Predicting Corporate Forward 12 Month Earnings

Michael F. Korns
The Shang Grand Tower U17E

98 Perea Street,
Legazpi Village, Makati 1229
Manila, Philippines
+639178171756

mkorns@korns.com

ABSTRACT

Valuation of securities via their forward 12 month price earnings ratio (*ftmPE*) is a very common securities valuation method in the industry. Obviously the *ftmPE* valuation depends heavily on the estimate of forward 12 month corporate earnings per share (*ftmEPS*). Obvious inputs to the *ftmEPS* prediction process are the past earnings time series plus one or more analyst predictions. Previously only linear regression and linear classification and regression trees (CART) have been available as techniques for analyzing these inputs. Nonlinear symbolic regression (SR) has not been used because of SR's difficulties optimizing imbedded constants. However, recent integrations of swarm intelligence (SI) with symbolic regression support a level of maturity and sophistication making nonlinear regression and nonlinear CART available for real world financial applications.

Automated *ftmEPS* prediction involving the analysis of many securities, often involves multiple training regressions each on hundreds of thousands of training examples, plus there is always a timeliness issue, so analytic tools must be strong and thoroughly matured. Symbolic regression systems incorporating only genetic programming are shown to be inadequate for optimizing imbedded constants; but, symbolic regression systems integrating swarm intelligence with genetic programming are shown to be quite effective.

This paper examines several different well known swarm intelligence algorithms as integrated components in an enhanced symbolic regression system. Each swarm algorithm is integrated into the symbolic regression system, to predict the forward 12 month earnings of approximately 1,500 companies over a twenty year period (1990 thru 2009). Utilizing both classification and regression scores in the training and testing periods, each swarm algorithm is analyzed for efficacy. Finally, all of the swarm algorithms are allowed to compete simultaneously, in multiple islands, to predict *ftmEPS*. Again utilizing both classification and regression scores in the training and testing periods, this competitive approach is compared with the best individual swarm algorithm. The goal is to aid in the development of a robust, mature, symbolic regression system.

Keywords

Value investing, symbolic regression, swarm intelligence, genetic programming, nonlinear regression, CART, particle swarm, differential evolution, bees algorithm.

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 Mathematica (www.evolved-analytics.com). There is at least one very well done open source symbolic regression package available for free down load (http://ccsl.mae.cornell.edu/eureqa).
In addition to our own ARC system [6], currently used internally for massive financial data nonlinear regressions, there are a number of other mature symbolic regression packages currently used in industry including [8] and [9]. Plus there is an interesting work in progress by McConaghy [10].

Nonlinear symbolic regression (SR) has not been widely applied to financial problems because of SR's difficulties optimizing imbedded constants. Optimizing imbedded constants is often a critical requirement in many financial applications. However, recent integrations of swarm intelligence (SI) with symbolic regression support a level of maturity and sophistication making nonlinear regression and nonlinear CART available for real world financial applications.

In this chapter we investigate the integration of two popular swarm intelligence algorithms (Bees, and Particle Swarm), and one popular evolutionary computation algorithm (Differential Evolution) with standard genetic programming symbolic regression to help optimize imbedded constants in a real world financial application: the prediction of forward 12 month earnings per share. We make the observations: that standard genetic programming does not optimize imbedded constants well; that swarm intelligence algorithms are adept at optimizing constants; and that allowing imbedded constants in SR greatly increases the size of the search space.

In the body of the chapter it is shown that the differences between the three popular constant managing algorithms is minimal for optimizing imbedded constants; yet without any swarm intelligence standard GP symbolic regression fails to optimize imbedded constants effectively.

We proceed with a general introduction to symbolic regression and the size of the search space.

Symbolic Regression is an approach to general nonlinear regression which is the subject of many scholarly articles in the Genetic Programming community. A broad generalization of general nonlinear regression is embodied as the class of *Generalized Linear Models* (GLMs) as described in [11]. A GLM is a linear combination of **I** basis functions $\mathbf{B_i}$; i = 1,2, … I, a dependent variable **y**, and an independent data point with **M** features **x** = <x_1, x_2, x_3, …x_m>: such that

1 $y = \gamma(x) = c_0 + \sum_{i=1}^{I} c_i B_i(x) + err$

As a broad generalization, GLMs can represent any possible nonlinear formula. However the format of the GLM makes it amenable to existing linear regression theory and tools since the GLM model is linear on each of the basis functions $\mathbf{B_i}$.

For a given vector of dependent variables, Y, and a vector of independent data points, X, symbolic regression will search for a set of basis functions and coefficients which minimize *err*. In [12] the basis functions selected by symbolic regression will be formulas as in the following examples:

2 $B_1 = x_3$
3 $B_2 = x_1+x_4$
4 $B_3 = \text{sqrt}(x_2)/\tan(x_5/4.56)$
5 $B_4 = \tanh(\cos(x_2*.2)*\text{cube}(x_5+\text{abs}(x_1)))$

If we are minimizing the least squared error, ***LSE***, once a suitable set of basis functions {**B**} have been selected, we can discover the proper set of coefficients {**C**} deterministically using standard univariate or multivariate regression. The value of the GLM model is that one can use standard regression techniques and theory. Viewing the problem in this fashion, we gain an important insight. Symbolic regression does not add anything to the standard techniques of regression. The value added by symbolic regression lies in its abilities as a search technique: how quickly and how accurately can SR find an optimal set of basis functions {**B**}.

The immense size of the search space provides ample need for improved search techniques In standard Koza-style tree-based Genetic Programming [12] 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 such as this depth 2 binary tree s-exp: (/ (+ x_2 3.45) (* x_0 x_2)), or this depth 2 irregular tree s-exp: (/ (+ x_2 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. 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 manageable 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-expression 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.

U_0: t
U_1: (f t t)
U_2: (f (f t t) (f t t))
U_3: (f (f (f t t) (f t t)) (f (f t t) (f t t)))
U_k: (f U_{k-1} U_{k-1})

Any basis function produced by the standard GP system will be represented by at least one element of U_k. In fact, U_k is isomorphic to the set of all possible basis functions generated by the standard GP system.

Given this formalism of the search space, it is easy to compute the size of the search space, and it is easy to see that the search space is huge even for rather simple basis functions. For our use in this chapter the function set will be the following functions: **F** = {**+ - * / abs sqrt square cube cos sin tan tanh log exp max min ℵ**} (where $\aleph(a,b) = \aleph(a) = a$). The terminal set is the features $\mathbf{x_0}$ thru $\mathbf{x_m}$ and the real constant **c**, which we shall consider to be 2^{64} in size. Where |**F**| = 17, **M**=20, and **k**=0 , the search space is S_0 = **M**+2^{64} = 20+2^{64} = 1.84x10^{19}. Where **k**=1, the search space is S_1 = |**F**|*S_0*S_0 = 5.78x10^{39}. Where **k**=2, the search space grows to S_2 = |**F**|*S_1*S_1 = 5.68x10^{80}. For **k**=3, the search space grows to S_3 = |**F**|*S_2*S_2 = 5.5x10^{162}. Finally if we allow three basis functions **B**=3 for financial applications, then the final size of the search space is S_3*S_3*S_3 = 5.5x10^{486}.

Clearly even for three simple basis functions, with only 20 features and very limited depth, the size of the search space is already very large; and, the presence of real constants accounts for a significant portion of that size. For instance, without real constants, S_0 = 20, S_3 = 1.054x10^{19}, and with **B**=3 the final size of the search space is 1.054x10^{57}. It is our contention that since real constants account for such a significant portion of the search space, symbolic regression would benefit from special constant evolutionary operations. Since standard GP does not offer such operations, we investigate the enhancement of symbolic regression with swarm intelligence algorithms specifically designed to evolve real constants.

As we apply our enhanced symbolic regression to an important real world investment finance application, the prediction of forward 12 month earnings per share, we discover a number of accuracy, believability, and regulatory issues which must be addressed. Solutions for those issues are provided and we proceed to apply an enhanced symbolic regression algorithm to the problem of estimating forward corporate earnings per share.

This chapter begins with a discussion of Symbolic Regression theory in Section (2) and with important theoretical issues in Section (3). Methodology is discussed in Section (4), then Sections (5) through (10) discuss the algorithms for Standard GP Symbolic Regression and the enhancements for merging swarm intelligence with standard GP symbolic regression. In Section (11) we compare the performance of standard GP symbolic regression with enhanced symbolic regression on a set of illustrative sample test problems. Sections (12) thru (15) give a background in investing and discuss the essential requirements for applying symbolic regression to predicting forward 12 month earnings in a real world financial setting. Finally, Sections (17) thru (19) compare the performance of enhanced symbolic regression with the swarm algorithm being Differential Evolution, the bees Algorithm, or Particle Swarm.

