Accuracy in Symbolic Regression

Michael F. Korns

Korns Associates, 98 Perea Street, Makati 1229, Manila Philippines
mkorns@korns.com.

Abstract.

This chapter asserts that, in current state-of-the-art symbolic regression engines, accuracy is poor. That is to say that state-of-the-art symbolic regression engines return a champion with good fitness; however, obtaining a champion with the correct formula is not forthcoming even in cases of only one basis function with minimally complex grammar depth.

Ideally, users expect that for test problems created with no noise, using only functions in the specified grammar, with only one basis function and some minimal grammar depth, that state-of-the-art symbolic regression systems should return the exact formula (or at least an isomorph) used to create the test data. Unfortunately, this expectation cannot currently be achieved using published state-of-the-art symbolic regression techniques.

Several classes of test formulas, which prove intractable, are examined and an understanding of why they are intractable is developed. Techniques in Abstract Expression Grammars are employed to render these problems tractable, including manipulation of the epigenome during the evolutionary process, together with breeding of multiple targeted epigenomes in separate population islands.

A selected set of currently intractable problems are shown to be solvable, using these techniques, and a proposal is put forward for a discipline-wide program of improving accuracy in state-of-the-art symbolic regression systems.

Key words: Abstract Expression Grammars, Differential Evolution, Grammar Template Genetic Programming, Genetic Algorithms, Particle Swarm, Symbolic Regression.

1 Introduction

The discipline of Symbolic Regression (SR) has matured significantly in the last few years. There is at least one commercial package on the market for several years *http://www.rmltech.com/*. There is now at least one well documented commercial symbolic regression package available for Mathmatica *www.evolved-analytics.com*. There is at least one very well done open source symbolic regression package available for free download *http://ccsl.mae.cornell.edu/eureqa*. In addition to our own ARC system (Korns 2010), currently used internally for massive (million row) financial data nonlinear regressions, there are a number of other mature symbolic regression packages currently used in industry including (Smits 2010) and (Castillo 2010). Plus there is an interesting work in progress by (McConaghy 2009).

During the process of enhancing our ARC system with the latest thinking in published symbolic regression papers, we ran across several test problems for which our system failed to return the correct formula. Normally this is not surprising in large scale regression with much noise; however, these test problems were generated with no noise and were fairly simplistic formulas of only one basis function with minimally complex grammar depth.

After further study it is now apparent that there are very large numbers of simple test formulas against which current state-of-the-art symbolic regression systems suffer poor accuracy. For these intractable problems state-of-the-art symbolic regression engines fail to return a champion with the correct formula.

This is a serious issue for several reasons. First, users expect to receive a correct formula when inputting a simple test case. When a correct formula is not forthcoming, user interest and trust in the symbolic regression system wanes. Second, if symbolic regression cannot return a correct formula in even simplistic test cases then symbolic regression loses its differentiation from other black box machine learning techniques such as support vector regression or neural nets. Third, from its very inception (Koza 1992) symbolic regression has been represented as a technique for returning, not just coefficients, but a correct formula. If this claim cannot be fulfilled by independent scientific review, a serious reputational issue will develop and research money will flow in other directions.

This chapter begins by outlining the accuracy issue. A simple symbolic regression grammar of fifteen obvious mathematical functions is established. All test cases are limited to a single basis function of no more than three grammar nodes deep. For all test cases the data is limited to ten thousand sample points, of five features each, and absolutely no noise. Even with these very severe limitations, large numbers of simple formulas are shown to be intractable.

The chapter continues with an examination of techniques which allow ARC to solve these previously intractable problems. The chapter closes with a proposal for a discipline wide approach to solving our symbolic regression accuracy issues.

1.1 Testing Regimen

Our testing regimen uses only statistical best practices out-of-sample testing techniques. We test each of the test cases on matrices of 10000 sample points and five features with absolutely no noise. Each sample point $\mathbf{x} = <\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3,\mathbf{x}_4,\mathbf{x}_5>$ is associated with a dependent variable $\mathbf{y}$. The data is constructed so that there is an exact functional relationship between each sample point and each associated dependent variable: $\mathbf{F(x)} = \mathbf{y}$.

For each test, a training matrix is filled with pseudo random numbers between -50 and +50 (*each test is random but reproducible on demand*). The test case target functions, **F**, are limited to one basis function whose maximum depth is three grammar nodes. The target function for the test case is applied to the training matrix to compute the dependent variable. The symbolic regression system is trained on the training matrix to produce the regression champion. Following training, a testing matrix is filled with random numbers between -50 and +50. The target function for the test case is applied to the testing matrix to compute the dependent variable. The regression champion is evaluated on the testing matrix for all scoring (i.e. out of sample testing).

Our fitness measure is normalized least squared error (NLSE) as defined in (Korns 2009). Normalized least squared error is the least squared error value divided by the standard deviation of Y. A returned regression champion formula is considered *accurate* if the normalized least squared error (NLSE) on the testing run is .0001 or below. This approach allows isomorphs, such as (x1+x3) and (x3+x1), to be included in the accurate category. It also allows formulas to be considered accurate if they are not isomorphic but are so statistically close as to be approximately identical.

All results in this paper were achieved on a workstation computer, specifically an Intel Core 2 Duo Processor T7200 (2.00GHz/667MHz/4MB), running our Analytic Information Server software generating Lisp agents that compile native code using the on-board Intel registers and on-chip vector processing capabilities so as to maximize execution speed. Details can be found at *http://www.korns.com/Document Lisp Language Guide.html*. Furthermore, our Analytic Information Server is available in an open source software project at *aiserver.sourceforge.net*.

All tables have omitted run timings in favor of numbers of candidate Well Formed Formulas (WFFs) examined. This allows the comparison of results across disparate computer systems and between disparate symbolic regression packages as well as cloud computing systems. Furthermore, numbers of individuals (WFFs) examined is often proposed as a fundamental measure of the actual work involved in a symbolic regression search, and is closely related to Koza's concepts of computational effort (Koza 1992).

1.2 A Tutorial on Abstract Expression Grammars

In standard Koza-style tree-based Genetic Programming (Koza 1992) the genome and the individual are the same Lisp s-expression which is usually

illustrated as a tree. Of course the tree-view of an s-expression is a visual aid, since a Lisp s-expression is normally a **list** which is a special Lisp data structure. Without altering or restricting standard tree-based GP in any way, we can view the individuals not as trees but instead as s-expressions.

