (typically, but theoretically necessarily not finite) set A_i. Accordingly, the genotypic representation space Ω takes the form:

$$\Omega = A_1 \times A_2 \times A_3 \ldots \times A_l . \tag{1}$$

In classical *Genetic Algorithms* (GAs), as introduced by Holland (1975), the elements a_{ki} of the sets A_i typically take on binary values:

$$\forall\, a_{ki} \in A_i, \quad a_{ki} \in \{0,1\} . \tag{2}$$

In *Evolution Strategies* (ES), on the other hand, the elements a_{ki} are real-valued numbers, i.e.

$$\forall\, a_{ki} \in A_i, \quad a_{ki} \in \Re . \tag{3}$$

In these cases we can regard the set of distinct elements a_{ki} as an *alphabet*. In *Evolutionary Programming* (EP) the sets A_i are more broadly defined and can be adapted to any problem at hand, ranging from numerical optimisation (Fogel 1992), through finite-state automation evolution (Fogel 1993) to connectionist network induction (Fogel *et al.* 1990).

The phenotype ξ of an evolving entity is an *interpretation* of a genotype ω in a problem domain. In the case of GAs, a typical genotype ω composed of l 'letters' or 'words' may appear in a more general form as $\omega\ (a_1 \times \ldots \times a_1) \in \Omega$. For example, when a particular a_i takes values of 0 or 1, then the genotype simply takes the form of a binary string, such as:

$$\omega = 1\ 0\ 0\ 1\ 1\ 0\ 1\ 1. \tag{4}$$

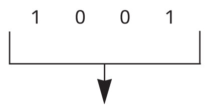
To extract any useful meaning, this code has to be interpreted in some way, and indeed in this case the

Downloaded from http://iwaponline.com/jh/article-pdf/2/1/35/392039/35.pdf
by guest

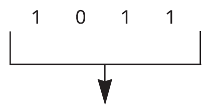
interpretation *is* the individual's phenotype ξ. The mapping between genotypes and phenotypes then depends only on the admitted physical symbol system—and the developers' ingenuity. For example, if a binary string is to be interpreted as a phenotype that is characterised by two traits that have numerical measures x_1 and x_2 only, defined on domains [$\check{x}_1$, $\hat{x}_1$] and [$\check{x}_2$, $\hat{x}_2$] respectively, a simple mapping can always be found that will provide this interpretation. For example, in the case of Equation (4), where $l = 8$, we can subdivide the genotype into, say, two equinumerous intervals of length $L_x = l/2 = 4$, and apply the following general mapping:

$$x_i = u_i + \frac{\hat{x}_i - \check{x}_i}{b^{L_x} - 1}\left(\sum_{j=1}^{L_x} a_j b^{j-1}\right), \tag{5}$$

where a_j is the value of the numeral in the binary string and b is the base number. In this way a genotype ω is interpreted as a phenotype ξ with just two traits, in this case of the form x_1 [0, 10] and x_2 [0, 10], according to the scheme exemplified in Equation (5) as follows:



$$x_1 = 0 + \frac{10-0}{2^6-1}\,(1.2^0 + 0.2^1 + 0.2^2 + 1.2^3)$$
$$x_1 = 6.0$$



$$x_2 = 0 + \frac{10-0}{2^4-1}\,(1.2^0 + 0.2^1 + 1.2^2 + 1.2^3)$$
$$x_2 = 8.67$$

Thus the genotype of Equation (4) generates a phenotype with two traits (6.0, 8.67), each of which is characterised by one real number.

Evolution Strategies (ES), on the other hand, represent evolving entities using real-valued numbers and not binary strings. In this way the mapping between genotype and phenotype is avoided. This provides certain advantages and accelerates the algorithms, but it naturally restricts the field of application to numerical domains only. One of the main advantages of the GA is its rudimentary alphabet, consisting only of 0s and 1s or other such sign pairs. Moreover, in the case of a GA there is in principle nothing preventing the application of a mapping in which, for example, 100 would stand for x_1, 11 for + and 011 for x_2, so that the binary representation of the genotype as exemplified in Equation (4) above would become a phenotype with an interpretation (x_1 + x_2).

Clearly, phenotypes generated applying this last mapping and a mapping of the kind exemplified in Equation (5) are meant for different purposes. The latter are typically used for optimisation and constraint satisfaction applications. For a general overview see Goldberg (1989) and, for applications in a water-related domain, Babovic (1993, 1996).

A third route of genotype to phenotype mapping is present in genetic programming (GP) (Koza 1992). Instead of using the identity mapping as in evolution strategies or keeping it completely open such as in genetic algorithms, genetic programming restricts the mapping from genotype to phenotype such that the phenotype is executable in the same sense as a computer program is executable. Problems tackled by genetic programming are then problems of *program induction*. Although this limits the scope of genetic programming, it remains broad enough to encompass seemingly remote problems such as: model induction, symbolic regression, optimal control, planning, sequence induction, discovering game-playing strategies, empirical discovery and forecasting, symbolic integration and differentiation, inverse problems and induction of decision trees. In his monograph, Koza (1992) argues (chapter 2, p. 9):

A wide variety of terms are used in various fields to describe this basic idea of program induction. Depending on the terminology of the particular field involved, the computer program may be called a formula, a plan, a control strategy, a computational procedure, a model, a decision tree, a game-playing strategy, a robotic action plan, a transfer function, a mathematical expression, a sequence of operations, or perhaps merely a composition of functions.

Similarly, the inputs to the computer program may be called sensor values, state variables, independent variables, attributes, information to be processed, input signals, input values, known variables, or perhaps merely arguments to a function.

The output from the computer program may be called a dependent variable, a control variable, a category, a decision,

Downloaded from http://iwaponline.com/jh/article-pdf/2/1/35/392039/35.pdf
by guest

an action, a move, an effector, a result, an output signal, an output value, a class, an unknown variable, or perhaps merely the value returned by a function.

Regardless of the differences in terminology, the problem of discovering a computer program that produces some desired output when presented with particular inputs is the problem of program induction.

The alphabet used in the GP code comprises a physical symbol system, so that it consists of symbolic operators, the choice of which depends only on the nature of the problem to be solved. One may, for example, choose an alphabet consisting solely of logical operators, such as AND, OR, NOT and IF THEN ELSE and evolve rule-based-like structures. One may restrict the alphabet to arithmetic–algebraic operators such as +, −, *, /, etc. so as to evolve arithmetic–algebraic formulae. On the other side of the symbolic spectrum, the alphabet can be chosen from operators containing side effects, memory addressing and loops such as READ, WRITE, WHILE which, together with arithmetic and logical functions, embodies a complete computer language.

As computer programs (and less generally mathematical formulae) have a well defined grammar and are of variable size and shape, in genetic programming the structures undergoing adaptation are often taken from the class of parse trees. Parse trees are general constructs in computer science and are used, among other things, as intermediate structures in compiling computer languages such as C(++), Pascal and Fortran. A parse tree is inductively defined in Table 2: In this definition the subscripts 0, 1, . . . , n of the sets F in L denote the *arity* of the functions i.e. the number of arguments they need. The set of functions of arity zero is often called the terminal set or T. When we restrict the function and terminal sets to mathematical operations, the parse trees that can be generated, and subsequently the search-space for genetic programming, will consist of *all* possible mathematical expressions, regardless of their shape and size.

Table 2 | Definition of parse tree.

Given sets of functions from the language $L = \{F_0, F_1, \ldots, F_n\}$ then:
• $p \in F_0$ is a parse tree,
• if $p_1, \ldots, p_m$ are parse trees and $p \in F_m$, then $p(p_1, \ldots, p_m)$ is a parse tree.
($p_1, \ldots, p_m$ will then be called the children of p.)

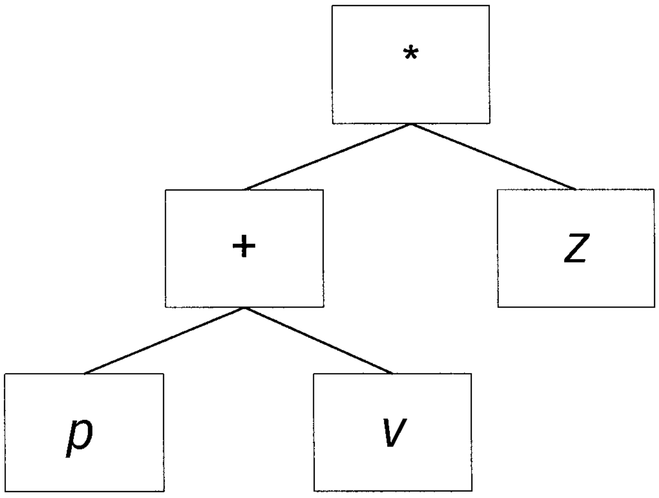


Figure 1 | A parse tree representing the expression $(p+v)*z$.

The great advantage of this change in representation will become clear when we represent these structures in tree form rather than bracketed strings. The distinction between the *terminals* and *functions* (non-terminals) will then become clear. Terminals are represented by the extremities or 'leaves' of the tree, like p, v or z. In addition to these, terminals may be constants that are inserted in the formulae. Functions are elements of a tree that act upon terminals and in Figure 1 these are exemplified by + and *.

It follows straightforwardly from the definition in Table 2 that replacing a child node of a well-defined parse tree by an arbitrary well-defined parse tree yields a well-defined parse tree. It follows that:

1. Every grammatically correct equation (and, in general, every *well-formed formula*, or *wff*) can be represented as a parse tree.
2. Every planar graph with grammatically correct terminals and functions represents a grammatically correct equation (or, generally, a *wff*).
3. Depth and size of a *wff* are easily defined as the longest non-backtracking path from a leaf to the root of the tree, and the number of nodes in the tree, respectively.