2. Symbolic Regression Theory

In standard Koza-style symbolic regression [12], a Lisp s-expression is manipulated via the evolutionary techniques of mutation and crossover to produce a new s-expression which can be tested, as a basis function candidate in a GLM. Basis function candidates that produce better fitting GLMs are promoted.

Mutation inserts a random s-expression in a random location in the starting s-expression. For example, mutating s-expression (4) we obtain s-expression (4.1) wherein the sub expression "tan" has been randomly replaced with the sub expression "**cube**". Similarly, mutating s-expression (5) we obtain s-expression (5.1) wherein the sub expression "cos(x_2*.2)" has been randomly replaced with the sub expression "**abs(x_2+ x_5)**".

4 B_3 = sqrt(x_2)/tan(x_5/4.56)
4.1 B_5 = cos(x_2)/***cube***(x_5/4.56)
5 B_4 = tanh(cos(x_2*.2)*cube(x_5+abs(x_1)))
5.1 B_6 = tanh(**abs(x_2+ x_5)***cube(x_5+abs(x_1)))

Crossover combines portions of a *mother* s-expression and a *father* s-expression to produce a *child* s-expression. Crossover inserts a randomly selected sub expression from the *father* into a randomly selected location in the *mother*. For example, crossing s-expression (5) with s-expression (4) we obtain *child* s-expression (5.2) wherein the sub expression "cos(x_2*.2)" has been randomly replaced with the sub expression "**tan(x_5/4.56)**". Similarly, again crossing s-expression (5) with s-expression (4) we obtain another *child* s-expression (5.3) wherein the sub expression "x_5+abs(x_1)" has been randomly replaced with the sub expression "**sqrt(x_2)**".

4 B_3 = sqrt(x_2)/tan(x_5/4.56)
5 B_4 = tanh(cos(x_2*.2)*cube(x_5+abs(x_1)))
5.2 B_7 = tanh(**tan(x_5/4.56)***cube(x_5+abs(x_1)))
5.3 B_8 = tanh(cos(x_2*.2)*cube(**sqrt(x_2)**))

These mutation and crossover operations are the main tools of standard GP, which functions as described in Algorithm 2, randomly creating a population of candidate basis functions, mutating and crossing over those basis functions repeatedly while consistently promoting the most fit basis functions. The winners being the collection of basis functions which receive the most favorable least square error in a GLM with standard regression techniques.

3. Theoretical Issue

A theoretical issue with standard GP symbolic regression is the poor optimization of embedded constants under the mutation and crossover operators. Notice that in basis functions (4) and (5) there are real constants embedded inside the formulas. These embedded constants, **4.56** and **.2**, are quite important. That is to say that basis function (4) behaves quite differently than basis function (4.2) while basis function (5) behaves quite differently than basis function (5.4).

4 B_3 = sqrt(x_2)/tan(x_5/**4.56**)
4.2 B_9 = sqrt(x_2)/tan(x_5)
5 B_4 = tanh(cos(x_2***.2**)*cube(x_5+abs(x_1)))
5.4 B_{10} = tanh(cos(x_2)*cube(x_5+abs(x_1)))

The behavior can be quite startling. For instance, if we generate a set of random independent variables for <x_1, x_2, x_3, …x_m> and we set the dependent variable, y = sqrt(x_2)/tan(x_5/**4.56**), then a regression on y = sqrt(x_2)/tan(x_5) returns a very bad LSE. In fact the bad regression fit continues until one regresses on y = sqrt(x_2)/tan(x_5/**4.5**). It is only until one regresses on y = sqrt(x_2)/tan(x_5/**4.55**) that we get a reasonable LSE with an R-Square of .56. Regressing on y = sqrt(x_2)/tan(x_5/**4.555**) achieves a better LSE with an R-Square of .74. Of course regressing on y = sqrt(x_2)/tan(x_5/**4.56**) returns a perfect LSE with an R-Square of 1.0.

Clearly, in many cases of embedded constants, there is a very small neighborhood, around the correct embedded constant, within which an acceptable LSE can be achieved.

In standard Koza-style symbolic regression [12], the mutation and crossover operators are quite cumbersome in optimizing constants. As standard GP offers no constant manipulation operators per se, the mutation and crossover operators must work doubly hard to optimize constants. For instance, the only way to optimize the embedded constant in s-expression (5) would be to have a series of mutations or crossovers which resulted in an s-expression with multiple iterative additions and subtractions as follows [12].

4 B_3 = sqrt(x_2)/tan(x_5/4.56)
4.2 B_3 = sqrt(x_2)/tan(x_5/(1.0+3.2))
4.3 B_3 = sqrt(x_2)/tan(x_5/((1.0+3.2)+.3))
4.4 B_3 = sqrt(x_2)/tan(x_5/(((1.0+3.2)+.3)+.07))
4.4 B_3 = sqrt(x_2)/tan(x_5/((((1.0+3.2)+.3)+.07)-.01))

Characteristically, the repeated mutation and crossover operations which finally realize an optimized embedded constant also greatly *bloat* the resulting basis function with byzantine operator sequences [18]. On the other hand swarm intelligence techniques are quite good at optimizing vectors of real numbers. So the challenge is how to collect the embedded constants found in a GP s-expression into a vector so they can be easily optimized by swarm intelligence techniques.

Recent advances in symbolic regression technology including Abstract Expression Grammars (*AEG*s) [3], [4], [5], [6], and [13] can be used to control bloat, specify complex search constraints, and expose the embedded constants in a basis function so they are available for manipulation by various swarm intelligence techniques suitable for the manipulation of real numeric values. This presents an opportunity to combine standard genetic programming techniques together with swarm intelligence techniques into a seamless, unified algorithm for pursuing symbolic regression.

The focus of this chapter will be an investigation of swarm intelligence techniques, used in connection with AEGs, which can improve the speed and accuracy of symbolic regression search, especially in cases where embedded numeric constants are an issue hindering performance.

4. Methodology

Our methodology is influenced by the practical issues in applying symbolic regression to a real world investment finance problem. First there is the issue that current standard GP symbolic regression cannot solve selected simple test problems required for the successful application of SR to predicting forward corporate earnings per share. This includes the methodological challenge of enhancing standard GP with swarm intelligence and modifying the necessary encodings to accommodate both GP and swarm intelligence algorithms. Second there is the issue of adapting symbolic regression to run in a real world financial application with massive amounts of data. Third there is the issue of modifying symbolic regression, as practiced in academia, to conform to the very difficult U.S. Securities Exchange Commission regulatory environment.

Sections (5) thru (10) discuss the methodological challenge of enhancing standard GP symbolic regression so that it can be effective when applied to the real world problem of predicting forward 12 month corporate earnings per share. In Section (11), the behavior of GP symbolic regression with and without the enhancement of swarm intelligence is compared on a few sample test problems.

For the sample test problems, we will use only statistical best practices out-of-sample testing methodology. A matrix of independent variables will be filled with random numbers. Then the model will be applied to produce the dependent variable. These steps will create the training data. A symbolic regression will be run on the training data to produce a champion *estimator*. Next a matrix of independent variables will be filled with random numbers. Then the model will be applied to produce the dependent variable. These steps will create the testing data. The *estimator* will be regressed against the testing data producing the final LSE and R-Square scores for comparison.

Sections (17) thru (19) compare the behavior of GP symbolic regression with and without swarm intelligence on a real world problem namely the forward estimation of corporate earnings on a database of stocks from 1990 thru 2009.

For the forward estimation of corporate earnings, this paper uses an historical database of approximately 1200 to 1500 stocks with daily price and volume data, weekly analyst estimates, and quarterly financial data from Jan 1986 to the present. The data has been assembled from reports published at the time, so the database is highly representative of what information was realistically available at the point when trading decisions were actually made.

From all of this historical data, twenty years (*1990 thru 2009*) have been used to support the results shown in this research. This two decade period includes a historically significant bull market decade followed by an equally historically significant bear market decade.

Multiple vendor sources have been used in assembling the data so that single vendor bias can be eliminated. The construction of this point in time database has focused on collecting weekly consolidated data tables, collected every Friday from Jan 3, 1986 to the present, representing detailed point in time input to this study and cover approximately 1200 to 1500 stocks on a weekly basis. Each stock record contains daily price and volume data, weekly analyst estimates and rankings, plus quarterly financial data as reported. The primary focus is on gross and net revenues.

The efficacy of several different swarm intelligence techniques are examined by running a full experimental protocol for each technique. Standard genetic programming, *without swarm intelligence techniques*, will be the base line for this study. We are interested in determining if the addition of swarm intelligence techniques improves symbolic regression performance – and if so, which swarm techniques perform best.