- (*S1*) **depth 0 binary tree s-exp**: 3.45
- (*S2*) **depth 1 binary tree s-exp**: (+ x2 3.45)
- (*S3*) **depth 2 binary tree s-exp**: (/ (+ x2 3.45) (* x0 x2))
- (*S3*) **depth 2 irregular tree s-exp**: (/ (+ x2 3.45) 2.0)

In standard GP, applied to symbolic regression, the non-terminal nodes are all operators (*implemented as Lisp function calls*), and the terminal nodes are always either real number constants or features. An important point to remember is that we are not making a substantive change in standard GP. We are simply refocusing our view of the individuals as s-expressions.

The maximum depth of a GP individual is limited by the available computational resources; but, it is standard practice to limit the maximum depth of a GP individual to some managable limit at the start of a symbolic regression run.

Given any selected maximum depth k, it is an easy process to construct a maximal binary tree s-espression $\mathbf{U}_k$, which can be produced by the GP system without violating the selected maximum depth limit. As long as we are reminded that each **f** represents a function node while each **t** represents a terminal node, the construction algorithm is simple and recursive as follows.

- $\mathbf{U}_0$: t
- $\mathbf{U}_1$: (f t t)
- $\mathbf{U}_2$: (f (f t t) (f t t))
- $\mathbf{U}_3$: (f (f (f t t) (f t t)) (f (f t t) (f t t)))
- $\mathbf{U}_k$: (f U_{k-1} U_{k-1})

Any individual produced by the standard GP system may be smaller than, may be irregular, but will not be larger than $\mathbf{U}_k$.

For the purposes of this tutorial section, we shall select the maximum depth limit to be $k = 3$. We will only allow two features **x0 x1** and the IEEE double real number constants in each terminal node. Our set of valid functions will be the binary operators: **+ - *** and /. Furthermore, to make node identification easier, we shall enumerate each node in $\mathbf{U}_3$, and revert to more easily read functional notation as follows.

- $\mathbf{U}_3$: f0(f1(f2(t0,t1),f3(t2,t3)),f4(f5(t4,t5),f6(t6,t7)))

An example of a individual, which fits the form of U_3 is as follows.

- (*I1*): (* (/ (- x0 3.45) (+ x0 x1)) (/ (- x1 1.31) (* x0 x1)))

Up until this point we have not altered or restricted standard GP in any way; but, now we are about to make a slight alteration.

It is a little difficult, in (I1), to tell which ***** operator is associated with which function node in $\mathbf{U}_3$, so we are going to add an *annotation* to the individual (I1). We are going to restore the enumerated function nodes in $\mathbf{U}_3$, and we are going to add a function chromosome vector to (I1). The individual (I1) will now have two components: an s-expression and a function chromosome as follows.

- ($I1_d$,**s-exp**): (**f0** (**f1** (**f2** x0 3.45) (**f3** x0 x1)) (**f4** (**f5** x1 1.31) (**f6** x0 x1)))
- ($I1_d$,**f-chrome**): (* / - + / - *)

In order to make (I1) evaluate as it used to, we will need to make one additional slight alteration in the definition of the enumerated function nodes. From now on the function notation **fn** will be evaluated as: call the *nth* function found in the function chromosome vector. Hence, in (I1), **f0** will call *****, **f1** will call **/**, **f2** will call **-**, etc. We still have not restricted standard GP in any way. The same population operators work in the same way they always worked. However, we have added a new annotation to each individual and added a level of indirection to each function evaluation.

In the same vein we will add a constant chromosome vector for each constant reference in $\mathbf{U}_3$, so that the individual (I1) now has a new *annotation*, and each abstract constant reference **cn** evaluates to: return the *nth* constant found in the constant chromosome vector. The individual (I1) will now have three components: an s-expression, a function chromosome, and a constant chromosome as follows.

- ($I1_c$,**s-exp**): (f0 (f1 (f2 x0 **c1**) (f3 x0 x1)) (f4 (f5 x1 **c5**) (f6 x0 x1)))
- ($I1_c$,**f-chrome**): (* / - + / - *)
- ($I1_c$,**c-chrome**): (0 3.45 0 0 1.31 0 0)

Also in the same vein we will add a variable chromosome vector for each feature reference in $\mathbf{U}_3$, so that the individual (I1) now has a new *annotation*, and each abstract feature reference **vn** evaluates to: return the *nth* feature found in the feature chromosome vector. The individual (I1) will now have four components: an s-expression, a function chromosome, a constant chromosome, and a variable chromosome as follows.

- ($I1_b$,**s-exp**): (f0 (f1 (f2 **v0** c1) (f3 **v2** **v3**)) (f4 (f5 **v4** c5) (f6 **v6** **v7**)))
- ($I1_b$,**f-chrome**): (* / - + / - *)
- ($I1_b$,**c-chrome**): (0 3.45 0 0 1.31 0 0 0)
- ($I1_b$,**v-chrome**): (x0 x0 x0 x1 x1 x0 x0 x1)

Finally in the same vein we will add a term chromosome vector for each term reference in $\mathbf{U}_3$, so that the individual (I1) now has yet another a new *annotation*, and each abstract term reference **tn** evaluates to: examine the *nth* value found in the term chromosome vector. If the term value is 0, then return the *nth* value in the variable chromosome; otherwise, return the *nth* value in the constant chromosome. The individual (I1) will now have four chromosomes: a function chromosome, a constant chromosome, a variable chromosome and a term chromosome; and a standard abstract functional-expression. Plus the original concrete s-expression will be retained as follows.

- ($I1_a$,**f-exp**): f0(f1(f2(**t0**,**t1**),f3(**t2**,**t3**)),f4(f5(**t4**,**t5**),f6(**t6**,**t7**)))
- ($I1_a$,**f-chrome**): (* / - + / - *)
- ($I1_b$,**c-chrome**): (0 3.45 0 0 1.31 0 0 0)
- ($I1_b$,**v-chrome**): (x0 x0 x0 x1 x1 x0 x0 x1)
- ($I1_a$,**t-chrome**): (0 1 0 0 0 1 0 0)
- ($I1_a$,**concrete**): (* (/ (- x0 3.45) (+ x0 x1)) (/ (- x1 1.31) (* x0 x1)))

At this point in the tutorial take a brief pause. Examine the final abstract annotated version ($I1_a$) above and compare it to the original concrete version (I1). Walk through the evaluation process for each version. Satisfy yourself that the concrete s-expression (I1) and the abstract annotated ($I1_a$) both evaluate to exactly the same interim and final values.