Downloaded from http://iwaponline.com/jh/article-pdf/2/1/35/392039/35.pdf
by guest

When applying genetic programming using mathematical functions and variables only, we have entered the field Koza termed *symbolic regression*. Symbolic regression, in contrast with (non-)linear regression tries to find both the functional form as well as the coefficients of a formula. Just like linear regression it works on a spreadsheet of data, and tries to fit a model to this data. There are two unique features in symbolic regression on data. First, the form of the model does not need to be specified beforehand. A specification of the more elementary building blocks (the language L above) will do. Second, the optimisation criterion is not restricted to a class of, for example, squared error measures. As will be shown below, the optimisation does not even need to be restricted to a single objective: a multi-objective optimisation is quite straightforward to implement.

The equations so represented may be grammatically correct, *but they may not necessarily be logically correct*, *or 'logical'*, and then in the special but immediate sense that they may not necessarily be *dimensionally consistent*.

Symbolic regression on Bernoulli's law

In the remainder of this section an introduction to genetic programming specifically used in symbolic regression will be offered. For pedagogic purposes, an artificial dataset with a known solution is chosen as a running example. The target is to find Bernoulli's law of energy conservation:

$$E = z_1 + \frac{p_1}{\gamma} + \frac{v_1^2}{2g} = z_2 + \frac{p_2}{\gamma} + \frac{v_2^2}{2g} = \text{constant} \qquad (6)$$

or in a regression context, to find:

$$E = z + \frac{p}{\gamma} + \frac{v^2}{2g}, \qquad (7)$$

where:

z denotes distance above a certain energy datum,
p denotes pressure,
v denotes velocity,
g denotes the Earth's gravitational acceleration [$g = 9.81\ \mathrm{m^2/s}$], and
γ denotes the specific gravity of a fluid [for water $\gamma = 9810\ \mathrm{N/m^3}$].

Table 3 | Dataset for the induction of Bernoulli's law.

	z	v	P	E
1	7.945	3.91	14121.67	10.94294
2	18.585	5.287	14506.06	22.91308
	...	...	...	...
1000	15.884	12.7	10770.01	33.42325

Formula (6) is a simplified formulation that ignores energy losses due to friction and local losses induced by sudden changes in a cross-section of the conduit. In accordance with tradition in hydraulics, energy in formulation (6) is expressed in implicit potential energy terms, as metres of a water column, rather than in more conventional energy units.

For the purpose of induction of Bernoulli's law a specific genetic programming system will be developed in this text. This system differs in almost all aspects from a more 'standard' implementation of a genetic programming system. In particular, it is almost entirely different from the system as described by Koza (1992). There is a great degree of freedom in implementing genetic programming systems and there are no known optimal choices for the various details of the system. However, the overall design of a genetic programming system needs to address a number of issues and these are condensed in this paper.

The data and the terminal set

First a dataset is created on the basis of the desired relation $E = z + p/\gamma + v^2/2g$ in numeric form using reasonable numeric values. See Table 3 for an example.

In real-world problems a similar dataset would be all that is available. The object of the search is now a formula that takes z, v, *and* p as input and produces E as output. The constants g and γ may also be included, but in the present example only randomly generated constants will be used.

The function set: sufficiency and closure

The next step is to define a function set for the problem. A function and terminal set are called *sufficient* if they are able to represent a solution to the problem. As in this case the optimal solution is known, the optimal function set can be determined easily. In the general case, when all that is known is the raw data, an informed guess must be made as to what functions and terminals to use for genetic programming. For this problem we choose:

Function Set $\{+, *, -, /, power, sin, ln\}$

There is one point that needs to be explicitly taken care of. As the crossover mechanism (as defined below) can swap any sub-tree to any location in the tree, it needs to be ensured that all values have the same type, and that all functions can accept all possible inputs from terminals and functions and will return a well-defined value. This property is called *closure*. In the function set above there are three functions that might violate this closure property. The *division* function is undefined when its second argument is zero, the *power* function is undefined when a negative value is raised to a non-integer power, and the *natural logarithm* is undefined for values smaller and equal to zero. Other potential pitfalls are underflows and overflows in the floating-point implementation of these functions.

One possible approach to satisfying closure is to protect the operators so that they return well-defined values when confronted with illegal inputs (Koza 1992). A protection for division might be to return the value 1 when confronted with a division by zero. For the power function a solution could be to use complex numbers instead. Similarly, one can also define protections for the square root and logarithm functions. Another approach (which is adopted here) is to use a special undefined value *NaN* that is returned by functions with illegal inputs. All functions that receive *NaN* in any of their inputs will also return *NaN*. Finally, the objective function will return the worst possible value when one of the inputs is *NaN*.

Initialisation of genetic programming

Evolutionary algorithms are typically initialised by creating an initial population randomly. This is in fact a

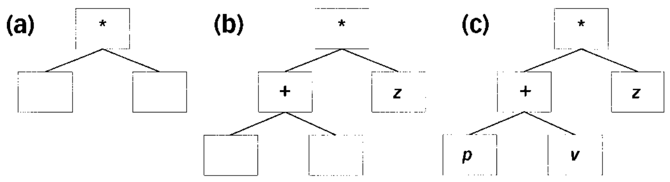


Figure 2 | Example of the systematic specialisation of an expression.

purposeful action, in that a randomly created collection of entities can provide a satisfactory initial *coverage* of the search space. At the same time, computationally speaking, this is an inexpensive process that calls for no initial or preconceived knowledge from the side of the problem-solver.

The process of random initialisation of a parse tree is depicted in Figure 2. First, a function is randomly selected from a function set, e.g. *. This function has an arity of 2 and thus requires two arguments. In the next step, arguments are randomly selected from the function sets. Figure 2a,b illustrates the situation in which the left branch of a tree is fed with a function + and the right with a terminal *z*. Since the left branch is non-terminal the process is repeated recursively until all the leaves of the tree are terminals. This is illustrated in Figure 2c, where the randomly selected terminals for + are *p* and *v*.

This process is repeated for every individual in a GP population of size *N*. It should be noted that such a creation of parse trees results in the generation of genotypes of variable length. To restrict the size of the initial population, a maximum depth parameter *D* may often be supplied. Several different initialisation procedures exist for GP; some of them are described below.

Full method

Assign zero probability to the terminal set *T* and uniform probability to the remaining functions from *L*. Generate a tree consisting of non-terminals only until the depth *D* is reached. Then assign uniform probability to the members of *T* and zero to the remaining functions from *L*, thus completing the tree with terminals.

Grow method

Assign uniform probability to all functions from *L*. If the

Downloaded from http://iwaponline.com/jh/article-pdf/2/1/35/392039/35.pdf
by guest

depth of the tree reaches D generate terminals only. This will result in variable shaped and sized trees.

Ramped half-and-half

A combination of the full and grow methods. Generate for each of the depths from 2 to D an equal number of trees using grow and full methods in equal proportions. This method is preferred by Koza (1992) as it generates parse trees of various shapes and sizes and consequently provides a good coverage of the search space. This initialisation procedure is less sensitive to the ratio of functions over terminals than the grow method by itself.

Exact uniform initialisation

None of the methods above succeeds in generating parse trees distributed uniformly over the space of all possible parse trees up to depth D. As there are exponentially more parse trees of depth D than of depth $D-1$, the ramped half-and-half method is biased towards generating trees of smaller depths. (The exact relation between the number of possible parse trees over increasing depth is given by $n(D+1)=\sum_{i=1}^{|L|} \mid F_i \mid \sum_{j=1}^{D} n(j)^i - \sum_{j=2}^{D} n(j)$, where $n(D)$ is the number of parse trees at a certain depth and $n(1)$ is the size of the terminal set F_0 or T. The recursion can be verified by considering that a tree of depth $D+1$ has children that are maximally D deep, given by the first summation. As the first summation calculates the number of parse trees up to depth $D+1$ including all trees from depth 2 to D, subtracting the second summation will give the exact result of all trees of a specific depth.) Bohm & Geyer-Schultz (1996) suggest using the recursion above to generate trees uniformly over all depths and shapes, such that every parse tree up to depth D has an equal probability of being generated.

Initialisation on size

Instead of using a maximum initial depth parameter D, use an initial size parameter S, and initialise using the pseudo-code from Table 4.

This implements a full method on size, as every tree will be of exact size S. To create a grow variant of this algorithm it is only necessary to include the terminal set in the sets of functions in line 4 (marked with *) of the pseudo-code above. This implementation method gives better control of the size of the initial parse trees.

Table 4 | Pseudo-code for initialization on size.

```
function tree = initialize_tree($S$)

   if $S$ = 1
      return random terminal from $F_0$
   else
      $f$ = select random function from $F_1$, …, $F_S$ (*)

      select sizes $S_1$, …, $S_i$ where $i$ is the arity of $f$,
         subject to $S_1$ + … + $S_i$ + 1 = $S$

      for $j$ = 1 to $i$
         $f$.child($j$) = initialize_tree($S_i$)
      end for

      return $f$
   end if
end function
```

In our running example of induction of Bernoulli's law, a variant of the grow initialisation on size is employed with $S=15$, $N=4$ and $D=4$, and provides, for example, the following kinds of expressions:

$(p+v)^*z$	(A)
p^2	(B)
$\ln(v)/(z+\sqrt{p})$	(C)
$4^*v+\sin p$	(D)

When a constant is created its value is set to the ratio of two independent uniform random variables.

Objective functions, selection and fitness

Although it is often claimed that evolutionary algorithms embody a computer-based version of natural selection (Koza 1992), this is not the case when they are used in an optimisation context. In contrast with optimisation, evolution is not goal-oriented in an optimisation sense. The only goal orientation that might be distinguished is that organisms are optimised in their ability to create offspring. All traits in all their wonderful complexity are mere side-effects of this all-encompassing goal.

To continue drawing analogies with biology, evolutionary algorithms as used for optimisation purposes

Downloaded from http://iwaponline.com/jh/article-pdf/2/1/35/392039/35.pdf
by guest

are more akin to breeding. If one wants to optimise some trait in, for example, pea plants, one will proceed by breeding with those plants that exhibit most of the specific trait to breed. These plants will be crossbred and grown again. By applying this procedure for a number of generations, the average quality of a trait of interest in the pea plants will increase. Contemporary agriculture is extensively optimised using this technique.