Our historical database contains 1040 weeks of data between January 1990 and December 2009. In a full training and testing protocol there is a separate symbolic regression run for each of these 1040 weeks. Each SR run consists of predicting the *ftmEPS* for each of the 1200 to 1500 stocks available in that week. A sliding training/testing window will be constructed to follow a strict statistical out-of-sample testing protocol.

For each of the 1040 weeks, the training examples will be extracted from records in the historical trailing five years behind the selected record BUT *not including any data from the selected week or ahead in time*. The training dependent variable will be extracted from the historical data record exactly 52 weeks forward in time from the selected record BUT *not including any data from the selected week or ahead in time*. Thus, as a practical observation, the training will not include any records in the first 52 weeks prior to the selected record – *because that would require a training dependent variable which was not available at the time*.

For each of the 1040 weeks, the testing samples will be extracted from records in the historical trailing five years behind the selected record *including all data from the selected week BUT not ahead in time*. The testing dependent variable will be extracted from the historical data record exactly 52 weeks forward in time from the selected record.

Each experimental protocol will produce 1040 symbolic regression runs over an average of 275,000 records for each training run and between 1200 and 1500 records for each testing run. Three hours will be allocated for training. Of course 1040 X 2 (*training and testing*) separate R-Square statistics will be produced for each experimental protocol. We will examine the R-Square statistics for evidence favoring the addition of swarm intelligence over the base line and for evidence favoring one swarm intelligence technique over another.

Finally we will need to adapt our methodology to conform to the rigorous United States Securities and Exchange Commission oversight and regulations on investment managers. The SEC mandates that every investment firm have a compliance officer. For any automated forward earnings prediction algorithm, *which would be used as the basis for later stock recommendations to external clients or internal portfolio managers,* the computer software code used in each prediction, the historical data used in each prediction, and each historical prediction itself, must be filed with the compliance officer in such form and manner so as to allow a surprise SEC compliance audit to reproduce each individual forward prediction exactly as it was at the original time of publication to external clients or internal portfolio managers.

Of course this means that we must provide a copy of all code, all data, and each forward prediction for each stock in each of the 1040 weeks, to our compliance officer. Once management accepts our symbolic regression system, we will also have to provide a copy of all forward predictions on an ongoing basis to the compliance officer.

Furthermore there is an additional challenge in meeting these SEC compliance details. The normal manner of operating GP, SI, and symbolic regression systems in academia will not be acceptable in a real world compliance environment. Normally, in academia, we recognize that symbolic regression is a heuristic search process and so we perform multiple SR runs, each starting with a different random number seed. We then report based on a statistical analysis of results across multiple runs. This approach produces *different results* each time the SR system is run. In a real world compliance environment such practice would subject us to serious monetary fines and also to jail time.

The SEC compliance requirements are far from arbitrary. Once management accepts such an SR system, the weekly automated predictions will influence the flow of millions and even billions of dollars into one stock or another and the historical back testing results will be used to sell prospective external clients and internal portfolio managers on using the system's predictions going forward.

First the authorities want to make sure that as time goes forward, *in the event that the predictions begin to perform poorly*, we will not simply rerun the original predictions again and again, with a different random number seed, until we obtain better historical performance and then substitute the new better performing historical performance results in our sales material.

Second the authorities want to make sure that, *in the event our firm should own many shares of the subsequently poorly performing stock of "ABC" Corp*, that we do not simply rerun the current week's predictions again and again, with a different random number seed, until we obtain a higher ranking for "ABC" stock thus improperly influencing our external clients and internal portfolio managers to drive the price of "ABC" stock higher.

In order to meet SEC compliance regulations we have altered our symbolic regression system, used in this chapter across all experiments, to use a pseudo random number generator with a pre-specified starting seed. Multiple runs always produce *exactly the same results*.

5. GP and Swarm in Symbolic Regression

In standard Koza-style tree-based Genetic Programming [12] 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 only 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.

6 depth 0 binary tree s-exp: 3.45
7 depth 1 binary tree s-exp: (+ x2 3.45)
8 depth 2 binary tree s-exp: (/ (+ x2 3.45) (* x0 x2))
9 depth 2 irregular tree s-exp: (/ (+ x2 3.45) 2.0)

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 so that the standard GP s-expression can be made swarm friendly. Let us use the following s-expression.

10 (* (/ (- x0 3.45) (+ x0 x2)) (/ (- x5 1.31) (* x0 2.1)))

In this individual (10), the real constants are embedded within the s-expression and are inconvenient for swarm algorithms. So we are going to add an annotation to the individual (10). We are going to add enumerated constant nodes, and we are going to add a constant chromosome vector creating a new individual (11). The individual (11) will now have three components: an abstract s-expression (11), the original s-expression (11.1), and a constant chromosome (11.2) as follows.

11 (* (/ (- x0 **c[0]**) (+ x0 x2)) (/ (- x5 **c[1]**) (* x0 **c[2]**)))
11.1 s-exp: (* (/ (- x0 3.45) (+ x0 x2)) (/ (- x5 1.31) (* x0 2.1)))
11.2 c: <3.45 1.31 2.1>

Individual (11) evaluates to the exact same value as (10). Each real number constant in (10) has been replaced with an indexed vector reference of the type **c[*i*]**, where **c** is a vector of real numbers containing the same real numbers originally found in (10). While this process adds some annotation overhead to (10), it does expose all of the real number constants in a vector which is swarm intelligence friendly.

At this point let us take a brief pause. Examine the original s-expression (10) also (11.1) and compare it to the new annotated abstract version (11). Walk through the evaluation process for each version. Satisfy yourself that the concrete s-expression (11.1) and the abstract annotated (11) both evaluate to exactly the same interim and final values.

We have made no restrictive or destructive changes in the original individual (10). Slightly altered to handle the new constant vector 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 (11), and replace each indirect reference with the proper value from the constant vector. This converts the abstract annotated (11) back into the concrete s-expression (11.1). Let your standard GP system operate on (11.1) any way it wishes to produce a new individual ($11^{\wedge}$.1). Now convert ($11^{\wedge}$.1) back into an abstract annotated version with the same process we used to annotate (10).

Furthermore, if we have a compiled a machine register optimized version, γ(x), of (10), we do not even have to perform expensive recompilation in order to change a value in the constant chromosome. We need only alter the values in the constant chromosome and re-evaluate the already compiled and optimized γ(x).

Armed with these newly annotated individuals, let's take a fresh look at how we might improve the standard process of genetic programming during a symbolic regression run. Consider the following survivor population in a standard GP island.

12.1 (* (/ (- x0 3.45) (+ x0 x2)) (/ (- x5 1.31) (* x0 2.1)))
12.2 (cos (/ (- x4 2.3) (min x0 x2)))
12.3 (* (/ (- x0 5.15) (+ x0 x2)) (/ (- x5 -2.21) (* x0 9.32)))
12.4 (sin (/ (- x4 2.3) (min x0 x2)))
12.5 (sin (/ (- x4 2.3) (avg x0 x2)))
12.6 (* (/ (- x0 3.23) (+ x0 x2)) (/ (- x5 -6.31) (* x0 7.12)))
12.7 (* (/ (- x0 2.13) (+ x0 x2)) (/ (- x5 3.01) (* x0 2.12)))

First of all, the GP mutation and crossover operators do not have any special knowledge of real numbers. They have a difficult time isolating and optimizing numeric constants. But the situation gets worse.

As generation after generation of training has passed, the surviving individuals in the island population have become specialized in common and predictable ways. Individuals (12.2), (12.4), and (12.5) are all close mutations of each other. Evolution has found a form that is pretty good and is trying to search for a more optimal version. GP is fairly good at exploring the search space around these individuals.

However, (12.1), (12.3), (12.6), and (12.7) are all identical forms with the exception of the values of their embedded numeric constants. As time passes, the survivor population will become increasingly dominated by variants of (12.1) and in time its progeny may crowd out all other survivors. GP has a difficult time exploring the search space around (12.1) largely because the form is already optimized – it is the constant values which need additional optimization.

In swarm friendly AEG enhanced symbolic regression system, the individuals (12.1), (12.3), (12.6), and (12.7) are all viewed as constant homeomorphs and they are stored in the survivor pool as one individual with another annotation: a swarm constant pool as follows.