We have made no restrictive nor destructive changes in the orginal individual (I1). Slightly altered to handle the new indirect references and the new chromosome annotations, any standard GP system will behave as it did before. Prove it to yourself this way. Take the annotated individual ($I1_a$), and replace each indirect reference with the proper value from the proper chromosome. This converts the abstract annotated ($I1_a$) back into the concrete s-expression (I1). Let your standard GP system operate on (I1) any way it wishes to produce a new individual (I2). Now convert (I2) back into an abstract annotated ($I2_a$) with the same process we used to annotate (I1).

We make three more slight alterations and enhancements in order to complete this tutorial on Abstract Expression Grammars (*AEGs*). First, we slightly alter the evaluation process for any uniary functions (*such as* **cos log** *etc.*) such that any excess arguments are simply ignored. Second, we add a special uniary operator (**lpass**) which simply returns its first argument (left-most) and, of course, ignores any excess arguments. Our valid functions are now: **+ - * /** and **lpass**. Third, we no longer think of evaluating ($I1_a$) the way we evaluate (I1). Instead we borrow important grammar template concepts from (O'Neill 2003) (Mackay 2010) and refocus our view of ($I1_a$) as an abstract expression grammar wherein the f-expression and chromosomes are evaluated as grammar rules and used to produced the concrete annotation

($I1_a$,**concrete**). Any changes in the chromosomes cause the individual to produced a new appropriate concrete annotation matching the new chromosome contents.

In summary we have simply added extra information as annotations and we've even kept the original s-expression. The standard GP system has not been restricted or limited in any way and may continue to operate much as it did before - albeit with these extra annotations. We've added an abstract functional-expression containing indirect: function references *fn*, constant references *cn*, variable references *vn*, and term references *tn*. We've also added annotations for four chromosomes for: functions, constants, variables, and terms.

Furthermore, given a selected maximum depth limit k, our **lpass** grammar rule allows the maximal binary tree s-expression $\mathbf{U}_k$ to generate ANY s-epression which the original GP system can generate *up to the selected depth limit*. Clearly the ($I1_a$) individual, shown above, can express any maximal 3-deep s-expression with appropriate changes to its chromosomes. But, what about smaller and irregular s-expressions which are not the same maximal shape as $\mathbf{U}_k$? The **lpass** grammar rule allows all smaller and irregular s-expressions to be generated as in the examples below.

- ($I2_a$,**f-exp**): f0(f1(f2(t0,t1),f3(t2,t3)),f4(f5(t4,t5),f6(t6,t7)))
- ($I2_a$,**f-chrome**): (lpass lpass lpass lpass lpass lpass lpass)
- ($I2_b$,**c-chrome**): (0 0 0 0 0 0 0 0)
- ($I2_b$,**v-chrome**): (x0 x0 x0 x0 x0 x0 x0 x0)
- ($I2_a$,**t-chrome**): (0 0 0 0 0 0 0 0)
- ($I2_a$,**concrete**): x0

- ($I3_a$,**f-exp**): f0(f1(f2(t0,t1),f3(t2,t3)),f4(f5(t4,t5),f6(t6,t7)))
- ($I3_a$,**f-chrome**): (lpass / * lpass lpass lpass lpass)
- ($I3_b$,**c-chrome**): (0 4.2 0 0 0 0 0 0)
- ($I3_b$,**v-chrome**): (x0 x0 x1 x0 x0 x0 x0 x0)
- ($I3_a$,**t-chrome**): (0 1 0 0 0 0 0 0)
- ($I3_a$,**concrete**): (/ (* x0 4.2) x1)

Because of this property we speak of $\mathbf{U}_k$ as a *universal* Abstract Expression to depth k.

Space restrictions prevent us from expanding on this brief tutorial. The basics of Abstract Expression Grammars and Universal Abstract Expressions are described in (Korns 2010). Reviewing Abstract Expression Grammars is strongly advised before continuing with this chapter.

With all this new information, we have a number of new options. We can keep our GP system operating as it did before; or, now that we have all the

constants in one vector, we can apply swarm operators to the constants for better constant mangement. Similarly we can apply genetic algorithm operations to the function, variable, and term vectors. If we decide to upgrade to a more complicated context sensitive grammar, many standard GP population operators become much easier with extra annotations. Finally when users ask for constraints on the search output and search process, user constraints become much easier with added annotations.

1.3 A Simple Symbolic Regression Grammar

In our symbolic regression research with financial data we set our maximum depth limit quite high. We allow many basis functions, and we allow many data features. However, for this chapter, we will set our maximum depth limit to only three and will allow only one basis function. Our purpose is to demonstrate that absolute accuracy issues arise quite quickly with current published symbolic regression techniques - even with simple problems. One would never dream of attacking an industrial regression problem with maximum depth set to three and only one basis function; but, that makes the appearance of absolute accuracy problems even more disappointing.

In this chapter we will create a series of simple test cases - all of which have absolute solutions within a maximum s-expression depth of three, only five features, and all with only one basis function. Theoretically each of our test cases should be easily solved by any GP symbolic regression system - even when set to these low depth and basis function limits. Unfortunately, we discovered that our ARC symbolic regression system could not solve ALL of these simple test cases.

Our simple symbolic regression grammar has the following basic elements.

- **Real Numbers**: 3.45, -.0982, 100.389, and *all other real constants.*
- **Features**: x0, x1, x2, x3, and x4 *a maximum of five features.*
- **Binary Operators**: +, -, *, /
- **Unary Operators**: sqrt, square, cube, cos, sin, tan, tanh, log, exp
- **lpass Operator**: lpass(expr,expr) *returns the left expression unaltered*
- **rpass Operator**: rpass(expr,expr) *expr returns the right expression unaltered*

Our concrete numeric expressions are C-like expressions containing the elements shown above and ready for compilation. All test cases are solved with univariate regression because, in this chapter, we limit our study to one basis function intractable problems. Our **lpass** and **rpass** operators are idempotents which allow our grammar to be used with AEG universal abstract expressions (*also all unary operators, should they receive two arguments, operate on the left expression and ignore the right expression*). Our basic expression

grammar is functional in nature, therefore all operators are viewed grammatically as function calls. The final regression champion will be the compilation of a basic concrete expression such as:

- (*E1*): f = (log(x3)/sin(x2*45.3))

In order to be considered *accurate* the NLSE of the regression champion, on the testing data (see Testing Regimen), must be .0001 or below.