Evolutionary algorithms in optimisation proceed in a similar fashion. First an *objective function* is defined that usually takes the form of a scalar value that is applied to every population member:

$$o{:}F_x \rightarrow \Re \qquad (8)$$

mapping the outputs of the formula F_x to a real-valued scalar. When using genetic programming to find equations that approximate data (i.e. symbolic regression), the objective functions used are often error measures of some form. This measure can simply be a root mean squared error, but also the number of points correctly predicted within some accuracy interval of tolerance (the number of correctly predicted data points is often called the number of *hits* of the population member). Since a population member within a genetic programming computes some function f using the input vectors $\boldsymbol{x}$, the error in producing the desired outputs y can be computed with any cost or error function. To compare objective function values, it is often enough to supply information a to whether this function should be minimised or maximised. Again, it is not necessary that the objective function is continuous nor is it necessary for it to be differentiable, as an evolutionary algorithm does not use this information.

The objective function chosen in the running example of Bernoulli's law induction is the root mean squared error (RMS), given by:

$$RMS = \left(\frac{1}{N} \sum_{i=1}^{N} (x_i - y_i)^2 \right)^{\frac{1}{2}} . \qquad (9)$$

After calculating the objective function values for all population members, the following step is to create the next generation. Similarly, as in breeding, the better solutions will have a larger expected number of offspring. This step is referred to as *selection*.

Selection is the optimisation force within the evolutionary algorithm, keeping it focused on the worthwhile regions of the search-space by mapping the objective function values to the number of offspring produced. Selection generally reduces the population to its observed best members; the population is subsequently enlarged by the mutation and crossover operations on the selected members. Several selection mechanisms have been proposed for evolutionary algorithms. Three of the most popular mechanisms are described here.

1. **Truncation Selection** simply keeps the best proportion of t of the population and discards the rest. The parameter t then governs the *selection intensity* of the mechanism. Evolution strategies (Schwefel 1981) use only this mechanism.
2. ***k*-Tournament Selection** proceeds by holding tournaments of k randomly selected individuals and selects the best as the winner. In a *steady-state* algorithm a subsequent tournament is held for finding a partner and an inverse tournament – selecting the worst individual – can be held to find a slot to fit in one of the offspring. The tournament size parameter k governs the selection intensity here.
3. **Fitness Proportional Selection** assigns every individual in the population a probability proportional to its objective function value. Selection is then governed by these probabilities. To obtain these probabilities the objective function needs first to be scaled to a non-negative quantity. This transformation from raw objective scores to a non-negative quantity is called the *fitness function* (Grefenstette 1997).

Although fitness proportional selection is one of the most frequently used selection mechanisms in genetic algorithms (and in fact the *schema theorem* (Holland 1975) depends on it), it is also one of the most cumbersome to use. The objective function values need to be transformed and scaled. To vary the selection intensity, several scaling mechanisms have been devised, often utilising some properties of the particular objective function used. In effect, such scaling mechanisms implement the entire selection mechanism, as fitness proportional selection by its very nature is a method that 'merely' translates values into probabilities. Likewise all other selection mechanisms can be translated as scaling mechanisms for fitness proportional selection through a straightforward, although tedious, calculation of probabilities. (A cautionary remark should be issued that owing to the relatively small sizes of evolutionary algorithm populations, the effect of sampling error is non-negligible.)

In his first treatise on GP, Koza (1992) used only fitness proportional selection, and subsequently intermediate scaling methods such as standardised, adjusted and finally normalised 'fitness'. The subsequent volume (Koza 1994) and most of the current implementations of genetic programming use tournament selection on a steady-state population. This selection mechanism has the advantage (together with truncation selection) that the objective function values are only compared and the mapping from objective function values to expected number of offspring is then *implicitly* (albeit with sampling error) performed by the selection mechanism.

In our Bernoulli example truncation selection with a truncation percentage of 15% has been chosen. This is applied on a population size of 500. Truncation selection has been chosen for its ease in implementation. The population is sorted using the RMS error, the worst 85% of the population are deleted, and the remaining part of the population is filled with variants of those 15% survivors. This implements rather high selection intensity.

In this section symbolic regression was treated using a single objective (an error measure). Evolutionary algorithms are, by virtue of their population-based search, well suited for a multi-objective approach. With a number of objectives to optimise on, evolutionary algorithms can develop the entire Pareto-front of non-dominated solutions by employing fairly simple mechanisms of dominance. This is particularly useful when no *a priori* preference or weighting scheme on the objectives is possible. The generalised objective function in multi-objective optimisation takes the form:

$$o{:}F_x \rightarrow \Re^n, \qquad (10)$$

where n is the number of objectives. One way of implementing a multi-objective mechanism is by a means of a so-called Pareto ranking, where the concept of dominance plays an important role. When working with more than one objective, a solution is said to be dominant over another solution of the problem when it is better on at least one objective and not worse on any of the others. The Pareto ranking method assigns each population member a rank based on the number of members that it dominates (Foseca & Fleming 1995). In this fashion the solutions at the front of non-dominated solutions will get the best rank, zero. Goldberg (1989) proposed an alternative Pareto ranking method. The first non-dominated solutions are assigned rank zero and are subseqently removed from consideration. The remaining non-dominated solutions in the population receive a rank of one and the process is repeated. Both Pareto ranking methods succeed in achieving the goal of multi-objective optimisation: no preference is given to either objective and all non-dominated solutions are assigned the same rank.

For the problems of symbolic regression a very natural second objective, in addition to the error measures, is the complexity, or size, of the solution. Instead of using weighting schemes such as regularisation or minimum description length, Pareto ranking methods as described above serve equally well. The only disadvantage may be that at the end of the run, the user is confronted with a front of solutions instead of a single solution.

Population models

There are several population models in addition to the existence of the already described *steady-state* population, where the population size remains unaltered during a run (or is decreased and increased by one, in another viewpoint). Another frequent population structure is a so-called *generational* model, where an intermediate *mating pool* is produced where the selected individuals reside in order to form the next generation of offspring. The *mating pool* is populated by applying truncation, tournament or fitness-proportional selection. Whether some members of the mating pool are simply copied (reproduced) to the next generation or not defines whether the population model is elitist or not. Elitism ensures that the best population member(s) survive the

Downloaded from http://iwaponline.com/jh/article-pdf/2/1/35/392039/35.pdf
by guest

generation gap in order to keep the best-so-far solution(s) in the evolving population. Tournament selection within a steady-state model is naturally elitist in that the best member is maintained in the population when the inverse tournament is held through picking members without replacement.

Another distinction in population structure can be made through *panmictic* and *distributed* populations. A panmictic population is thoroughly mixed. Selection operates on a global scale, taking the fitnesses of the entire population into account. A distributed population is distributed in a certain spatial sense, and selection operates locally. Examples of such distributed population models are: the *island model* in which there are independently evolving sub-populations, with the occasional *migration* of individuals between islands to exchange genetic material; and the *diffusion model* where the population is distributed within a one-, two- or multi-dimensional grid in which local mixing and selection occur. By distributing the population some parallel evolution can occur, making the algorithm less susceptible to convergence to a local optimum.

In the Bernoulli equation problem, again for simplicity, a *panmictic, generational* model is adopted. The algorithm used is strongly *elitist* as the 15% best solutions are always kept in the population.

The crossover operator

The principal inspiration for the formulation of crossover in GP arises from the biological practice of sexual reproduction. Similarly to what occurs in diploidal reproduction, the action of crossover affects two parental entities' genotypes and recombines them in to produce offspring genotypes. Thus the crossover operator θ affects two evolving entities in such a way that θ: $\Omega \times \Omega \rightarrow \Omega$. Effectively, the crossover operator represents a higher-order transformation of parental entities.

In principle, crossover requires that two 'parent' expressions are divided and recombined into two offspring expressions. The representations of the expressions that are undergoing recombination must be such that the resulting offspring are grammatically (and, specifically, syntactically) correct. The first efforts of the GA community to evolve symbolic expressions (Cramer 1985) were based on the interpretation of a binary string in symbolic form. Despite some limited successes of this approach, it was found that it still necessitated an extensive checking of the grammatical correctness of the resulting formulae, and in cases when the resulting expressions were not correct it necessitated patching and other methods of rehabilitation. In certain cases, the algorithm was even found to spend most of its time carrying out these rehabilitation measures.

In view of these difficulties, Koza (1992) followed a somewhat different path, in which parse trees were manipulated directly in an evolutionary process. The consequence of this approach is that the syntax of the formulae necessarily remains correct (well-defined) regardless of the crossover site. There is then no need for additional corrections and patching. As remarked earlier, this means that the grammatical correctness of statements is not violated under the operation of crossover when these statements are expressed as a parse tree; and this is the principal benefit conferred by using this form of sign vehicle.

The principle of the crossover operator is schematised in Figure 3 using the expression already introduced.

The crossover θ: $\Omega \times \Omega \rightarrow \Omega$, between two parse trees, ξ_1, $\xi_2 \in \Omega$, can be defined through the following procedure:

- select a node, n_1, randomly in the tree ξ_1;
- select a node, n_2, randomly in the tree ξ_2;
- interchange the two sub-trees.