13.1 (* (/ (- x0 **c[0]**) (+ x0 x2)) (/ (- x5 **c[1]**) (* x0 **c[2]**)))
  13.1.1 (* (/ (- x0 3.45) (+ x0 x2)) (/ (- x5 1.31) (* x0 2.1)))
  13.1.2 c: <3.45 1.31 2.1>
  13.1.3 Swarm Constant Pool
  13.1.3[0] <3.45 1.31 2.1>
  13.1.3[1] <5.15 -2.21 9.32>
  13.1.3[2] <3.23 -6.31 7.12>
  13.1.3[3] <2.13 3.01 2.12>
13.2 (cos (/ (- x4 2.3) (min x0 x2))) {*annotations omitted*}
13.3 (sin (/ (- x4 2.3) (min x0 x2))) {*annotations omitted*}
13.4 (sin (/ (- x4 2.3) (avg x0 x2))) {*annotations omitted*}

The AEG enhanced SR system has combined the individuals (12.1), (12.3), (12.6), and (12.7) into a single constant homeomorphic canonical version (13.1) with all of the constants stored in a swarm constant pool inside the individual. Now the GP island population does not become dominated inappropriately. Plus, we are free to apply swarm intelligence algorithms to the constants inside (13.1) without otherwise hindering the GP algorithms in any way.

The remainder of this chapter is devoted to comparing the effects of several hybrid algorithms on symbolic regression accuracy in predicting forward twelve month corporate earnings. The chosen algorithms are Standard Koza-style GP, GP with Particle Swarm, GP with Differential Evolution, and GP with the Bees algorithm.

6. AEG Conversion Algorithm

The Abstract Expression Grammar constant conversion algorithm is a straight forward search and replace type algorithm in which a standard Koza-style s-expression is converted into an annotated AEG individual as shown in Algorithm (1).

Algorithm 1: AEG Conversion

```
Input: in // Koza-style s-expression
Output: out // AEG annotated individual
Parameters: k, r, n, N
   Summary: AEG Conversion removes all of the constants from
   an input s-expression and places them in a vector where swarm
   intelligence algorithms can easily optimize them. The output is
   a constant vector and the original s-expression modified to
   refer indirectly into the constant vector instead of referencing
   the constants directly.
set out = <aexp,sexp,c,pool>  // empty AEG individual
set out.aexp = in
set out.sexp = in
set out.c = <..empty vector of reals..>
set out.pool = <..empty vector of vectors..>
set N  = length of out.aexp
for n from 0 until N do
  if  out.aexp[n] is a real number constant then
     set r = out.aexp[n]
     set k = length of out.c
     set out.c[k] = r
     set out.aexp[n] = "c[k]" // replace r with c indexed reference
  end if
set out.pool[0] = out.c
return out
```

7. GP Algorithm

Symbolic Regression with standard GP [8], [9], [10], and [12] evolves the GLM's basis functions as Lisp s-expressions. Evolution is achieved via the population operators of mutation, and crossover. We use a simple elitist GP algorithm which is outlined in Algorithm (2). The inputs are a vector of N training points, X, a vector of N dependent variables, Y, and the number of generations to train, G. Each point in X is a member of $R^M = <x_1,x_2,…,x_m>$. The fitness ***score*** is the root mean squared error divided by the standard deviation of Y, ***NLSE***.

Algorithm 2: Standard GP

```
Input: X // N vector of independent M-featured training points
Input: Y // N vector of dependent variables
Input: G // Number of generations to train
Output: champ // Champion s-expression individual
Parameters: K, P
   Summary: Standard GP searches for a champion s-expression
   by randomly growing and scoring a large number of candidate
   s-expressions, then iteratively creating and scoring new
   candidate s-expressions via mutation and crossover. After each
   iteration, the population of candidate s-expressions is
   truncated to those with the best score. After the final iteration,
   the champion is the s-expression with the best score.
function: mutateSExp(me)
   Summary: mutateSExp randomly alters an input s-expression
   by replacing a randomly selected sub expression with a new
   randomly grown sub expression.
  me = copy(me)
  set L = number of nodes in me // me is a list of Lisp Pairs
  set s = generate random s-expression
  set n = random integer between 0 and L
  set me[n] = s  // Replaces nth node with s
  return me
end fun
function: crossoverSExp(mom,dad)
   Summary: crossoverSExp randomly alters a mom input s-
   expression by replacing a randomly selected sub expression in
   mom with a randomly selected sub expression from dad.
  dad = copy(dad)
```

```
16   mom = copy(mom)
17   set $L_d$ = number of nodes in dad // dad is a list of Pairs
18   set $L_m$ = number of nodes in mom // mom is a list of Pairs
19   set n = random integer between 0 and $L_m$
20   set m = random integer between 0 and $L_d$
21   set mom[n] = dad[m]  // Replaces nth node with mth node
22   return mom
23 end fun
24 main logic
25 for k from 0 until K do // Initialize population
26   set w = generate random s-expression
27   set population.last = score(w)
28 end for k
29 sort population by fitness score
30 truncate population to P most fit individuals
31 set champ = population.first
32 for g from 0 until G do // Main evolution loop
33   for p from 0 until P do // Main evolution loop
34     set w = mutateSExp(population[p])
35     set population.last = score(w)
36     set dad = population[p]
37     set i = random integer between p and P
38     set mom = population[i]
39     set w = crossoverSExp(dad,mom)
40     set population.last = score(w)
41   end for p
42   sort population by fitness score
43   truncate population to P most fit individuals
44   set champ = population.first
45  end for g
46 return champ
```

Adding Abstract Expression Grammars to standard GP Symbolic Regression [3], [4], [5], and [6] evolves the GLM's basis functions as AEG individuals. Our simple modified elitist GP Algorithm (3) is outlined below. The inputs are a vector of N training points, X, a vector of N dependent variables, Y, and the number of generations to train, G. Each point in X is a member of $R^M = <x_1,x_2,\ldots,x_m>$. The fitness ***score*** is the root mean squared error divided by the standard deviation of Y, ***NLSE***.

Algorithm 3: AEG GP with Swarm

```
1  Input: X // N vector of independent M-featured training points
2  Input: Y // N vector of dependent variables
3  Input: G // Number of generations to train
4  Output: champ // Champion AEG individual
5  Parameters: K, P, S
```

Summary: *AEG GP with swarm searches for a champion s-expression as in standard GP (see Algorithm 2). However, before inserting s-expression candidates into the survivor population they are converted into AEGs and then merged with any similar AEGs (s-expressions with matching constant positions), then iteratively creating and scoring new candidate s-expressions via mutation, crossover, and swarm. After each iteration, the population of candidate AEG s-expressions is truncated to those with the best score. After the final iteration, the champion is the AEG s-expression with the best score.*

```
6  function: swarm(X,Y,aeg) // aeg = <aexp,sexp,c,pool>
7    …see Algorithm 5, 6, or 7…
8    return aeg
9  end fun
10 function: convertToAEG(sexp)
11   …see Algorithm 1…
12   return aeg
13 function: convertToSExp(aeg) // aeg = <aexp,sexp,c,pool>
14   …see Algorithm 4…
12   return sexp
15 function: insertInPop(aeg) // aeg = <aexp,sexp,c,pool>
```

Summary: *insertInPop accepts an input AEG s-expression then searches the population of AEG candidate s-expressions for a constant homeomorphic AEG s-expression (an AEG with matching form and constant locations ... although the value of the constants may be different). If a constant homeomorphic AEG is found, the input AEG is merged with the existing canonical version already in the population; otherwise, the input AEG is inserted in the population in order of its score.*

```
16   I = length of population
17   for i from 0 until I do // Search population
18     set w = population[i]
19     if (w.aexp = aeg.aexp) then
20       set w.pool = append(w.pool,aeg.pool)
21       sort w.pool by fitness score
22       truncate w.pool to S most fit constant vectors
23       set w.c = w.pool.first
24       set w.sexp = convertToSExp(w)
25       return population
26     end if
27   end for i
28   set population.last = aeg
29   return population
30 function: mutateSExp(me) // me = <aexp,sexp,c,pool>
```

Summary: *mutateSExp randomly alters an input s-expression by replacing a randomly selected sub expression with a new randomly grown sub expression.*

```
31   me = copy(me.sexp)
32   set L = number of nodes in me // me is a list of Lisp Pairs
33   set s = generate random s-expression
34   set n = random integer between 0 and L
35   set me[n] = s  // Replaces nth node with s
36   set me = convertToAEG(me)
37   return me
38 end fun
39 function: crossoverSExp(dad,mom)
```

Summary: *crossoverSExp randomly alters a **mom** input s-expression by replacing a randomly selected sub expression in **mom** with a randomly selected sub expression from **dad**.*

```
40   dad = copy(dad.sexp)
41   mom = copy(mom.sexp)
42   set $L_d$ = number of nodes in dad // dad is a list of Pairs
43   set $L_m$ = number of nodes in mom // mom is a list of Pairs
44   set n = random integer between 0 and $L_d$
45   set m = random integer between 0 and $L_m$
46   set dad[n] = mom[m]  // Replaces nth node with mth node
47   set dad = convertToAEG(dad)
48   return dad
49 end fun
50 main logic
51 for k from 0 until K do // Initialize population
52   set w = generate random s-expression
53   w = score(convertToAEG(w))
54   set population = insertInPop(w)
55 end for k
```

```
sort population by fitness score
truncate population to P most fit individuals
set champ = population.first
for g from 0 until G do // Main evolution loop
  for p from 0 until P do // Main evolution loop
    set w = swarm(population[p])
    set w = mutateSExp(population[p])
    set population = insertInPop(score(w))
    set dad = population[p]
    set i = random integer between p and P
    set mom = population[i]
    set w = crossoverSExp(dad,mom)
    set population = insertInPop(score(w))
  end for p
  sort population by fitness score
  truncate population to P most fit individuals
  set champ = population.first
end for g
return champ
```

Conversion from an AEG individual back to a standard s-expression is accomplished as outlined in Algorithm (4).