The *universal* abstract expression $\mathbf{U}_3$, used for all test cases, consists of one basis function with a maximum depth of three function applications, as follows.

- (*E2*): f0(f1(f2(t0,t1),f3(t2,t3)),f4(f5(t4,t5),f6(t6,t7)))

By way of a review example if, in the AEG (E2), we replace f0 with **cos**, f1 with **lpass**, f2 with *****, t0 with **3.45**, and t1 with **x2**, then (E2) generates the following WFF candidate with the following concrete annotation ***cos(3.45*x2)*** ready for compilation.

- (*E3*$_a$,**f-exp**): f0(f1(f2(t0,t1),f3(t2,t3)),f4(f5(t4,t5),f6(t6,t7)))
- (*E3*$_a$,**f-chrome**): (cos lpass * lpass lpass lpass lpass)
- (*E3*$_b$,**c-chrome**): (3.45 0 0 0 0 0 0 0)
- (*E3*$_b$,**v-chrome**): (x0 x2 x0 x0 x0 x0 x0 x0)
- (*E3*$_a$,**t-chrome**): (1 0 0 0 0 0 0 0)
- (*E3*$_a$,**concrete**): **cos(3.45*x2)**

Even though our test cases are as limited as they are (*we have purposely limited them in this chapter so we can focus on absolute accuracy issues*), the size of the search space is quite large. This is one of the main issues faced by symbolic regression systems. Even simple test problems can have very large search spaces.

Our universal abstract expression $\mathbf{U}_3$ has seven abstract function placeholders. Each abstract function placeholder may hold one of fifteen function values (*4 binary operators, 9 uniary operators,* **lpass**, *and* **rpass**). Our universal abstract expression also has eight abstract term placeholders. Each term placeholder may hold one of 2^{64} IEEE double constant values or one of the five features. On test cases with five data features the size of the search space is 10^{162}. The presence of constants does make the search space much larger; however, even if we do not allow constants, thus limiting each term to features only, the size of the search space is still 10^{14}.

By way of comparison, the number of quarks in our universe is estimated to be roughly in the neighborhood of 10^{80}. The age of our universe in nanoseconds is estimated to be roughly in the neighborhood of 10^{23}, and the age of our universe in seconds is estimated to be roughly in the neighborhood of 10^{14}.

So theoretically, if we could evaluate each individual in a second, we might be able to exhaustively search the restricted space (*with no constants allowed*); but, it would require approximately the age of the universe to complete the exhaustive search.

1.4 ARC System Architecture

The ARC system is now restructured into three nested collections (*pools*). The first collection is the pool of islands. There may be more than one population island and the number of population islands may grow or wane during an evolutionary run (the search). Inside each island is the second collection: a survivor pool. In each population island there may be more than one surviving individual (WFF) and, in each island, the number of individuals may grow or wane during the search. Finally, inside each individual is the third collection: a swarm constant pool. The number of constants in each swarm constant pool, inside each individual, may grow or wane during the search.

By way of further explanation, let me point out that it is often the case that several individuals, in an island population, will be identical in every way except the values of their real number constants as shown below with (E4) and (E6). Notice how the survivor pool is over-supplied with two individuals (which are essentially the same WFF - except for constant values); and, it tends to get much worse with entire survivor pools exclusively dominated by essentially the same WFF forms differing only in constant values.

- (*E4*): cos(3.45*x2)
- (*E5*): sin(x3)/exp(x1*4.5)
- (*E6*): cos(-5.1*x2)
- (*E7*): sqrt(x4/x3)-exp(-log(x2))

ARC thinks of individuals (E4) and (E6) as *constant homeomorphs* and automatically combines them into an *abstract constant homeomorph*. All concrete real number constant references are replaced with abstract constant references, and the appropriate constant vectors are stored in a constant pool inside the individual as shown below.

- (*E8*): cos(**c0***x2), **constant pool** = *(3.45, -5.1)*
- (*E5*): sin(x3)/exp(x1*4.5)
- (*E7*): sqrt(x4/x3)-exp(-log(x2))

The ARC system supports operations on each of its nested collections. The swarm operators, such as particle swarm (Eberhart 2001) and differential evolution (Price 2005), act upon the constant pools inside each individual. The population operators, such as mutation and crossover (Man 1999), act upon the individuals within each island population. The island operators act upon the islands themselves.

By way of further explanation, the ARC island operators (*breeders*) are algorithms, such as *Age Layered Population Structures* (Hornby 2006) and *Age-Fitness Pareto* (Schmidt 2010), which govern the sorting, pruning, and synchronizing of fitness and survival within each island population plus any migration between islands. ARC currently supports several island operators. The *epoch* breeder initially fills its empty island with a large number of random individuals then does nothing unless the population grows stagnant at which time it kills off every individual and begins again. The *age-fitness* breeder implements a version of the age-fitness pareto algorithm explained in (Schmidt 2010). The *alps* breeder implements a version of the Age-Layered Population Structure algorithm explained in (Hornby 2006) altered to operate inside a single island. The *weed* breeder initially fills its empty island with a large number of random individuals then does nothing unless the population grows stagnant at which time it kills off every individual *over a specified age*. All of these island operators introduce a small number of random individuals at the start of each generation. Additionally, there is a special *national* island which records the best (most fit) individuals ever seen at any time during the search. The individuals collected in the national island are the answers which ARC provides the user at the termination of the search.

Finally, there is a specification language which guides the search process, states the goal formula, sets up disparate islands for breeding individuals, and specifies the island operators and search constraints.

In summary a bullet point list of the main architectural components of ARC is as follows

- *islands*: a flexible number of islands for breeding individuals
- *survivors*: a flexible number of individuals in each island
- *swarm*: a flexible number of constants inside each individual
- *swarm operators*: act on the constant pools inside each individual
- *population operators*: act on the individual individuals in each island
- *island operators*: act on the islands themselves
- *specification language*: guides the entire search process

1.5 Search Specification Language

In (Korns 2010) a number of important concepts are developed including: abstract expression grammars (AEG), universal abstract expressions, a new prototype search specification language inspired by the SQL/database-search relationship, and the concept of epigenome constraints to focus the search process in specific island populations. In this chapter, we will employ these concepts to help solve previously intractable problems.

Our universal abstract expression (E2), can be further constrained with where clauses. Each where clause sets up an independent island population, with a possibly further constrained epigenome, as follows.