Mutation

Mutation, μ, is a unary-type transformation that alters the individual $\omega \in \Omega$ such that μ: $\Omega \rightarrow \Omega$. Each evolutionary-algorithmic technique defines mutation in a sense that best suits its own purpose. Biologically speaking, mutation denotes a haploid, asexual manipulation of the genome. Traditionally, in genetic algorithms, mutation is referred to as 'bit-flipping', or a segment-inverting operation (Babovic 1993, 1996). In evolutionary programming, a

Downloaded from http://iwaponline.com/jh/article-pdf/2/1/35/392039/35.pdf
by guest

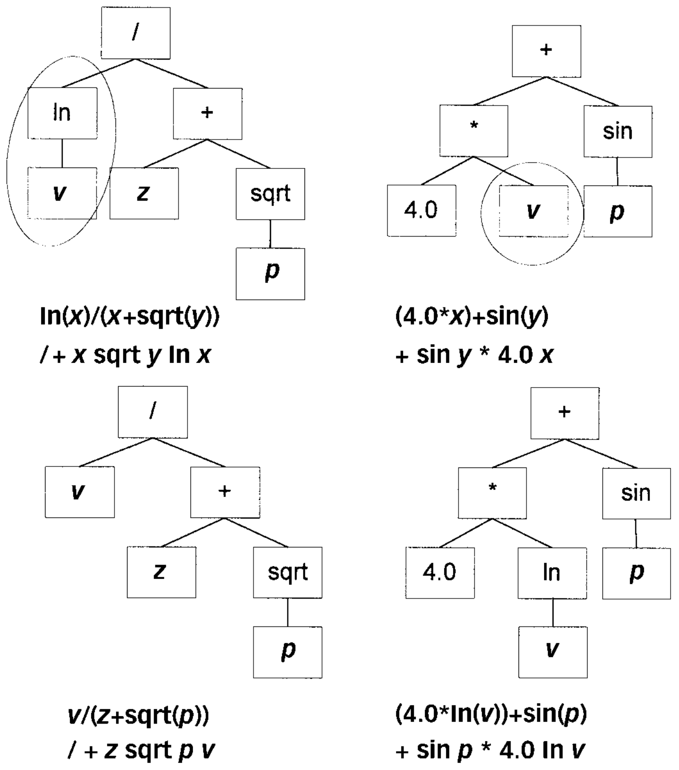


Figure 3 | The action of the crossover operator: the upper part of the figure illustrates parents that are selected for reproduction, and the lower part of the figure illustrates the offspring that are generated after crossover.

Table 5 | Pseudo-code for a mutation.

```
function tree = create_offspring(parent1, parent2)

   tree = cross(parent1, parent2); // Apply crossover always

   if (random(10) = 0)          // 10% of the cases
      mutate_branch(tree, 15); // replace a randomly chosen sub-tree
                               // with a tree of maximum size 15
   end if

   if (random(10) = 0)
      mutate_constant(tree, 5%);
   end if
```

mutation is understood as any manipulation of a structure (Fogel 1992). Evolutionary strategies use the term to describe a variety of different operators (Schwefel 1981).

In genetic programming, however, the action of the mutation operator has frequently been described as a random substitution of a sub-tree with another sub-tree. There are several kinds of (computational) mutations possible. Some examples are:

1. *Branch-mutation*, where a complete sub-tree is replaced with another, and in principle, arbitrary sub-tree that can be generated by one of the initialisation methods described above.
2. *Node-mutation*, which applies a 'random' change to a single node, replacing its original value by another and, again in principle, arbitrary value. In the case of constants, it's value is often slightly altered by using some white noise.
3. *Constant-mutation,* similar as above but now a constant will be selected and mutated by using some white noise.
4. *Inversion-mutation*, which inverts the order in which operands are ordered in an expression in a random way; thus $f(x_1, x_2, x_3)$ might become, for example, $f(x_3, x_1, x_2)$.
5. *Edit-mutation*, which edits the parse tree based on semantic equivalence. This mutation does not alter the phenotype (the function computed) but just produces an equivalent smaller formula. An example of edit-mutation is to replace all occurrences of $(0 + x)$ in a parse tree with x.

Both the crossover and mutation operators try to balance the heredity and variability components of the algorithm. Much research has investigated the influence of specific crossover and mutation schemes with respect to these two criteria.

In the system that is being developed through these pages, the following algorithm outlined in Table 5 is used.

In the present example, the algorithm is employed with two randomly chosen 'parents' from the 15% of survivors of the generation gap as described above.

Soft brood selection

It might be clear from the previous discussion that crossover and mutation are randomised methods that are likely to produce rather erroneous formulae from time to time.

Downloaded from http://iwaponline.com/jh/article-pdf/2/1/35/392039/35.pdf
by guest

Therefore, it might be worthwhile to detect such formulae early before spending too much effort in evaluating them on the entire data set. In soft brood selection (Tackett 1994), when one or two parent formulae are chosen to produce offspring, more offspring than will actually be inserted in the population will be generated using crossover and mutation. Subsequently a *culling function* will be used to determine which offspring will be inserted in the population. This culling function often takes the form of the same objective function that is used overall, with the difference that it will use only a tiny fraction of the data. This culling function is then a computationally cheap method to assess the worth of the offspring. This way early detection of bad offspring can be enforced and less computational effort is spent on evaluating these offspring.

Soft brood selection is not used in our running example.

Determination of the stopping criterion

An evolutionary algorithm is in principle an infinite iteration. In practice however, the run needs to be stopped at a certain point. Two methods of stopping a run may be used: stop after a certain number of generations, or stop after a certain length of wall-clock time has passed. In the Bernoulli example the latter was chosen: a run is stopped after 2 minutes of optimisation.

Summary of the parameter settings

So far several parameters have been introduced. These parameters define a reasonable genetic programming system. One of the properties of most evolutionary algorithms is that they are very robust in the precise parameter settings. Small, or sometimes even large, changes in the parameters have only a very small effect in the optimisation capabilities of the system. Table 6 describes the experiment.

Table 6 | Description of experiment.

Objective	**Find Bernoulli's Law of Energy Conservation**
Terminal set	$\{z, v, p, \Re\}$
Function set	$\{+, *, -, /, \text{power}\}$
Population model	Panmictic, generational, elitist
Selection method	Truncation selection
Truncation percentage	15%
Population size	500
Initialization method	Grow method on size
Initial size of formulae	15
Crossover rate	100%
Branch mutation rate	10%
Constant mutation rate	10%
Constant mutation magnitude	5%
Maximum size of formula	71
Stopping criterion	2 minutes of processing time on a Pentium 233 MHz computer

Results

The genetic programming system was run 30 times with different initial random seeds. Unfortunately, in this setup, genetic programming was not able to find the proper formula. The best formula produced over 30 runs was (in simplified form):

$$E = z + v - 3\,\frac{v}{1+v} + 9\,\frac{v^3}{p}\,. \qquad (11)$$

This formula had a RMS error of 1.71 metres. There are a few reasons as to why the genetic programming system was not able to find the optimal formula. This has mainly to do with the magnitude of the numerical values and the dimensions the problem is stated in. The pressure variable for instance is stated in a numeric magnitude of 10^4, while the other variables and the desired output have a magnitude around 10. As the generation of constants adopted is also biased towards smaller values, it is difficult for the GP system to scale the pressure variables to workable values.

Downloaded from http://iwaponline.com/jh/article-pdf/2/1/35/392039/35.pdf by guest

A second, related, problem is that the problem is stated using particular units of measurement. The formula produced by GP does not take these into account and thus renders a dimensionally incorrect formula. On the surface it seems that by scaling the inputs and outputs the problem would be cast in dimensionless and commensurable terms, thus removing the problems encountered in this experiment. In fact, regression techniques rely on such a scaling.

Conclusion for the Bernoulli experiment

In one sense the experiment failed: this genetic programming system was unable to find the Bernoulli equation in any of the 30 runs. As it is generally more insightful to consider failures than successes, this presents the opportunity to learn something. First, it was identified that the problem lies mostly in the scaling and, more generally, in GP's inability to handle units of measurement. Secondly, the parameters were set at some general values, not optimised for the particular problem at hand. In (symbolic) regression one usually does not approach a problem with such an all-or-nothing attitude, but rather one looks at the results from one or more runs, changes some settings (by for instance adding or removing variables or enlarging or restricting the function set) and performs some more runs. With this strategy very good results can be obtained with this technique. But keep in mind that when using a randomised technique such as GP, *your mileage may vary*.

The following sections will introduce another approach to GP that provides better results with problems involving units of measurement. This approach transcends the level of symbolic regression towards the arguably more useful goal of model induction. With this approach the Bernoulli equation problem was successfully solved (consult Keijzer & Babovic (1999) for details).

GENETIC PROGRAMMING FOR SCIENTIFIC DISCOVERY

Engineering data, in most cases, cannot attain their maximum usefulness until they are connected by a reliable and practicable empirical equation. GP lends itself quite naturally to the process of induction of mathematical models based on observations: GP is an efficient search algorithm that need not assume the functional form of the underlying relation. Given an appropriate set of basic functions, GP can discover a (sometimes very surprising) mathematical model that approximates the data well. At the same time, GP-induced models come in a symbolic form that can be interpreted by scientists (see, for example, Babovic 1996).

However, the application of standard GP in a process of scientific discovery does not always guarantee satisfactory results. Extensions of standard GP as described previously in this text have been an object of several studies (see, for example, Davidson *et al.* 1999). In certain cases, GP-induced relations are too complicated and provide little new insight on the process that generated the data. One may argue that GP, in such situations, blindly fits parse trees to the data (in almost the same way as in Taylor or Fourier series expansion). It can be argued that GP then results in a model with accurate syntax, but with meaningless semantics. In these cases, the dimensions of the induced formulae often do not fit, pointing at the physical uselessness of induced relations.

UNIT TYPING IN GP

The present work is based on an augmented version of GP – dimensionally aware GP – which is arguably more useful in the process of scientific discovery (Keijzer & Babovic 1999).

Nature of measurements

Throughout science, the units of measurement of observed phenomena are used to classify, combine and manipulate experimental data. Measurement is the practice of applying arithmetic to the study of quantitative relations. Every measurement is made on some scale. According to Stevens (1959), to make a measurement is simply to make 'an assignment of numerals to things according to a rule–any

Downloaded from http://iwaponline.com/jh/article-pdf/2/1/35/392039/35.pdf
by guest

rule'. There is a close connection between the concept of a scale and the concept of an application of arithmetic. Different kinds of scales represent different kinds of application of arithmetic. Units of measurement are the names of these scales. Simple unit names such as 'kilogram', 'second', '°C' are used for fundamental and associative scales. Complex unit names, such as 'kg m s^{-2}' are used for derivative scales.