Algorithm 4: AEG To S-Expression Conversion

```
Input: in // AEG annotated individual <aexp,sexp,c,pool>
Output: out // Koza-style s-expression
Parameters: k, r, n, N
```

Summary: *AEG To S-Expression Conversion accepts an AEG annotated individual and returns a Koza-style s-expression with all of the indirect constant references replaced with the direct constant values taken from the AEG constant vector.*

```
set out = copy(in.aexp)
set N  = length of out.aexp
for n from 0 until N do
  if  out[n] is a constant reference "c[k]" then
    set r = in.aexp.c[k]
    set out[n] = r  // replace constant reference with constant
  end if
return out
```

8. AEG Differential Evolution

Abstract Expression Grammar GP can be used with differential evolution [7] which evolves the GLM's basis functions as AEG individuals. The DE algorithm encodes each individual as a constant vector. Each AEG <aexp,sexp,c,pool> stores the population of DE individuals in its constant **pool** and the current most fit champion as its constant vector **c**. In Algorithm (3) swarm evolution is seamlessly merged with standard GP and our AEG differential evolution algorithm is outlined In Algorithm (5).

The Differential Evolution algorithm is a straightforward attempt to keep a sorted list of the best constant vectors seen so far. Pairs of these constant vectors are selected at random along with the best constant vector seen so far. The algorithm then averages the differences between these constant vectors, in several obvious ways, to move closer to a global optimum.

Algorithm 5: AEG Differential Evolution

```
Input: X // N vector of independent M-featured training points
Input: Y // N vector of dependent variables
Input: in // AEG annotated individual <aexp,sexp,c,pool>
Output: in AEG annotated individual <aexp,sexp,c,pool>
Parameters: S
```

Summary: *AEG Differential Evolution optimizes a pool of vectors by selecting the best scoring vector along with a randomly selected pair of constant vectors, then the distances between these vectors are averaged in various ways to produce a new candidate vector to be scored. After scoring, the population of vectors is truncated to those with the best scores.*

```
function: randomNudge(c) // constant vector = <$c_0$,$c_2$,…,$c_j$>
```

Summary: *randomNudge accepts an input constant vector then produces a new constant vector by adding or subtracting small random increments from each constant in the input vector.*

```
var (defaultSkew .90) (defaultRange .20)
c = copy(c)
I = length of c
for i from 0 until I do
  set r = random number from 0 to defaultRange
  set r = defaultSkew + r
  set c[i] = r*c[i]
end for i
end fun
function: search(a,b,c)
```

Summary: *search accepts **a**, **b**, and **c** constant vectors in an input vector pool **in**. A new output constant vector **w** is created by randomly averaging the distances between the three vectors. The new vector **w** is used to score the AEG whose constant pool is being optimized. After scoring, the **in** pool is truncate to the constant vectors with the best scores. The score of the AEG is set to the score of the best constant vector in its pool.*

```
var (F .50)
w = copy(a)
I = length of a
for i from 0 until I do
  set r = random number from 0 to 1.0
  set r = F + r
  set w[i] = a[i] + (r*(b[i]-c[i]))
end for i
set in.pool.last = w
set in.c = w
score(in)
sort in.pool by fitness score
truncate in.pool to S most fit constant vectors
set in.c = in.pool.first
set in.sexp = convertToSExp(in)
return in
end fun
main logic
set I length of in.pool
if (I=0) then return in end if
set best = in.pool[0]
set $j_1$ = random integer from 0 until I
set $j_2$ = random integer from $j_1$ until I
set $b_1$ = in.pool[$j_1$]
if ($j_1$=0) then set $b_1$ = randomNudge(best)
set $b_2$ = in.pool[$j_2$]
if ($j_2$= $j_1$) then set $b_2$ = randomNudge($b_2$)
set r = random number from 0 until 1.0
// Modest momentum
if (r<.50) then search(best,best,$b_1$)
```

```
// Aggressive momentum
else if (r<.80) then search(best,best,$b_2$)
// Modest Mediation
else if (r<.85) then search($b_1$,best,$b_1$)
// Aggressive mediation
else if (r<.90) then search($b_2$,best,$b_2$)
// Wandering up
else if (r<.95) then search($b_2$,$b_1$,$b_2$)
// Wandering down
else set in.pool = search($b_1$,$b_2$,$b_1$)
return in
```

9. AEG Bees Algorithm

Abstract Expression Grammar GP can be used with Bees algorithm [14] and [15] which evolves the GLM's basis functions as AEG individuals. Each AEG <aexp,sexp,c,pool> stores the population of Bees individuals in its constant **pool** and the current most fit champion as its constant vector **c**. In Algorithm (3) swarm evolution is seamlessly merged with standard GP and our AEG bees algorithm is outlined in Algorithm (6) below.

Our Bees algorithm has been modified to fit within the larger framework of an evolving GP environment. Therefore, the evolutionary loop is in the GP algorithm and has been removed from the Bees algorithm. Instead the Bees algorithm is repeatedly called from the main GP loop during evolution. Furthermore, we must execute the Bees algorithm on all AEG individuals with a non-empty constant pool; therefore, care must be taken such that any one AEG individual does not monopolize the search process.

The Bees algorithm gets its inspiration from the cooperative behavior of bees foraging for food. There is the concept of a visited food site (which in our case is one of the constant vectors in the constant pool) and a bee which searches these food sites and assigns them a fitness value (in our case a bee is the AEG individual wrapped around and evaluating the constant vector). Since we have only one bee (the AEG individual), when multiple bees are required, we will have our single AEG individual search multiple times.

In the original Bees algorithm, there are S food sites selected for search (in our case the AEG's constant pool). Of the S selected sites, the E fittest sites are "*elite*" sites and the remaining (S-E) sites are "non-elite" sites. In the original Bees algorithm there are B bees. Since we have only one bee (the AEG individual), we will have our AEG individual search B times. Of the total B bees available, BEP bees are recruited to search the neighborhood around each elite food site, and BSP bees are recruited to search the neighborhood around each non-elite food site. The remaining BRP bees search at random anywhere they please. This all assumes that B = BEP+BSP+BRP.

In the original Bees algorithm, *for each elite food* site there are BEP neighborhood searches performed, *for each non-elite food* site there are BSP neighborhood searches performed, and there are BRP random searches performed in each iteration of the main evolutionary loop. Thus the total number of searches devoted to all elite food sites can be expressed as (E*BEP), while the total number of searches devoted to all non-elite food sites can be expressed as ((S-E)*BSP), and the total number of random searches can be expressed by the fraction BRP. From these counts of total searches performed, we can derive the probability that an elite site will be searched, that a non-elite site will be searched, and that a random search will be performed. These computed percentages will be the parameters of our modified Bees algorithm: BEp, BSp, and BRp.