- (*E9*): f0(f1(f2(t0,t1),f3(t2,t3)),f4(f5(t4,t5),f6(t6,t7)))
- (*E9.1*): island(alps,256,50)
- (*E9.2*): where island(alps,500,50)
- (*E9.3*): where f0(cos,sin,tan) v0(x1,x4) c1(1.0)
- (*E9.4*): where et() ec()

Each where clause is important both because of the choice of breeder and island population size, but also because the increased search focus and reduced search space size possible. For example, (E9.1) sets the default size of all islands to 256 individuals, 50 constants, and the breeder to Age-Layered Population Structure. Where clause (E9.2) overrides the defaults so that island one has 500 individuals, 50 constants, and the breeder to Age-Layered Population Structure and no constraints. The search space size is 10^{162}. Where clause (E9.3) constrains island two with f0 to the three trig functions, v0 to x1 and x4, and c1 to 1.0. All individuals in island two will be constrained as specified, and the search space size is reduced to 10^{159}. Where clause (E9.4) constrains island three to have no constants - effectively limiting search to features only. All individuals in island three will be constrained as specified, and the search space size is reduced to 10^{14}.

In this chapter, our approach to solving intractable problems will employ intelligent choices of multiple islands, breeders, and epigenome constraints.

1.6 Example Test Problems

In this chapter we list the example test problems which we will address. All of these test problems are no more than three grammar nodes deep (Note: in problem P10, *quart(x)* = x^4). All test problems reference no more than five input features. Some are easily solved with current Symbolic Regression techniques. Others are not so easily solved.

- (*P1*): y = 1.57 + (24.3*x3)
- (*P2*): y = 0.23 + (14.2*((x3+x1)/(3.0*x4)))
- (*P3*): y = -5.41 + (4.9*(((x3-x0)+(x1/x4))/(3*x4)))
- (*P4*): y = -2.3 + (0.13*sin(x2))
- (*P5*): y = 3.0 + (2.13*log(x4))
- (*P6*): y = 1.3 + (0.13*sqrt(x0))
- (*P7*): y = 213.80940889 - (213.80940889*exp(-0.54723748542*x0))
- (*P8*): y = 6.87 + (11*sqrt(7.23*x0*x3*x4))
- (*P9*): y = ((sqrt(x0)/log(x1))*(exp(x2)/square(x3)))
- (*P10*): y = 0.81 + (24.3*(((2.0*x1)+(3.0*square(x2)))/((4.0*cube(x3))+(5.0*quart(x4)))))
- (*P11*): y = 6.87 + (11*cos(7.23*x0*x0*x0))
- (*P12*): y = 2.0 - (2.1*(cos(9.8*x0)*sin(1.3*x4)))
- (*P13*): y = 32.0 - (3.0*((tan(x0)/tan(x1))*(tan(x2)/tan(x3))))
- (*P14*): y = 22.0 - (4.2*((cos(x0)-tan(x1))*(tanh(x2)/sin(x3))))

- (*P15*): y = 12.0 - (6.0*((tan(x0)/exp(x1))*(log(x2)-tan(x3))))

As a discipline, our goal is to demonstrate that *all* of the 10^{162} possible test problems can be solved after a reasonable number of individuals have been evaluated. This is especially true since we have limited these 10^{162} possible test problems to target functions which are univariate, reference no more than five input features, and which are no more than three grammar nodes deep. On the hopeful side, if the Symbolic Regression community can achieve a demonstration of absolute accuracy, then the same rigorous statistical inferences can follow a Symbolic Regression as now follow a Linear Regression, which would be a significant advancement in scientific technique.

As a base line for current state-of-the-art symbolic regression technique, we use aged layered population structure (ALPS) with an island population size of 256. If ARC runs all the test problems with a single ALPS island, we see that, at least with ARC, we are far from our accuracy goal. In fact we quickly demonstrate that there are large sets of test problems which are intractable with current state-of-the-art ALPS symbolic regression.

The base line search specification is as follows, and the search results are shown in Table 1.

- (*E10*): f0(f1(f2(t0,t1),f3(t2,t3)),f4(f5(t4,t5),f6(t6,t7)))
- (*E10.1*): where island(alps,256,50)

Table 1. Results with one ALPS island

Test	*WFFs*	*Train-NLSE*	*Train-TCE*	*Test-NLSE*	*Test-TCE*
P01	.14K	0.00	0.00	0.00	0.00
P02	.96K	0.00	0.00	0.00	0.00
P03	74.90K	0.00	0.00	0.00	0.00
P04	0.34K	0.00	0.00	0.00	0.00
P05	.94K	0.00	0.00	0.00	0.00
P06	.12K	0.00	0.00	0.00	0.00
P07	82.58K	0.00	0.00	0.00	0.00
P08	1.03K	0.00	0.00	0.00	0.00
P09	71.89K	0.01	0.00	0.97	0.37
P10	85.50K	0.83	0.39	1.00	0.49
P11	81.29K	0.99	0.46	1.00	0.49
P12	89.24	0.99	0.48	1.04	0.50
P13	85.98K	0.81	0.30	1.00	0.93
P14	83.49K	0.53	0.17	1.53	0.47
P15	1.27K	0.00	0.00	0.00	0.00

(*Note: the number of individuals evaluated before finding a solution is listed in the Well Formed Formulas (WFFs) column*)

A number of our test cases are solved very quickly with current state-of-the-art ALPS symbolic regression. Unfortunately, a number are not solved. Furthermore, these unsolved example test cases are representative of larger sets of intractable test problems within the 10^{162} possible target functions.

Throughout the remainder of this chapter, we will try a number of enhanced techniques to improve our search results on the example problems. These enhanced techniques will include:

- Employing a cloud of islands
- Employing an opening rule book
- Employing a closing rule book

1.7 Employing a Cloud Of Islands

Employing a cloud of islands is one obvious approach to solving more of our test problems. Of course, employing a cloud of islands will increase the total number of individuals evaluated; but, those additional individuals will be evaluated in parallel. With a cloud, we increase our computational effort by a factor of ten while making some progress on some previously intractable problems. Unfortunately we see that, at least with ARC, even with a cloud of islands, performance improves over the base line; but, we are still far from our accuracy goal. Furthermore we quickly demonstrate that, there remain large sets of intractable problems even with a cloud of ALPS islands.

The search specification is as follows, and the search results are shown in Table 2.