Nature of derivative measurement

Derivative measurement is a measurement by means of constants in numerical laws. Let us more precisely define this concept through consideration of a system A_1, to be any system that meets certain specifications, and suppose that:

$$k_1 = f(l_1, t_1, m_1, \ldots) \tag{12}$$

is a numerical law which is found to be obeyed by all systems of the class (A). At the same time $l_1, t_1, m_1, \ldots$, are the results of any simultaneous measurements made on the system A_1 on any scales of the classes of similar scales (L), (T), (M), ... Let us suppose further that k_1 is a system-dependent constant and that if the systems of the class (A) are arranged in the order of this constant they are also arranged in the order of some quantity d which is known to be possessed by this system. The conditions for saying that we have a scale of measurement of d are then clearly satisfied. Hence, we may take k_1 to be the measure of the quantity d which is possessed by the system A_1 on a derivative scale that depends on the choices of scales from within classes (L), (T), (M), ... Derivative measurement of a quantity d is possible, therefore, if and only if there exists some numerical law relating to the system that possesses the quantity d in which there appears a system-dependent constant such that if these systems are arranged in the order of this system-dependent constant, they are also arranged in the order of d. A derivative scale for the measurement of d is then one that is defined by taking the value of the system-dependent constant (or some strictly monotonic increasing function of it) for some particular choice of independent scales as the measure of the quantity d.

The derivative scales D_1 and D_2 are similar if and only if they are defined on the basis of the same physical law, expressed with respect to the same classes of similar scales. Thus for example, the 'kg m s^{-2}' and the 'ft lb s^{-2}' scales of force are similarly defined. From this follows a straightforward theorem that *similarly defined derivative scales are similar to each other*. This theorem (proved by Ellis 1965, p. 133) is important because it implies that classes of similarly defined derivative scales are simply classes of similar scales and hence may serve as reference classes for the expression of numerical laws in the standard form of the equation as introduced in the sequel.

A second important theorem reads as follows, let:

$$k = f(l, t, m, \ldots) \tag{13}$$

be a numerical law expressed with respect to the classes of similar scales (L), (T), (M), ..., where k is the only system or scale-dependent constant that appears. Let us also refer to the law of the kind given in Equation (13) as a law expressed in the standard form. Then the theorem states that:

Any law expressed in the standard form must be of the form

$$k = \mathrm{C} l^{\mathrm{a}} t^{\mathrm{b}} m^{\mathrm{c}} \ldots, \tag{14}$$

where C, a, b, c, ..., *are constants which are neither system dependent nor scale dependent (see Ellis 1965, pp. 204–206)*.

The importance of this theorem is that it enables us to explain complex unit names and dimensional formulae. Thus Equation (14) defines a class of similar derivative scales for the measurement of some quantity d. To designate this class we could use a simple name, say 'N', but it is obviously more informative to use the dimensional formula $(L)^a (T)^b (M)^c, \ldots$. By doing so, we say something about the form of the law (14) on which our derivative scales are defined.

The theory of dimensional analysis cannot be developed here, but its power depends on the information we pack into dimensional formulae. If we wish to increase this power, we must include more information. This can be done only if we adopt the basic convention of expressing

Downloaded from http://iwaponline.com/jh/article-pdf/2/1/35/392039/35.pdf
by guest

our laws with respect to the classes of similar scales. Thus instead of expressing laws, including angular displacement, with respect to the radian scale and saying (absurdly) that angular displacement is a dimensionless quantity, we should always express laws with respect to the class of scales similar to our radian scale and introduce the dimension of angle into our dimensional formulae.† As demonstrated by Ellis (1965, pp. 145–151), this would increase the power of dimensional analysis.

Introduction of units of measurements in GP

To accommodate the additional information available through units of measurement, the following extensions of standard GP were proposed (Keijzer & Babovic 1999).

In the dimension-augmented setup, every node in the tree maintains a description of the units of the measurement it uses. These units (entirely in the spirit of the standard form of Equation (14)) are stored as arrays of real-valued exponents of the corresponding dimensions. In the present set of experiments only the dimensions of length, time and mass (LTM) are used, but the set up may be trivially extended to include all other SI dimensions (amount of substance, electric current, thermodynamic temperature and luminous intensity). Square brackets are used to designate units, for example [1, −2, 0] corresponds to a dimension of acceleration ($L^1T^{-2}M^0$). Similarly, [0, 0, 0] defines a dimensionless quantity.

Definition of the terminal set

The definition of the terminal set is straightforward, in that variables and constants are accompanied with the exponents of their respective units of measurement. In the running example of the Bernoulli equation, this would read:

$$T = \{z[1, 0, 0], v[1, -1, 0], p[-1, -2, 1], \\ g = 9.81[1, -2, 0], \gamma = 9810[-2, -2, 1]\}. \qquad (15)$$

†Using radians as a separate dimension is not the end of the controversy. For the sake of argument let us introduce the quantity Q measured in radians. Let us also introduce two additional quantities P and R, measured in kilograms. Let us now calculate $W = Q + \text{arcsine}(P/R)$. This would be syntactically perfectly correct, as the arcsine function would expect a dimensionless quantity while returning a value in radians. However, the derivative measurement of W is unacceptable.

For example, $v[1, -1, 0]$ designates a variable, v, with a derived dimension of velocity. User-defined constants can be defined along with their dimensions, such as 9.81[1, −2, 0] defining the Earth's gravitational acceleration.

Randomly generated constants are allowed only as dimensionless quantities ([0, 0, 0]). There is a definitive reason for allowing random numbers to be dimensionless only. Should random constants with random dimensions be allowed, GP would have an easy way of correcting the dimensions by introducing transformation from one arbitrary unit of measurement to another. Some form of pressure should be applied to the application of unit transformation.

Definition of the function set

Application of arithmetic functions on dimension-augmented terminals violates the closure property for these functions (Koza 1992). For example, adding metres to seconds renders a dimensionally incorrect result of the operation. Therefore, the definition of arithmetic operators is augmented to:

(1) specify the transformation of units of measurement;
(2) accommodate units of measurement-related constraints on the application of functions; and
(3) introduce a protection mechanism in order to satisfy the closure property.

Table 7 summarises the effects of the application of functions on units of measurement and specifies constraints on the applications of functions. For example, exponentiation of a value can only take place when the operand is dimensionless, in which case the result of the operation is also a dimensionless value. Similarly, addition and subtraction are constrained so that their operands must have the same dimensions. Multiplication and division combine the exponents by adding and subtracting the dimension exponents respectively. The standard Power function can be applied to dimensionless values only, whereas PowScalar can be applied to dimensional operands, affecting their dimensions correspondingly. Other functions can be defined in similar ways.

Downloaded from http://iwaponline.com/jh/article-pdf/2/1/35/392039/35.pdf by guest

Table 7 | Effects and constraints that units of measurement impose on the function set.

Function	Operand dimensionality	Result
Exponentiation:	[0, 0, 0]	[0, 0, 0]
Logarithm:	[0, 0, 0]	[0, 0, 0]
Square root:	$[x, y, z]$	$[x/2, y/2, z/2]$
Addition:	$[x, y, z]$, $[x, y, z]$	$[x, y, z]$
Subtraction:	$[x, y, z]$, $[x, y, z]$	$[x, y, z]$
Multiplication:	$[x, y, z]$, $[u, v, w]$	$[x + u, y + v, z + w]$
Division:	$[x, y, z]$, $[u, v, w]$	$[x - u, y - v, z - w]$
Power:	[0, 0, 0], [0, 0, 0]	[0, 0, 0]
PowScalar (c):	$[x, y, z]$	$[x$*c, y*c, z*c$]$
If less than zero:	[0, 0, 0], $[x, y, z]$, $[x, y, z]$	$[x, y, z]$

Closure and strong typing

The definition of the function and terminal set as it stands violates the closure property (Koza 1992). The closure property requires that each of the functions in the function set be able to accept, as its arguments, any value and data type that may be returned by any function, and any value and data type that may possibly be assumed by any terminal.

This problem was already encountered in symbolic regression where the division function as such violates the closure problem. One of the methods to enforce closure was to protect such functions. In the unit augmented version of genetic programming this path is taken.

The functions in this system are protected so that whenever incompatible data types are encountered as arguments to a function, the arguments are multiplied with a constant of magnitude 1.0 whose unit of measurement is such that it will transform the argument's unit to the desired unit. Because of the particular structure of the dimension equations, it is possible to transform a unit of measurement to any other unit of measurement with such a multiplication.

For an example of this operation, consider the addition function that receives one argument stated in metres and another argument stated in seconds. As such it is an undefined operation. It is however possible to transform the first argument to seconds by multiplying it with a constant of magnitude 1, stated in seconds per metre, or, equivalently, multiply the second argument with a constant of magnitude 1 stated in metres per second. A third possibility is to render both arguments dimensionless.

Although at first sight, it may appear that this method eliminates all the problems related to dimensions, it is evident that this method 'fixes' any dimensionality-related problem by introducing physically meaningless transformations into evolving formulae. As such this method does not contribute anything to standard GP.

To help the system find dimensionally correct formulations, a second objective next to the common goodness-of-fit criterion is employed. This objective takes full advantage of the explicit representation of the dimensions as a vector of real valued exponents. Define the distance of an expression between a dimensionally correct formulation and its dimensional 'fix' to be the sum of all these arbitrary transformations as:

$$\text{Goodness-of-dimension} = \sum(\,|x_i| + |y_i| + |z_i|\,), \quad (16)$$

where the subscript i ranges over all transformations applied to the formula, and x, y and z are the components of the corresponding dimension vector. This goodness-of-dimension acts as an effective measurement of distance from desired dimensions and is treated as an additional measure of fitness. Goodness-of-dimension can be combined with the goodness-of-fit statistic in a multi-objective optimisation fashion.