Algorithm 6: AEG Bees Algorithm

```
Input: X // N vector of independent M-featured training points
Input: Y // N vector of dependent variables
Input: in // AEG annotated individual <aexp,sexp,c,pool>
Output: in AEG annotated individual <aexp,sexp,c,pool>
Parameters: BEp, BSp, BRp, E, S
    Summary: AEG Bees Algorithm optimizes a pool of vectors by
    incrementally selecting each vector from the pool of constant
    vectors, then either producing a new candidate vector in a
    random neighborhood around the selected vector or producing
    a new random vector. The new vector is scored. After scoring,
    the population of vectors is truncated to those with the best
    scores.
function: neighborSearch(c) // constant vector = <$c_0$,$c_2$,...,$c_j$>
    Summary: neighborSearch accepts an input constant vector
    then produces a new constant vector by adding or subtracting
    small random increments from each constant in the input
    vector. The new vector is scored and inserted into the constant
    pool.
w = copy(c)
d = copy(c)
I = length of c
J = length of in.Pool
// compute local neighborhood radius vector
for j from 1 until J do
  for i from 0 until I do
    set d[i] += (abs(in.Pool[j-1][i]-in.Pool[j][i])/(J-1))
  end for i
 end for j
 // Search the local neighborhood
for i from 0 until I do
  set r = random number from 0 to (2*d[i])
  set r = r – d[i]
  set w[i] = w[i]+r;
end for i
 set in.pool.last = w
 set in.c = w
 score(in)
 sort in.pool by fitness score
 truncate in.pool to S most fit constant vectors
 set in.c = in.pool.first
 set in.sexp = convertToSExp(in)
end fun
function: randomSearch()
    Summary: randomSearch produces a new constant vector by
    randomly setting a value to each constant in the new vector.
    The new vector is scored and inserted into the constant pool.
 w = random constant vector
 set in.pool.last = w
 set in.c = w
 score(in)
 sort in.pool by fitness score
 truncate in.pool to S most fit constant vectors
 set in.c = in.pool.first
 set in.sexp = convertToSExp(in)
```

```
40  return in
41 end fun
42 main logic
43 vars ($I_e$ starts at 0) ($I_f$ starts at E)
44 set I length of in.pool
45 if (I=0) then return in end if
46 set ce = if ($I_e$<E) then in.pool[$I_e$] else in.pool.first end if
47 set $I_e$ = $I_e$ + 1
48 if ($I_e$>=E) then set $I_e$ = 0 end if
49 set cf = if ($I_f$<I) then in.pool[$I_f$] else in.pool.first end if
50 set $I_f$ = $I_f$ + 1
51 if ($I_f$>=I) then set $I_f$ = E end if
52 set choice = random integer between 0 and 1.0
53 if (choice<BEp) then neighborSearch(ce) end if
54 if (choice<BSp) then neighborSearch(cf) end if
55 if (choice<BRp) then randomSearch() end if
56 return in
```

10. AEG Particle Swarm

Abstract Expression Grammar GP can be used with particle swarm [2] which evolves the GLM's basis functions as AEG individuals. In Algorithm (3) swarm evolution is seamlessly merged with standard GP and our AEG particle swarm algorithm is outlined in Algorithm (7) below.

Our Particle Swarm (PSO) algorithm has also been modified to fit within the larger framework of an evolving GP environment. Therefore, the evolutionary loop is in the GP algorithm and has been removed from the PSO algorithm. Instead the PSO algorithm is repeatedly called from the main GP loop during evolution. Furthermore, we must execute the PSO algorithm on all AEG individuals with a non-empty constant pool; therefore, care must be taken such that any one AEG individual does not monopolize the search process.

The PSO algorithm gets its inspiration from the clustering behavior of birds or insects as they fly in formation. There is the concept of an individual swarm member called a particle, the current position of each particle, the best position ever visited by each particle, a velocity for each particle, and the best position every visited by any particle (the global best). In our case, each particle will be one of the constant vectors in our AEG individual's constant pool. A fitness value will be assigned to each constant by wrapping the AEG individual around the constant vector and scoring.

Each AEG <aexp,sexp,c,pool> stores the population of PSO individuals in its constant **pool** and the current most fit champion as its constant vector **c**. However, implementing the PSO algorithm requires adding a few new items to our AEG individual. Let ***aeg*** be an AEG individual in our system. The best position ever visited by any particle will be designated as ***aeg*.best** (global best). The best position ever visited by each particle, i, will be designated as ***aeg*.pool[i]→best** (local best). The velocity of each particle, i, will be designated as ***aeg*.pool[i]→v**. The score of a constant vector, ***c***, will be designated as **fitness(*c*)**. And, of course, each particle, i, is nothing more than one of the constant vectors in the AEG individual's constant pool ***aeg*.pool[i]**.

Algorithm 7: AEG Particle Swarm

```
1  Input: X // N vector of independent M-featured training points
2  Input: Y // N vector of dependent variables
3  Input: in // AEG annotated individual <aexp,sexp,c,pool>
4  Output: in AEG annotated individual <aexp,sexp,c,pool>
5  Parameters: $W_L$, $W_G$, $W_V$, S
   Summary: AEG Particle Swarm optimizes a pool of vectors by
   randomly selecting a pair of constant vectors from the pool of
   constant vectors. A new vector is produced when the pair of
   vectors, together with the global best vector, are randomly
   nudged closer together based upon their previous approaching
   velocities. The new vector is scored. After scoring, the
   population of vectors is truncated to those with the best scores.
6  main logic
7  vars ($I_c$ starts at 0)
8  set J = length of in.pool
9  if (J<=0) then return in end if
10 i = $I_c$
11 c = copy(in.pool[i])
12 v = copy(in.pool[i]→v)
13 if (v = null) then
14  set v = random velocity vector
15  set in.pool[i]→v = v
16 end if
17 lbest = in.pool[i]→best
18 if (lbest = null) then
19  set lbest = c
20  set in.pool[i]→best = lbest
21 end if
22 gbest = in.best
23 if (gbest = null) then
24  set gbest = c
25  set in.best = gbest
26 end if
27 // Compute the velocity weight parameters
28 maxg = maximum generations in the main GP search
29 g = current generation count in the main GP search
30 $W_L$ = .25 + ((maxg – g)/maxg) // local weight
31 $W_G$ = .75 + ((maxg – g)/maxg) // global weight
32 $W_V$ = .50 + ((maxg – g)/maxg) // velocity weight
33 I = length of c
34 set r1 = random number from 0 to 1.0
35 set r2 = random number from 0 to 1.0
36 // Update the particle's velocity & position
37 for i from 0 until I do
38   set lnudge = ($W_L$*r1*(lbest[i]-c[i]))
39   set gnudge = ($W_G$*r2*(gbest[i]-c[i]))
40   set v[i] = ($W_V$*v[i])+lnudge+gnudge
41   set c[i] = c[i]+v[i]
42 end for i
43 // Score the new particle position
44 set in.c = c
45 score(in)
46 // Update the best particle positions
47 if (fitness(c)>fitness(lbest)) then lbest = c end if
48 if (fitness(c)>fitness(gbest)) then gbest = c end if
49 in.best = gbest
50 set in.pool.last = c
51 set in.pool.last→best = lbest
52 set in.pool.last→v = v
53 // Enforce elitist constant pool
54 sort in.pool by fitness score
```

```
55 truncate in.pool to S most fit constant vectors
56 set in.c = in.pool.first
57 set in.sexp = convertToSExp(in)
58 // Enforce iterative search of constant pool
59 set $I_c = I_c + 1$
60 if ($I_c$>=S) then set $I_c = 0$ end if
61 return in
```

11. Sample Test Problems

Several sample test problems have been collected upon which we can compare the performance of standard GP symbolic regression and hybrid AEG symbolic regression. Each of these test problems contains an embedded real constant which greatly affects the behavior of the formula during regression. If our theory is correct, these test problems should receive better results with AEG symbolic regression than with standard GP symbolic regression. The test problems are as follows.

14.1 y = -2.3 + (0.13*sin(**4.1***x2))
14.2 y = 3.0 + (2.13*log(**1.3**+x4))
14.3 y = 2.0 - (2.1*cos(**9.8**/x0))

Two symbolic regressions are performed for each test problem: standard GP symbolic regression, and AEG symbolic regression (using the Bees Algorithm 6). Clearly the AEG symbolic regressions perform much better than standard GP symbolic regression. Table 1 shows the results.

Table 1: Sample Test Problem Regressions

Formula	NLSE GP	RSQ GP	NLSE AEG	RSQ AEG
14.1	.47	.77	0.0	1.0
14.2	.18	.96	0.0	1.0
14.3	.36	.81	0.0	1.0

Note: *NLSE is the least squared error divided by the standard deviation of Y, and RSQ is the R-Square statistic from the regression*. An NLSE of 0.0 is perfect while an RSQ of 1.0 is perfect.

Clearly the AEG symbolic regression runs are discovering and optimizing the embedded constants correctly; however, the standard GP symbolic regression runs are unable to optimize the constants and get confused. It is simply too difficult for standard GP to optimize these difficult embedded constants *using only mutation and crossover*. Furthermore, the standard GP runs produce estimators which are far from the correct form. The following are the top five estimators, produced by the standard GP symbolic regression, for test problem (14.1).