- (*E11*): f0(f1(f2(t0,t1),f3(t2,t3)),f4(f5(t4,t5),f6(t6,t7)))
- (*E11.1*): where island(alps,256,50)
- (*E11.2*): where island(alps,256,50)
- (*E11.3*): where island(alps,256,50)
- (*E11.4*): where island(alps,256,50)
- (*E11.5*): where island(alps,256,50)
- (*E11.6*): where island(alps,256,50)
- (*E11.7*): where island(alps,256,50)
- (*E11.8*): where island(alps,256,50)
- (*E11.9*): where island(alps,256,50)
- (*E11.10*): where island(alps,256,50)

Table 2. Results with ten ALPS island

Test	*WFFs*	*Train-NLSE*	*Train-TCE*	*Test-NLSE*	*Test-TCE*
P01	.15K	0.00	0.00	0.00	0.00
P02	3.26K	0.00	0.00	0.00	0.00
P03	804.49K	0.00	0.00	0.00	0.00
P04	.59K	0.00	0.00	0.00	0.00
P05	.25K	0.00	0.00	0.00	0.00
P06	.13K	0.00	0.00	0.00	0.00
P07	187.26K	0.00	0.00	0.00	0.00
P08	5.99K	0.00	0.00	0.00	0.00
P09	97.24K	0.00	0.00	0.00	0.00
P10	763.53K	0.01	0.02	0.99	0.29
P11	774.89K	0.99	0.47	1.00	0.48
P12	812.79K	0.99	0.47	1.04	0.51
P13	624.78K	0.00	0.00	0.00	0.00
P14	454.15K	0.00	0.00	0.00	0.00
P15	.045K	0.00	0.00	0.00	0.00

1.8 Employing an Opening Rule Book

Employing a cloud of islands with a well thought out opening book of search constraint rules is the next obvious approach to solving more of our test problems. Of course, employing an opening rule book will increase the total number of individuals evaluated; but, those additional individuals will be evaluated in parallel, and each island can be tailored, with constraint rules, to search its own limited area of expertise for difficult-to-learn patterns that are commonly encountered.

One very interesting aspect of our experimentation with opening books is that the resources in many of the islands were successfully concentrated. Notice that the default survivor population has been reduced from 256 to 10 and the default swarm pool size has been reduced from 50 to 25. Thus we use nine islands but our computational resources are much less than nine times those used by a single island general purpose ALPS approach.

Our **opening rule book** is as follows, and the search results, with our opening book, are shown in Table 3.

- (*E11*): f0(f1(f2(t0,t1),f3(t2,t3)),f4(f5(t4,t5),f6(t6,t7)))
- (*E11.1*): national(10,25)
- (*E11.2*): island(weed,10,25)
- // Opening Book
- (*E11.3*): where
- (*E11.4*): where ec() et()
- (*E11.5*): where f0(*) f1..6(lpass,*) ef(f1,f2,f3,f4,f5,f6) ec() et()
- (*E11.6*): where f0(*,square,sqrt,cube,cos,sin,tan,tanh,log,exp) f1..6(lpass,*)

- (*E11.7*): where op(lpass,rpass,*,/,cos,sin,tan,tanh) island(smart,gade,256,25,50)
- (*E11.8*): where f0(*,cos,sin,tan,tanh) f1..6(lpass,*)
- (*E11.9*): where f0(cos,sin,tan,tanh) ef(f0,f1,f2,f3) ev(v0,v1,v2,v3) ec(c0,c1,c2,c3) et(t0,t1,t2,t3)
- (*E11.10*): where f0,f2,f5(*) f1,f4(cos,sin,tan,tanh) ef(f1,f4) ev(v1,v5) ec(c0,c4) et() island(smart,256,25,10)
- (*E11.11*): where f0,f1,f4(+,-,*,/) f2,f3,f5,f6(cos,sin,tan,tanh) ev(v0,v2,v4,v6) ec() et() island(smart,256,25,10)

A brief explanation of the search constraint rules in our **Opening Book** is as follows. Rule (E11.1) tells ARC to apply population operators to the national island. Rule (E11.2) sets the default island using the **weed** breeder with 10 individual individuals and 25 constants in its swarm pool. Rule (E11.3) requests an unrestricted search island. Rule (E11.4) requests an unrestricted search island with no terms and no constants. Rule (E11.5) requests a search for all possible two, three, and four way cross correlations of features. Rule (E11.6) requests a search for all possible two, three, and four way cross correlations of features with a possible unary cap. Rule (E11.7) requests a search of only the restricted trigonometric operators specified. Rule (E11.8) requests a search for all possible two, three, and four way cross correlations of features with a possible trigonometric cap. Rule (E11.9) requests a search of only the restricted trigonometric operators. Rule (E11.10) requests a search of only the products of trigonometric operators. Rule (E11.11) requests a search of only the restricted operators specified.

Table 3. Results with an opening book

Test	*WFFs*	*Train-NLSE*	*Train-TCE*	*Test-NLSE*	*Test-TCE*
P01	.06K	0.00	0.00	0.00	0.00
P02	113K	0.00	0.00	0.00	0.00
P03	222.46K	0.00	0.00	0.00	0.00
P04	0.86K	0.00	0.00	0.00	0.00
P05	0.16K	0.00	0.00	0.00	0.00
P06	0.01K	0.00	0.00	0.00	0.00
P07	4.10K	0.00	0.00	0.00	0.00
P08	11.00K	0.00	0.00	0.00	0.00
P09	116.81K	0.00	0.00	0.00	0.00
P10	214.27K	0.83	0.45	1.00	0.46
P11	206.04K	0.99	0.47	1.00	0.49
P12	217.25K	0.99	0.47	1.00	0.49
P13	22.40K5	0.00	0.00	0.00	0.00
P14	9.54K	0.00	0.00	0.00	0.00
P15	1.99K	0.00	0.00	0.00	0.00

We arrived at this opening book of search constraint rules experimentally. It has been tailored to the behavior of the ARC system plus the chosen operators. It has been designed to capture a large number of commonly occurring test problems without increasing computation resources excessively. Creating an opening rule book is time consuming; but, it need only be created once for each SR system and each operator set. Employing an opening book moves one forward in solving a broad range of test problems with fewer individuals searched.

Unfortunately we see that, at least with ARC, even with an opening book we have still not achieved our accuracy goal. In fact employing an opening book did not significantly improve accuracy over the cloud of ALPS islands; but, it did reduce the computational resources required to arrive at the same accuracy.

One polynomial and two of the more complicated trigonometric test problems continue to resist solution. Largely this is because the sine and cosine function produce a wavy response surface which makes it very difficult for ARC to distinguish the local from the global minima.