It is however also possible, at the expense of a rather large modification, to modify the present approach and include dimensional constraints more strongly. One can adopt strict admissibility and admit only those formulations that are dimensionally correct. Montana (1995) proposed a so-called strongly typed genetic programming system (STGP) where the closure property is not enforced,

Downloaded from http://iwaponline.com/jh/article-pdf/2/1/35/392039/35.pdf
by guest

but rather the system is constrained so that data types are respected and only well-formed expressions are ever created. This solves the problem of a naïve implementation where expressions that violate the typing scheme are destroyed. In such a naïve setting the number of ill-formed expressions (the ones with goodness-of-dimension larger than zero) is very likely to be large. It is consequently very likely that the system will spend most of its time creating, evaluating and dismissing solutions, which is not far from a random walk.

Consequently the mechanism proposed here is not a form of strongly typed GP as it *does* maintain the closure property through a protection mechanism. A strongly typed GP would initialise and keep all expressions dimensionally correct throughout the evolutionary process. For ill-posed and incomplete problems (for example the ones in which one of the observations is not supplied), a strongly typed GP would fail even at initialisation. The adopted, weakly typed approach allows dimensionally incorrect solutions that are repaired at run-time. At the same time, evolutionary pressure on dimensional incorrectness is added through a supplementary objective. The present version of GP is therefore based on a preference towards dimensional consistency rather than an enforcement of this quality.

It is believed that this version of GP may be more appropriate in the process of scientific discovery. Let us accept it—science is packed with 'magical numbers' (for example, the Kolmogorov–Obukhkov 2/3 exponent, which is a constant source of controversy in turbulence modelling). Dimensions of some of these coefficients are everything but transparent. For example, the Chezy number, at the very centre of this study has dimensions of $m^{1/2}\,s^{-1}$ and is still referred to as a roughness coefficient! The fact is that the Chezy number, *C*, allows for a range of phenomena not explicitly taken into account in the well-known formula defining average flow velocity in steady conditions:

$$u = C\sqrt{(Ri)}. \qquad (17)$$

Since functional similarity to the natural system is supposed to be comprehended by the equation itself, it is this *calibration coefficient C* that then must capture the exact correspondence between the model and the real world. Parameters of this sort serve in effect as *error compensation devices* that artificially adjust the model results to compensate for the fundamental discrepancies that exist between the real world and its representation within the model. Chezy's *C* is much more than a coefficient that can be associated with roughness forces only, and as such is not well-defined in nature. Instead it exists only at the interface between nature and model. It has to accommodate both: a part related to physical processes, but also a part that has to do with our schematisation of nature within a model. One may even ask 'What is the physical meaning of such a parameter—how well is it grounded, and indeed is it grounded at all?' We may be able to read a certain 'physical meaning' into such calibration parameters, but they do not exist as such and are thus 'disconnected' in a fundamental way from the world that they are supposed to model (Minns & Babovic 1996).

At the same time, it is quite obvious that the dimensions of such calibration coefficients are chosen in a way that the overall dimension within a model will match. As such, they may have little physical meaning.

Strongly typed GP would only reinforce such a position and that may not be the most appropriate one for the process of scientific discovery. If we are to design an algorithm that can be truly useful in a process of scientific discovery, we have to be able to utilise all the available information (including units of measurements) and possibly extend upon it.

Dimension-based brood selection

An alternative brood selection is applied in this system (Keijzer & Babovic 1999). In this case the culling function used in dimensionally aware GP is the goodness-of-dimension of a formula. This evaluation is very inexpensive as it can be calculated independently of the training set and it requires a single pass through the parse tree.

The present implementation reads as follows: two parents are chosen for crossover; they produce *m* offspring by repeated application of the random sub-tree crossover operation; constraint violations are corrected for dimensions in the manner outlined above; and, finally,

Downloaded from http://iwaponline.com/jh/article-pdf/2/1/35/392039/35.pdf
by guest

the best among the m offspring with respect to goodness-of-dimension are added to the intermediate population.

ADDITIONAL RESISTANCE TO FLOW INDUCED BY FLEXIBLE VEGETATION

Based on the bitter experiences of recent floods in Europe and the USA, many pressure groups have promoted the restoration of natural wetlands that would act as natural 'sponges' capable of absorbing excess water and thus reducing flooding risks. Various governments around the world (The Netherlands, USA, etc.) have embarked on major wetland protection and restoration tasks. This accelerating movement to restore wetlands, and especially to return flood plains to more natural conditions, has lead to the widespread introduction and propagation of reeds and other forms of more or less flexible vegetation. These reeds and other plants then influence flows across wetlands. The need to control flooding events and intensity using numerical models and other tools necessitates a better understanding of the influence of this type of vegetation on flow.

The wetland restoration projects favour the growth of reeds and other similar vegetation within a river basin. The presence of vegetation influences the flow conditions, and in particular the bed resistance, to a large degree. However, the influence of the rigid and flexible vegetation on flow conditions is not understood well enough. The present work is concerned with the understanding and formulation of the underlying physical processes. Some laboratory experiments using physical scale modelling have been performed (Larsen *et al.* 1990; Tsujimoto *et al.* 1993), but only over a limited range and with variable success. Similarly, although field experiments are continuing, the data available remains scarce.

More recently, a numerical model has been developed with the intention of deepening the understanding of the underlying processes (Kutija & Hong 1996). This model is a one-dimensional vertical model based on the equations of conservation of momentum in the horizontal direction. This numerical model is employed here as an experimental apparatus in the sense that this, fully deterministic (even if highly parameterised), model is used as a source of data that are then further processed by two apparently different methodologies in order to induce a more compact model of the additional bed resistance caused by vegetation.

Short description of the Kutija-Hong model

As indicated earlier, the Kutija-Hong model is based on a differential equation of the conservation of momentum in the horizontal direction:

$$\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}+w\frac{\partial u}{\partial z}+\frac{1}{\varrho}\frac{\partial P}{\partial x}-gi-\frac{1}{\varrho}\frac{\partial \tau}{\partial z}+\frac{F_x}{\varrho\Delta x\Delta z}=0 \qquad (18)$$

where:

u	denotes horizontal velocity
w	denotes vertical velocity
P	denotes pressure
ϱ	denotes density of water
g	denotes gravity acceleration
i	denotes bottom slope
τ	denotes shear stress

and the term $F_x/\varrho\Delta x\Delta z$ represents the additional specific drag forces.

The model is developed to address the effects of flexible vegetation in steady flow conditions only, so the convective momentum terms and pressure term are neglected. The numerical model is based on a finite difference implicit approximation for the unknown velocities at the discretisation points.

The model takes into account the effects of shear stresses on the bed and the additional forces induced by flow through vegetation, and it possesses a facility to adapt initial distributions of velocities over discretisation points on the vegetation stems. In addition to representing these influences, the model takes into account the bending of the reeds under the loads produced by the drag forces. Owing to bending, the effective reed height is reduced, which implies a lesser drag, and this in turn implies a reduction in the drag-induced deflections of the vegetation. These processes are resolved through an iterative numerical process. For a more detailed description and a discussion of the specifics of the model see Kutija & Hong (1996).

Downloaded from http://iwaponline.com/jh/article-pdf/2/1/35/392039/35.pdf
by guest

MODEL INDUCTION

Data

The Kutija-Hong model, used as a generator of data, was in effect used as a truthful representation of a physical reality, while providing the conveniences of fast calculation and an ability to produce results with any degree of scale refinement. In this way, the numerical model not only replaced physical scale modelling facilities within this exploratory environment, but also introduced several intrinsic advantages over scale models. It is well known that so-called *roughness scaling* is one of the principal difficulties in the development of physical models. Since the roughness is the primary phenomenon in question here, the issue of its physical correctness remained critical. The 'realism' of the complete numerical model was reasonably well proven against experimental data in the case of stiff (non-flexible) vegetation (Kutija & Hong 1996).

The model has been run with a wide range of input parameters in order to create training data. As the first attempt towards the development of a model of additional roughness, only the effects of non-flexible reeds with high stiffness were simulated. Altogether, some 4,800 items of training data were generated. The training data consisted, in the first instance, of dimensional numbers formed from:

- water depth h_w,
- reed height h_r,
- reed diameter d,
- number of individual reed shoots per square metre m,
- numerical parameter p related to the eddy-viscosity approximation and its further relation to the vegetated layer height.

The input data varied in the ranges:

$$2.5 \le h_w \le 4.0$$

$$0.25 \le h_r \le 2.25$$

$$50 \le m \le 350$$

$$0.001 \le d \le 0.004$$

$$0.4 \le p \le 1.0$$

Table 8 | The best performing expressions for the dimensional case (Babovic 1996).

```
(/ (exp (-d hr))
 (-(rlog (/ (rlog (/ m hw))
     (sqrt (sqrt (/ d d)))))
  (* (/ hr
    (sqrt (sqrt (/ d
       (sqrt (sqrt (exp (-d hr))))))))
   (* (* (rlog (/ (rlog (/ m hr))
      (exp (-d hr))))
   (* (-(-d hr) hr) -0.00410)) p))))
```

Results obtained using genetic programming

Previously reported work

The following two sections are adapted from Babovic (1996) and Babovic & Keijzer (1999). In the first instance, a standard genetic programming environment has been set up. The results of Kutija-Hong simulations were presented as C_{new}: the Chezy number corresponding to the flow conditions with developed vegetation.

Dimensional values

In the first attempt, all the data were used in their original, dimensional form. Such an approach was adopted so as to introduce the least possible level of 'presuppositions' in the model evolution. From the perspective of accuracy (goodness-of-fit), the obtained results were quite satisfactory (see Table 8).

The shear complexity of the formulation above almost immediately eliminates it from a knowledge induction framework. In order to improve interpretability, Babovic & Keijzer (1999) employed an early version of dimensionally aware GP with the best performing formula being:

$$C_{\text{new}} = \frac{55.93 h_w}{h_w + d + 3dm + 3h_r - p} - 11.39 \ . \qquad (19)$$

There is some dispersion for the higher values of C_{new} but otherwise the equation exhibits good accuracy (see Table 9). The scatter plot for Equation (19) is depicted in Figure 4.