14.1.1 y = 4.6+(-2.45*(sqrt(log(x0))));
14.1.2 y = -11919+(-0.86*((-13824+log(x0))));
14.1.3 y = -1891+(-0.8624*((-2197+log(x0))));
14.1.4 y = -2073+(-0.8624*((-2401+log(x0))));
14.1.5 y = -1749+(-0.8624*((-2025+log(x0))));

The results are so absolute that statistical analysis is unnecessary. Standard GP symbolic regression cannot solve these problems, while AEG symbolic regression always solves these problems exactly. Furthermore, it is clear that the standard GP run is trying to optimize constants but it has gotten stuck in a local minimum with the wrong formula and its population of champions is dominated by the attempt to optimize constants rather than trying to find a better fitting formula.

Incidentally, it made no difference when the Bees Algorithm was replaced with the Differential Evolution Algorithm or with the Particle Swarm Algorithm. The results of an AEG symbolic regression on the sample test problems was a perfect score no matter which swarm algorithm was chosen.

Furthermore, on the issue of scientific reproducibility, we have included detailed algorithms in this chapter. No matter what random seed is used, standard GP SR will not optimize sample problems 14.1, 14.2, and 14.3 in any practical time. This is because the population operators available to standard GP SR do not manage imbedded constants. Plus no matter what random seed is used, SR with any one of the three popular swarm algorithms will optimize the sample problems 14.1, 14.2, and 14.3 very quickly. These results are easily scientifically reproduced.

Now that we have tested AEG symbolic regression on several sample test problems, achieving much better performance than standard GP symbolic regression, it is time to compare AEG with standard GP symbolic regression on a real world investing problem: estimating forward 12 month earnings per share for a database of companies between 1990 and 2009. We begin with some background on investing. In addition, we will also compare the results of the three different swarm intelligence algorithms.

12. Investing Strategies

Value investing [1] has produced several of the wealthiest investors in the world including Warren Buffet. Nevertheless, value investing has a host of competing strategies including momentum [16] and hedging [17].

One of the most difficult challenges in devising a securities investing strategy is the a priori identification of pending regime changes. For instance, momentum investing strategies were very profitable in the 1990's and not so profitable in the 2000's while value investing strategies were not so profitable in the 1990's but turned profitable in the 2000's. Long Short hedging strategies were profitable in the 1990's and early 2000's but collapsed dramatically in the late 2007 thru 2008 period. Knowing when to switch from Momentum to Value, Value to Hedging, and Hedging back to Value was critical for making consistent above average profits during the twenty year period from 1990 thru 2009.

The challenge becomes even more difficult when one adds the numerous technical and fundamental buy/sell triggers to currently popular active management investing strategies. Bollinger Bands, MACD, Earning Surprises, etc. all have complex and dramatic effects on the implementation of securities investing strategies, and all are vulnerable to regime changes. The question arises, "*Is there a simple securities investing strategy which is less vulnerable to regime changes than other strategies?*".

An idealized value investing hypothesis is put forward: "*Given perfect foresight, buying stocks with the best future earning yield **(Next12MoEPS/CurrentPrice)** and holding for 12 months will produce above average securities investing returns*".

Using our database *of the 1500 Valueline stocks from 1986 thru 2009*, we studied three ideal concentrated portfolios: five, twenty five, and fifty stock portfolios. Each of these idealized concentrated portfolios are sampled each month for the twenty years from 1990 thru 2009. Fixed holding periods of one month, one quarter, and one year were examined. The per annum compound return for each decade and each holding period are shown in Table 2 along with the compounded returns, including dividends, of the Standard & Poor's 500 for each decade.

Table 2: Returns for idealize future earnings yield

Holding period	Decade	5 stocks	25 Stocks	50 Stocks
month	1990s	76%	69%	63%
month	2000s	120%	69%	53%
quarter	1990s	58%	73%	64%
quarter	2000s	69%	74%	53%
year	1990s	48%	46%	41%
year	2000s	103%	61%	45%
SP500	1990s	18%	18%	18%
SP500	2000s	(2%)	(2%)	(2%)

Note: *Per annum compound returns for each decade*.

The data supports the conclusion that the ideal hypothesis yields highly above average investing profits for all portfolio sizes and all holding periods across both decades. Furthermore the ideal hypothesis appears less vulnerable to regime changes than many other popular active securities investment strategies given that the 1990s decade was a raging bull environment while the 2000s decade was a terrible bear environment.

13. Buying Current Earnings Yield

Of course the ideal hypothesis is impossible to implement because it requires perfect foresight which is, in the absence of time travel, unobtainable. Nevertheless the ideal hypothesis represents the theoretical upper limit on the profits realizable from a strategy of buying future revenue cheaply; yet, the theoretical profits are so rich that one cannot help but ask the question, "*Are there revenue prediction models which will allow one to capture some portion of the profits from the ideal hypothesis?*".

The easiest revenue prediction model involves simply using the current year's trailing 12 month revenue as a proxy for future revenue.

The data supports the conclusion that even using this current revenue proxy model buying the top five, twenty five, and fifty stocks with the highest ***(current12MoEPS/currentPrice)*** produces above average securities investing profits, *as least for the 1500 Valueline stocks*, as shown in Table 3.

Table 3: Returns for current revenue prediction

Holding period	Decade	5 stocks	25 Stocks	50 Stocks
month	1990s	29.0%	16.5%	16.6%
month	2000s	8.2%	11.4%	15.4%
quarter	1990s	41.7%	14.9%	14.9%
quarter	2000s	22.7%	13.5%	15.6%
year	1990s	36.4%	17.6%	15.6%
year	2000s	42.1%	19.7%	17.4%
SP500	1990s	18%	18%	18%
SP500	2000s	(2%)	(2%)	(2%)

Note: *Per annum compound returns for each decade.*

Clearly using this current revenue prediction model buying the top five, twenty five, and fifty stocks with the highest ***(current12MoEPS/currentPrice)***, produces above average securities investing profits, in most cases, *especially with one year holding periods.*

Like buying stocks *with the best future earning yield* ***(Next12MoEPS/CurrentPrice)***, buying current earnings yield ***(current12MoEPS/currentPrice)*** is an *ideal* method. By *ideal* we mean that all information is known and exact. There is no predictive aspect, no guess work. We already know what current earnings are for any stock.

Nevertheless, buying a stock with low PE but whose future 12 month earnings will plummet bringing on bankruptcy is an obviously poor choice. So why is low PE investing so successful given that future 12 month earnings can vary significantly? Placing current earnings yield investing in this context puts a new spin on this standard *value investing* measure. In this context we are saying that current earnings yield (also known as low PE investing) works precisely to the extent that *current earnings are a reasonable predictor of future earnings*! In situations where current earnings are NOT a good predictor of future earnings, then current earnings yield investing looses it efficacy.

This agrees with our common sense understanding. For instance, given two stocks with the same high current earnings yield, where one will go bankrupt next year and the other will double its earnings next year; we would prefer the stock whose earnings will double. Implying that, in the ideal, current earnings are just a data point. We want to buy *future earnings* cheap!

Precisely because the per annum returns from this current revenue prediction model are far less than the returns achieved with perfect prescience, we must now look for more accurate methods of net revenue prediction.

14. Future Revenue Prediction *Inputs*

One very simplistic revenue prediction input model involves simply adding last year's revenue delta to current revenue as a prediction of future revenue, as follows:

15 2010EPS = (2009EPS-2008EPS)+2009EPS

...to generalize, we have:

15.1 forwardRevenue = (revenue-pastRevenue)+revenue

Another simple revenue prediction input is the broker estimates. Each week there appears a broker consensus estimate for the next 12Mo EPS for each of the stocks in our database. This broker revenue prediction can be used as a model for future revenue.

If we combine a number of these simple future revenue prediction inputs together we can construct a set of consensus inputs for prediction of future revenue. Constructing this consensus revenue inputs requires the following components.

16 margin = (currentEPS/currentSPS)

17 brokerEPS = *broker consensus estimate*

18 forwardEPS = (currentEPS-pastEPS)+currentEPS

19 projectEPS = (4*(currentEPS-pastQtrEPS))+currentEPS

20 forwardSPS = (currentSPS-pastSPS)+currentSPS

21 projectSPS = (4*(currentSPS-pastQtrSPS))+currentSPS

22 forwardSEPS = forwardSPS*margin

23 projectSEPS = projectSPS*margin

The five bolded elements above (brokerEPS, forwardEPS, projectEPS, forwardSEPS, and projectSEPS) are the consensus inputs to all of our future revenue prediction efforts in the remainder of this chapter.

15. Future Revenue: *GP-only*

Each week we can construct a GP-only symbolic regression estimate (using Algorithm 2) for next 12Mo EPS for each of the stocks in our database, using the following five inputs as dependent variables: **brokerEPS**, **forwardEPS**, **projectEPS**, **forwardSEPS**, and **projectSEPS**. Each week we train a symbolic regression model on approximately 375,000 training examples (250 weeks of backward historical data times approximately 1,500 stocks), and each week we use the newly trained symbolic regression model to predict the earnings per share of each stock in our database for the new week. This is a text book case of in-sample-training with out-of-sample-testing using a sliding forward 250 week training window.