It has been our experience that attempting to provide more islands and a much longer search time does not improve the situation. We have run these problems thorough several months of elapsed time and millions and billions of individuals evaluated with no significant improvement. When the search space is 10^{162}, with a very choppy surface, throwing more brute force resources does not seem to be helpful.

1.9 Employing a Closing Rule Book

Employing a cloud of islands with a well thought out opening and closing book of search constraint rules is our next approach to solving more of our test problems. We create a so called **smart** breeder which initially implements the same opening book strategy as shown in the previous section. however, **smart** maintains a set of closing search constraint rules ready to be used when the opening book search is not progressing well.

Smart constantly monitors each one of the opening book islands it has allocated. When an island search is not progressing to its satisfaction, smart can employ a search constraint rule from its *closing rule book*. The island is then co-opted to perform the search specified in the selected closing book search constraint rule.

Employing a cloud of islands with a well thought out **closing rule book** is yet another approach to solving more of our intractable test problems. In the case of a closing book, smart does not employ the closing book until the opening book rules have failed. This allows ARC to focus the individuals searched, in each affected island, toward problem areas which we know *a priori* to be

difficult. Adding a closing book will increase the individuals evaluated; but, each island can be tailored to search its own limited area of expertise for common difficult-to-learn patterns that are known to be problematic. Furthermore, without the closing book, even after billions of individuals searched there was no convergence on the intractable problems; however, with the closing book there was convergence on some of the problems after only millions of individuals searched.

The closing rule book is as follows, and the search results, with our closing book, are shown in Table 4.

- (*E12*): f0(f1(f2(t0,t1),f3(t2,t3)),f4(f5(t4,t5),f6(t6,t7)))
- (*E12.1*): national(10,25)
- (*E12.2*): island(smart,10,25)
- // Opening Book
- (*E12.3*): where
- (*E12.4*): where ec() et()
- (*E12.5*): where f0(*) f1..6(lpass,*) ef(f1,f2,f3,f4,f5,f6) ec() et()
- (*E12.6*): where f0(*,square,sqrt,cube,cos,sin,tan,tanh,log,exp) f1..6(lpass,*)
- (*E12.7*): where op(lpass,rpass,*,/,cos,sin,tan,tanh) island(smart,gade,256,25,50)
- (*E12.8*): where f0(*,cos,sin,tan,tanh) f1..6(lpass,*)
- (*E12.9*): where f0(cos,sin,tan,tanh) ef(f0,f1,f2,f3) ev(v0,v1,v2,v3) ec(c0,c1,c2,c3) et(t0,t1,t2,t3)
- (*E12.10*): where f0,f2,f5(*) f1,f4(cos,sin,tan,tanh) ef(f1,f4) ev(v1,v5) ec(c0,c4) et() island(smart,256,25,10)
- (*E12.11*): where f0,f1,f4(+,-,*,/) f2,f3,f5,f6(cos,sin,tan,tanh) ev(v0,v2,v4,v6) ec() et() island(smart,256,25,10)
- // Closing Book
- (*E12.12*): where f0(cos,sin,tan,tanh) f1(*) f2(*) f3(*) ef(f0) ev(v0,v2,v3) ec(c1) et() island(smart,gade,1000,100,50,1000)
- (*E12.13*): where f0(cos,sin,tan,tanh) f1(*) f2(*) f3(lpass,*) ef(f0) ec(c1) et() eb(b0) island(smart,gade,100,100,50,100)
- (*E12.14*): where f0(lpass,*,/) f1(cos,sin,tan,tanh) f2(*) f4(cos,sin,tan,tanh) f5(*) ef(f0,f1,f4) ev(v1,v5) ec(c0,c4) et() island(weed,gade,1000,100,50,1000)
- (*E12.15*): where f0(+,-,*,/) f1(+,-,*,/) f2(cos,sin,tan,tanh) f3(cos,sin,tan,tanh) f4(+,-,*,/) f5(cos,sin,tan,tanh) f6(cos,sin,tan,tanh) ev(v0,v2,v4,v6) ec() et() eb(b0) island(smart,gade,1000,100,100,1000)
- (*E12.16*): where f0(lpass,*,+,/) f1(lpass,+) f2(*,psqrt,psquare,pcube,pquart) f3(*,psqrt,psquare,pcube,pquart) f4(lpass,+) f5(*,psqrt,psquare,pcube,pquart) f6(*,psqrt,psquare,pcube,pquart) ec(c0,c2,c4,c6) ev(v1,v3,v5,v7) et() eb(b0) island(smart,gade,1000,100,100,100)
- (*E12.17*): where op(lpass,rpass,+,-,*,/) island(smart,gade,1000,256,200,1000)

A brief explanation of the search constraint rules in our **Closing Book** is as follows. Rule (E12.12) tells ARC to look for three way cross correlations capped with any of the trigonometric functions. Rule (E12.13) requests

a search for all possible two way cross correlations capped with any of the trigonometric functions. Rule (E12.14) requests a search for all possible constants times variables capped with any of the trigonometric functions. Rule (E12.15) requests a search for all possible products, sums, ratios, and differences of trigonometric singleton products or ratios. Rule (E12.16) requests a search for all possible products, sums, and ratios of polynomials. Rule (E12.17) requests a search for all possible arithmetic functions.

Table 4. Results with a closing book

Test	*WFFs*	*Train-NLSE*	*Train-TCE*	*Test-NLSE*	*Test-TCE*
P01	.06K	0.00	0.00	0.00	0.00
P02	113K	0.00	0.00	0.00	0.00
P03	222.46K	0.00	0.00	0.00	0.00
P04	0.86K	0.00	0.00	0.00	0.00
P05	0.16K	0.00	0.00	0.00	0.00
P06	0.01K	0.00	0.00	0.00	0.00
P07	4.10K	0.00	0.00	0.00	0.00
P08	11.00K	0.00	0.00	0.00	0.00
P09	116.81K	0.00	0.00	0.00	0.00
P10	1.34M	0.00	0.00	0.00	0.00
P11	4.7M	0.00	0.00	0.00	0.00
P12	16.7M	0.99	0.47	1.00	0.49
P13	22.40K	0.00	0.00	0.00	0.00
P14	9.54K	0.00	0.00	0.00	0.00
P15	1.99K	0.00	0.00	0.00	0.00

We arrived at this closing rule book experimentally. It has been tailored to the behavior of the ARC system plus the chosen operators. It has been designed to capture a large number of commonly occurring test problems without increasing computation resources excessively. Creating a closing book is time consuming; but, it need only be created once for each SR system and each operator set. Employing a closing book moves us one step forward toward solving a broader range of more difficult test problems.