However, it has to be emphasised that Equation (19) is not dimensionally correct. This shortcoming can be

Downloaded from http://iwaponline.com/jh/article-pdf/2/1/35/392039/35.pdf
by guest

Table 9 | Statistical summary for expressions (19) and (25).

	r	R^2	RMS	NRMS
(19)	0.96	0.92	2.880	27.89%
(25)	0.94	0.89	0.076	37.47%

corrected by introducing an auxiliary constant with dimension of length and magnitude of 1.0 that should be multiplied with dimensionless p to correct dimensions. At the same time, both constants 55.93 and 11.39 should be assigned dimensions of the Chezy number $[\mathrm{m}^{1/2}\ \mathrm{s}^{-1}]$ to make this formula dimensionally correct.

As indicated earlier, such behaviour is not surprising when applying standard instances of GP. Satisfactory goodness-of-fit may be obtained, but the semantics of the generated expressions cannot be warranted.

Dimensionless values

One standard approach in avoiding potential conflicts with incorrect dimensionality of induced formulations is to use dimensionless values. This is a 'standard scientific practice'. Units of measurements are effectively eliminated through the introduction of dimensionless ratios. Once the dimensionless numbers are used instead of the original dimensional values the problem of dimensional correctness is conveniently avoided, as all analysed quantities are dimension-free. It is also argued that dimensionless ratios collapse the original search space, making it more compact, thus resulting in a more effective behaviour of algorithms that fit models to the data. At the same time, the information contained in the units of measurements is ignored entirely, effectively violating the basic premise of dimensional analysis.

In this, dimensionless case, the results of Kutija-Hong simulations were presented as a dimensionless ratio η of an original Chezy number, that corresponding to an absence of vegetation, and a new Chezy number, that corresponding to developed vegetation. This ratio η can be conveniently incorporated in the Chezy formula for velocity under steady flow conditions:

$$u = \eta C \sqrt{(Ri)}\ . \tag{20}$$

For example, for $\eta = 0$, the resistance to flow becomes infinitely large, thus stopping the water flow, which is, physically a highly unlikely situation. The smallest values of η experienced with the Kutija-Hong numerical model were η 0.1. For $\eta = 1$, the influence of vegetation on the roughness is zero.

Another set of model induction experiments has been performed, but in this case a collection of dimensionless numbers has been used. The dimensionless ratios introduced were defined as follows:

$$h_{rel} = h_w / h_r; \tag{21}$$

$$w_d = h_w / d; \tag{22}$$

$$r_d = h_r / d; \tag{23}$$

$$h_{hwhrd} = (h_w - h_r)/md. \tag{24}$$

In addition to these, parameters p and m were used without any changes. The best performing expression is:

$$\eta = \left[2 \left(\frac{h_w}{h_r} \right)^{0.75} + \sqrt{\frac{h_w - h_r}{md}} \right] 0.06 - 0.09\ . \tag{25}$$

The performance statistics are presented in Table 9.

Where:

r	denotes correlation coefficient,
R^2	denotes Pearson's product moment correlation squared,
RMS	denotes Root Mean Squared Error,

NRMS denotes Normalised Root Mean Squared Error where RMS is normalised by the standard deviation of the desired outcome (Kutija-Hong model in this case).

The first interesting observation is slightly counter-intuitive: the accuracy of induced formulation in a dimensionless case is not as good as the accuracy of dimensional formulation. Even if the accuracy of η would be acceptable, it is the behaviour of Chezy's C (calculated as $C_{\mathrm{new}} = \eta C_{\mathrm{org}}$) that is highly undesirable in this case (see the upper two graphs in Figure 4). Compression of the search space through the use of

Downloaded from http://iwaponline.com/jh/article-pdf/2/1/35/392039/35.pdf
by guest

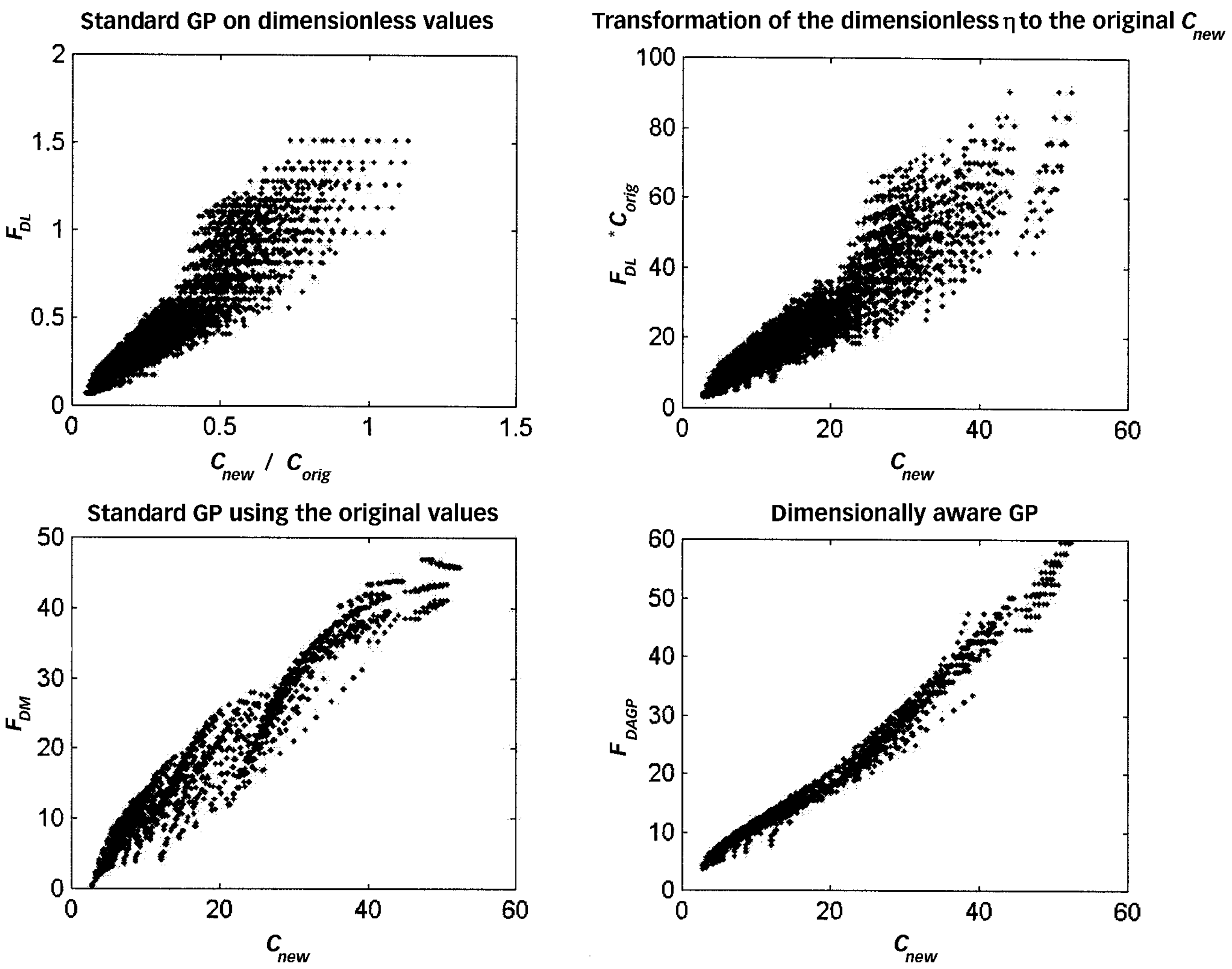


Figure 4 | Scatter plots for expressions (19), (25) and (26). The upper two graphs depict the behaviour of an induced relation in the dimensionless case (denoted as F_{DL}). The graph in the upper left corner is a scatter plot for dimensionless h, whereas the graph in the upper right corner is a plot for the corresponding Chezy coefficient. The graph in the lower left corner depicts performance in the dimensional case (F_{DM}). Finally the graph in the lower right corner depicts the results for the dimensionally aware GP (F_{DAGP}).

dimensionless values obviously incorporates several hidden risks that need to be handled with considerable care.

Results based on dimensionally aware genetic programming

By way of comparison, a dimensionally aware genetic programming environment was set up in such a way as to comprehend all measured data and not the corresponding dimensionless parameters based on the measurements. The purpose for conducting such an experiment was to test whether such a dimensionally aware GP setup is capable of creating a dimensionally correct and still accurate formulation. Since the pre-processing of raw observations 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 one is presented here:

Downloaded from http://iwaponline.com/jh/article-pdf/2/1/35/392039/35.pdf
by guest

Table 10 | Statistical summary for expressions (19), (25) and (26).

	r	R^2	RMS	NRMS
(19)	0.96	0.92	2.880	27.89%
(25)	0.94	0.89	0.076	37.47%
(26)	0.98	0.97	1.800	17.44%

$$C_{\mathrm{new}} = \frac{1}{h_r^{\frac{1}{2}}} g \left[\frac{h_w - h_r}{dm} p \right]^{\frac{1}{4}}. \qquad (26)$$

The degree of accuracy of the induced expression is quite satisfactory. A statistical measure of conformity, such as the Pearson's R^2, gives a value of 0.97. This provides a considerable improvement over the values of 0.92 or 0.89 based on the relations described in Equations (19) and (25) respectively. The total error over the data set is reduced and all other statistical measures of accuracy disclose improvements (see Table 9 and Table 10 for details).

At the same time the formula is dimensionally consistent, it uses some of the most relevant physical properties in the relevant context. For example, the dimensionless term $(h_w - h_r)\ p/dm$ describes a ratio between the effectively available cross-section $(h_w - h_r)\ p$ and a part of the cross-section that is blocked by the plants per unit width of the channel.

The remaining group $g/h_r^{\frac{1}{2}}$ represents a ratio of gravity forces and flow resistance 'force' expressed through the reed height.

In the present case, evolution produced a dimensionally consistent, meaning-rich formulation that is very accurate. It did so without employing assumptions (other than units of measurement); the process operated only on raw observations. Still, Equation (26) is not dimensionally correct; it does not produce the derived units for the Chezy coefficient. This may originate in at least two causes:

1. Incomplete data: in the present data set neither time nor elasticity components were provided (the authors supplied $g = 9.81\ \mathrm{m/s^2}$). The next iteration in this direction must resolve this deficiency in one way or another.
2. Another possibility may be that the problem is simply ill-posed: the authors attempted to model Chezy's C despite their reservation about its grounding (however, without any reservations about its usefulness).

The authors maintain that this particular approach is very useful for the purposes of model induction. This dimensionally aware attitude is open-ended in that it does not strictly adhere to the dimensional analysis framework. Such an approach provides the possibility of achieving a superior goodness-of-fit while sacrificing goodness-of-dimension (while obtaining good performance in both objectives). The authors will go even further to claim that the dimensionally aware approach is much more useful than a strict use of dimensional analysis to create and use only dimensionless ratios.

At the same time, the authors remind the reader that the object of the presented exercise is to find an *empirical equation* based on data. The present work cannot be characterised as a search for a universal law (though it might help). The rationale of this approach is to utilise whatever information is available, yet not to bias this search too strongly. Being able to use units of measurements (either through the dimensionally aware or through strong adherence to dimensional analysis) provides an opportunity to truly *mine the knowledge from the data,* to learn more from data and other associated information.

One open question still remains: *what to do with the proposed formulae*? Fortunately, the primary purpose of this text (in the light of the previous discussion) is not to propose a formulation for the additional resistance to flow induced by vegetation, but to demonstrate an alternative discovery process. The physical interpretation of Equations (19), (25) and (26) requires a wealth of knowledge in flow retardance and hydraulic resistance. The authors feel that such a discussion would fall outside the immediate scope of the present paper.

Thus, the final proposal of the best formula is not even attempted here. Such a choice should be made by the experts in the field, by the people who can competently

Downloaded from http://iwaponline.com/jh/article-pdf/2/1/35/392039/35.pdf
by guest

judge the quality of the data sets used, interpret the induced model and in the end choose the one that makes most sense. We would rather view such results as a potential opening of a new research direction in this field (and for that matter any other domain), so that we can talk about knowledge–discovery-driven domain research. Domain experts can be exposed to a completely new set of formulations, off the beaten track, yet within the domain of physical validity. Unfortunately, no physical experimental data were available for verification of the model. Owing to this lack of experimental data, the authors invite other researchers to check these models using their own data. The experts should make the ultimate step in this process.

Although no direct evidence has been offered in this work, a few concluding remarks need to be forwarded in the favour of the performance of dimensionally aware GP. The convergence rate of such an augmented version of GP is much better than the one in standard GP. The reasons for such a behaviour are fairly obvious: units of measurement are taken into account as an additional objective so that this information also directs the search process. One may say that GP effectively spends more of its time on performing physically more sensible operations in an altered search space. At the same time, standard GP often grows parse trees of a considerable size in order to meet goodness-of-fit criterion only. The interested reader is referred to Keijzer & Babovic (1999) for a direct comparison of the two approaches.

Fundamentally, augmentation of GP with dimensional information adds a descriptive, semantic component to the algorithm. This is in addition to the functional semantics that define the manipulation on numbers. While functional semantics ground the formulae in mathematics, the dimensional semantics ground them in the physical domain.

CONCLUSIONS

The described work is part of a research effort aiming at providing new (and sometimes provocative) hypotheses built from data alone. The ultimate objective is to build models that can be interpreted by the domain experts. Once a model is interpreted, it can be used with confidence. It is only in this way that one can take full advantage of knowledge discovery and advance our understanding of physical processes.

Although the results presented here are based on preliminary experiments, one main conclusion can be drawn. Traditionally, dimensionless numbers are used as the dominant vehicle in interpretation and modelling of experimental values. Such a choice is natural as this alternative 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 the one achieved when a more conventional approach is followed.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge Dr Vedrana Kutija for the use of data. This work was in part funded by the Danish Technical Research Council (STVF) under the Talent Project No. 9800463 entitled 'Data to Knowledge—D2K'. Their support is greatly appreciated. For more information on the project visit *http://projects.dhi.dk/d2k*.

REFERENCES

Abbott, M. B. 1993 The electronic encapsulation of knowledge in hydraulics, hydrology and water resources. *Adv. Wat. Res.* **16**, 21–39.

Babovic, V. 1993 Evolutionary algorithms as a theme in water resources. In *Scientific Presentations, AIO meeting '93*, pp. 21–36. Delft University of Technology.

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

Downloaded from http://iwaponline.com/jh/article-pdf/2/1/35/392039/35.pdf
by guest

Babovic, V. & Keijzer, M. 1999 Computer supported knowledge discovery—a case study in flow resistance induced by vegetation. In *Proceedings of the XXVI Congress of International Association for Hydraulic Research, Graz, 1999*.

Bohm, W. & Geyer-Schulz, A. 1996 Exact uniform initialization for genetic programming. In *Foundations of Genetic Algorithms IV* (ed. R. K. Belew & M. Vose). University of San Diego, CA, USA, Morgan Kaufmann.

Box, G. E. P. 1957 Evolutionary Operation—a method for increasing industrial productivity. *Appl. Stat.* **6**, 81–101.

Bremermann, R. M. 1958 A learning machine. *IBM Jl Res. Dev.* **2**, 2–13.

Cramer, N. L. 1985 A representation for the adaptive generation of simple sequential programs. In *Proceedings of an International Conference on Genetic Algorithms and Their Applications* (ed. J. J. Grefenstette), pp. 183–187.

Davidson, J. W., Savic, D. & Walters, G. A. 1999 Method for the identification of explicit formulae for the friction in turbulent pipe flow. *J. Hydroinformatics* **1**(2), 115–126.

Ellis, B. D. 1965 *Basic Concepts of Measurements*. Cambridge University Press.

Fayyad, U., Piatetsky-Shapiro, G. & Smyth, P. 1996 From data mining to knowledge discovery: an overview. In *Advances in Knowledge Discovery and Data Mining* (ed. U. Fayyad, G. Piatetsky-Shapiro, P. Smyth & R. Uthurusamy), pp. 1–36. MIT Press, Cambridge, MA.

Fogel, D. B. 1992 Evolving Artificial Intelligence. Ph.D. thesis, University of California at San Diego.

Fogel, D. B. 1993 Evolving behaviours in the iterated prisoner's dilemma. *Evolutionary Computation* **1**(1), 77–97.

Fogel, D. B., Fogel, L. J. & Porto, V. W. 1990 Evolving neural networks. *Biological Cybernetics* **63**(6), 487–493.

Fogel, L. J., Owens, A. J. & Walsh, M. J. 1966 *Artificial Intelligence Through Simulated Evolution.* Ginn, Needham Height.

Foseca, C. M. & Fleming, P. J. 1995 *An overview of Evolutionary Algorithms in Multiobjective Optimization* **3**(1), 1–16

Friedberg, R. M. 1958 A learning machine. *IBM Jl Res. Dev.* **2**, 2–13.

Goldberg, D. E. 1989 *Genetic Algorithms in Search, Optimisation and Machine Learning*. Addison Wesley, New York.

Grefenstette, J. 1997 Proportional selection and sampling algorithms. In *Handbook of Evolutionary Computation* (ed. D. Back, B. Fogel & A. Michalewicz), C2:2:1. Oxford University Press.

Harvey, I. 1993 Artificial Evolution and Adaptive Behaviour. Ph.D. thesis, University of Sussex, UK.

Holland, J. H. 1975 Adaptation in Natural and Artificial Systems. University of Michigan, Illinois.

Keijzer, M. & Babovic, V. 1999 Dimensionally aware genetic programming. In *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference, July 13–17, 1999, Orlando, Florida USA* (ed. W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela & R. E. Smith). San Francisco, Morgan Kaufmann.

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

Koza, J. R. 1994 *Genetic Programming 2: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge, MA

Kutija, V. & Hong, H. T. M. 1996 A numerical model for addressing the additional resistance to flow introduced by flexible vegetation. *J. Hydraul. Res.* **34**(1), 99–114.

Larsen, T., Frier, J. O. & Vestergraard, K. 1990 Discharge/stage relation in vegetated Danish stream. In *International Conference on River Flood Hydraulics.*

Margalef, R. 1968 Perspectives in Ecological Theory. University of Chicago, Ill.

Maynard Smith, J. 1975 *The Theory Of Evolution*. Penguin, London.

Michalewicz, A. 1992 *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, Berlin.

Minns, A. W. & Babovic, V. 1996 Hydrological Modelling in a Hydroinformatics Context. In *Distributed Hydrological Modelling* (ed. M. B. Abbott & J. C. Refsgaard), pp. 297–312. Kluwer Academic Publishers.

Minsky, M. 1963 Steps towards artificial intelligence. In *Computers and Thought* (ed. E. A. Feigenbaum & J. Feldman), pp. 379–384. McGraw Hill, New York.

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

Radcliffe, N. J. & Surry, P. D. 1994 The reproductive plan language RPL2: motivation, architecture and applications. In *Genetic Algorithms in Optimisation, Simulation and Modelling* (ed. J. Stender, E. Hillebrand & J. Kingdom), pp. 65–94. IOS Press, Amsterdam.

Schwefel, H.-P. 1981 *Numerical Optimisation of Computer Models.* Wiley, Chichester.

Stevens, S. S. 1959 Measurement, psychophysics and utility. In *Measurements: Definitions and Theories* (ed. C. W. Churchman & P. Ratoosh). New York.

Tackett, W. A. 1994 Recombination, Selection, and the Genetic Construction of Computer Programs. Ph.D. thesis, University of Southern California.

Tsujimoto, T., Okada, T. & Omata, A. 1993 Field measurements of turbulent flow over vegetation on flood plain of river Kakehaski. KHL Progressive Report, Hydraulic Laboratory, Kanazawa University.

Downloaded from http://iwaponline.com/jh/article-pdf/2/1/35/392039/35.pdf
by guest