We see that, at least with ARC, with a closing rule book we have achieved additional improvement with respect to these chosen test problems. Unfortunately, we are far from asserting that ARC is absolutely accurate on *ALL* possible test problems of one basis function and three grammar nodes in depth. In fact, from copious end user experimentation, we know that ARC, even with the current enhancements, cannot solve all of the possible 10^{162} test problems. The problem space is so large that end users routinely uncover problems which return incorrect answers or fail to find even an acceptable champion.

The next step is clearly to find credible answers to some of the current mysteries. Is it possible to identify the larger areas of more complicated intractable problems within the 10^{162} search space? Are these problems isolated all over the 10^{162} search space, or are they clustered together in sets amenable to closing book island search specifications? How many identifiable intractable problems clusters are there? What are the best approaches for solving each identified problems set? Will each vendor be required to develop opening and closing books specific to each of their systems and chosen grammars, or will some, as of now unknown, automated method be discovered such as in (Ryan 2005) or as in (Spector 2010)?

Once we have found answers to the many current mysteries, verification will be the next serious challenge. Is there some internationally acceptable way to declare victory given the fact that we cannot hope to test all possible 10^{162} problems? The best situation would be to have the independent scientific community supportive of SR claims. The worst situation would be to have the independent scientific community skeptical of SR claims. What are the necessary conditions for even hazarding an assertion that SR is absolutely accurate on ALL possible test problems of one basis function and three grammar nodes in depth?

2 Conclusion

The use of abstract grammars in symbolic regression provides the end user with fine tuned control of the search process. Joint projects with end users have pointed out that user control of both the search process and the fitness measure will be essential for many applications. The use of opening and closing books, and multiple island intelligent breeding with epigenome constraints, moves the entire discipline much closer to *industrial ready* for many applications.

Nevertheless, state-of-the-art Symbolic Regression techniques continue to suffer poor accuracy on very large categories of test problems even under the most favorable minimalist assumptions.

The opportunity is unprecedented. If the symbolic regression community is able to offer accuracy, even within the favorable minimalist assumptions of this chapter, and if that accuracy is vetted or confirmed by an independent body (distinct from the SR community), then symbolic regression will realize its true potential. SR could be yet another machine learning technique (such as linear regression, support vector machines, etc.) to offer a foundation from which hard statistical assertions can be launched. Furthermore, we would finally have realized the original dream of returning not just accurate coefficients but accurate formulas as well.

The challenge is significant. It is unlikey that any single research team, working alone, will be sufficient to meet the challenge. We will have to band

together as a community, developing standardized test problems, and standardized grammars. More importantly we will have to reach out to the theoreticians in our GP discipline and in the mathematical and statistical communities to establish some body of conditions whereby independent communities will be willing to undertake the task of confirming SR accuracy. And, of course, first we, in the SR community, will have to work together to achieve such accuracy.

It is not clear that opening and closing books are the final solution to our accuracy problem. The most desirable situation would be to discover some automated self-adaptive method.

What is clear is that if we, in the symbolic regression community, wish to continue making the claim that we return *accurate formulas*; and, if we wish to win the respect of other academic disciplines, then we will have to solve our accuracy issues.

References

1. Flor Castillo, Arthur Kordon, and Carlos Villa (2010). Genetic Programming Transforms in Linear Regression Situations, in *Genetic Programming Theory and Practice VIII*. Springer, New York.
2. Russell Eberhart, Yuhui Shi, James Kennedy (2001). Swarm Intelligence. Morgan Kaufman, New York.
3. Gregory S Hornby (2006). Age-Layered Population Structure For reducing the Problem of Premature Convergence, in *GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation*. ACM Press, New York.
4. Michael Korns (2009). Symbolic Regression of conditional target expressions, in *Genetic Programming Theory and Practice VII*. Springer, New York.
5. Michael Korns (2010). Abstract Expression Grammar Symbolic Regression, in *Genetic Programming Theory and Practice VIII*. Springer, New York.
6. John R Koza (1992). Genetic Programming: On the Programming of Computers by Means of Natural Selection. The MIT Press, Cambridge Massachusetts.
7. John R Koza (1994). Genetic Programming II: Automatic Discovery of Reusable Programs. The MIT Press, Cambridge Massachusetts.
8. John R Koza, Forrest H Bennett III, David Andre, Martin A Keane (1999). Genetic Programming III: Darwinian Invention and Problem Solving. Morgan Kaufmann Publishers, San Francisco California.
9. Robert Mckay, Nguyen Hoai, Peter Whigham, Yin Shan, Michael O'Neill (2010). *Grammar-based Genetic Programming: a survey*. Genetic Programming and Evolvable Machines archive, Volume 11 Issue 3-4, September 2010, Kluwer Academic Publishers Hingham, MA, USA
10. Trent McConaghy, Pieter Palmers, Gao Peng, Michiel Steyaert, Georges Gielen (2009). *Variation-Aware Analog Structural Synthesis: A Computational Intelligence Approach*. Springer, New York.
11. Michael O'Neill, Conor Ryan (2003). Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language. Kluwer Academic Publishers, Dordrecht Netherlands.
12. Kim-Fung Man, Kit-Sang Tang, Sam Kwong (1999). Genetic Algorithms. Springer, New York.
13. Kenneth Price, Rainer Storn, Jouni Lampinen (2005). Differential Evolution: A Practical Approach to Global Optimization. Springer, New York.
14. Conor Ryan, Maarten Keijzer, and Mike Cattolico (2005). Favourable Biasing of Function Sets using Run Transferable Libraries, in *Genetic Programming Theory and Practice II*. Springer, New York.
15. Michael Schmidt, Hod Lipsone (2010). Age-Fitness Pareto Optimization, in *Genetic Programming Theory and Practice VI*. Springer, New York.
16. Guido Smits, Ekaterina Vladislavleva, and Mark Kotanchek (2010). Scalable Symbolic Regression by Continuous Evolution with Very Small Populations, in *Genetic Programming Theory and Practice VIII*. Springer, New York.
17. Lee Spector (2010). Towards Practical Autoconstructive Evolution: Self-Evolution of Problem-Solving Genetic Programming Systems, in *Genetic Programming Theory and Practice VIII*. Springer, New York.