The per annum returns using this symbolic regression revenue prediction model buying the top five , twenty five, and fifty stocks with the highest ***(regression12MoEPS/currentPrice)*** produces above average securities investing profits as shown in Table 4.

Table 4: Returns for GP-only

Holding period	Decade	5 stocks	25 Stocks	50 Stocks
month	1990s	33.2%	17.9%	18.2%
month	2000s	9.7%	13.2%	17.6%
quarter	1990s	43.9%	16.8%	15.1%
quarter	2000s	25.6%	15.3%	18.5%
year	1990s	39.2%	18.8%	17.8%
year	2000s	45.6%	21.2%	18.9%
SP500	1990s	18%	18%	18%
SP500	2000s	(2%)	(2%)	(2%)

Note: *Per annum compound returns for each decade.*

Clearly using the GP-only symbolic regression revenue prediction model buying the top five , twenty five, and fifty stocks with the highest ***(regression12MoEPS/currentPrice)*** produces above average securities investing profits, in most cases. *In fact, compared with all simple prediction methods shown so far, for reasonably diversified fifty stock portfolios, the annual hold returns are the best we have seen so far.*

Nevertheless, despite the satisfying accuracy and high returns, there are issues with the GP symbolic regression model. The main issue with the GP regression approach is a fundamental issue of believability. Every mathematical model, however highly correlated with market behavior over a period, must withstand the test of believability.

Because the standard GP process is difficult to constrain, many of the basis functions reach sizes and complexities beyond reasonable. For instance, in March of 1998 the GP regression creates an earnings model containing the term: ***tanh(forwardEPS/brokerEPS)***. This strains the credulity of any fund portfolio manager and is very difficult to explain using standard financial concepts. It clearly works statistically in that training period; but, it is not believable.

Worse still, in order to achieve its high accuracy, the GP regression process drives the coefficients on some of the basis functions to negative values. This also creates a financial model which does not make common sense, and is therefore **unbelievable**. When the champion estimator, produced by symbolic regression is ridiculous, it undermines the acceptance of the whole symbolic regression process vis a vis investing, and no fund manager will risk assets based upon the SR models.

For instance, for the month of April 2001 the GP regression method creates an earnings model with a highly weighted basis function where the coefficient for **forwardEPS** is negative. …

24 eps = …+(-1.293*forwardEPS$^{2)}$+…

Since forwardEPS is the result of adding last year's earnings growth to this year's earnings to get an estimate for next year, a negative coefficient has the SR model telling us that companies with big earnings growth last year are bad! AND the larger last year's earnings growth the ***worse*** the model penalizes the company.

A statistician will immediately suspect over fitting in this SR champion model. Professional investors are less kind in their incredulity. Unfortunately standard GP symbolic regression produces many champions with these believability problems.

Many of the champion estimator models produced by standard GP symbolic regression simply do not pass the common sense test. Investing large amounts of risk assets based on these GP models is very problematic *because of the GP model's fundamental lack of believability*. Even in the unlikely event that management were to sign off, regulatory and compliance sign off would be impossible.

16. Basis Function Constraints using AEG

Abstract Expression Grammars (AEGs) can be used to constrain the basis functions searched in a symbolic regression so that the believability issues with standard GP are resolved [6] and [13]. In our case it is reasonable and believable to constrain the basis functions to either sigmoid or Classification and Regression Tree (CART) sigmoid.

Using our five future revenue predictions as inputs to a nonlinear sigmoid regression, we can construct a more believable prediction model. Our first attempt will be to stay with an almost linear regression, but where the model coefficients are forced into the sigmoid domain. The model coefficients cannot go negative and they cannot rise above 1.0. This creates a more believable regression model in which the coefficients act more like significance weights attached to each of the five input EPS predictions as follows.

25 eps = c_1***brokerEPS**+ c_2***forwardEPS**+ c_3***projectEPS**

+ c_4***forwardSEPS**+ c_5***projectSEPS**

where $0 \le c_i \le 1.0$ for $1 \le i \le 5$

In this sigmoid linear regression model each coefficient represents the significance given to one of the five input predictions. Therefore if c_1=.2 while c_2=.4, the model is saying that the higher the brokerEPS estimate and the higher the forwardEPS estimate the better; BUT, the model gives twice as much weight to forwardEPS estimates as it does to brokerEPS estimates. This is a far more intuitively believable model.

Also it is possible to construct a more sophisticated sigmoid Classification and Regression Tree (CART) model by using the sigmoid model (24) as a template for four leaf nodes of a simple classification tree as follows.

25.1 μ_1 = c_1***brokerEPS**+ c_2***forwardEPS**+ c_3***projectEPS**

+ c_4***forwardSEPS**+ c_5***projectSEPS**

where $0 \le c_i \le 1.0$ for $1 \le i \le 5$

25.2 μ_2 = c_6***brokerEPS**+ c_7***forwardEPS**+ c_8***projectEPS**

+ c_9***forwardSEPS**+ c_{10}***projectSEPS**

where $0 \le c_i \le 1.0$ for $6 \le i \le 10$

25.3 μ_3 = c_{11}***brokerEPS**+ c_{12}***forwardEPS**+ c_{13}***projectEPS**

+ c_{14}***forwardSEPS**+ c_{15}***projectSEPS**

where $0 \le c_i \le 1.0$ for $11 \le i \le 15$

25.4 μ_4 = c_{16}***brokerEPS**+ c_{17}***forwardEPS**+ c_{18}***projectEPS**

+ c_{19}***forwardSEPS**+ c_{20}***projectSEPS**

where $0 \le c_i \le 1.0$ for $16 \le i \le 20$

We can then place these sigmoid leaf nodes into a simple CART formula as follows.

25.5 eps = $(v_1<v_2)?((v_3<v_4)?\mu_1:\mu_2):(v_5<v_6)?\mu_3:\mu_4)$

where **V** = {**brokerEPS,forwardEPS,projectEPS, forwardSEPS,projectSEPS**}

where $v_i \varepsilon$ **V** for $1 \le i \le 4$

In this sigmoid CART nonlinear regression model each of the four leaf nodes is a sigmoid nonlinear model of the type shown in (24). Each of the decision variables, v_i, is one of the five possible inputs.

By constraining the basis functions searched to be either sigmoid or CART sigmoid, we automatically eliminate the issues associated with GP-only future revenue prediction, and we achieve future earnings models which pass the test all important test of believability.

Unfortunately, having imposed these important basis function constraints, we encounter an additional issue. GP-only symbolic regression is very poor at evolving real number constants. These constraints place a heavy emphasis on the evolution of real number constants within the basis function and its sigmoid coefficients. Therefore we must add, to our hybrid AEG algorithm, evolutionary techniques which are better able to evolve real number constants. The remainder of this chapter will compare the efficacy of three hybrid evolutionary algorithms on the task of future revenue prediction.

17. GP with Particle Swarm

Testing the algorithm in (6.1) and limiting our basis functions to either sigmoid or CART sigmoid as in Section 13, each week we can construct a symbolic regression estimate for next 12Mo EPS for each of the stocks in our database, using the following five inputs as dependent variables: **brokerEPS**, **forwardEPS**, **projectEPS**, **forwardSEPS**, and **projectSEPS**.

Each week we train a symbolic regression model on approximately 375,000 training examples (250 weeks of backward historical data times approximately 1,500 stocks), and each week we use the newly trained symbolic regression model to predict the earnings per share of each stock in our database for the new week.

The per annum returns using this symbolic regression revenue prediction model buying the top five , twenty five, and fifty stocks with the highest ***(regression12MoEPS/currentPrice)*** produces above average securities investing profits as shown in Table 5.

Table 5: Returns for GP with Particle Swarm

Holding period	Decade	5 stocks	25 Stocks	50 Stocks
month	1990s	21.2%	26.1%	22.2%
month	2000s	7.6%	13.9%	17.8%
quarter	1990s	12.9%	29.2%	25.1%
quarter	2000s	9.2%	14.7%	19.2%
year	1990s	37.7%	26.3%	21.3%
year	2000s	5.6%	22.5%	22.6%
SP500	1990s	18%	18%	18%
SP500	2000s	(2%)	(2%)	(2%)

Note: *Per annum compound returns for each decade*.

Clearly using the GP with particle swarm symbolic regression revenue prediction model buying the top five , twenty five, and fifty stocks with the highest ***(regression12MoEPS/currentPrice)*** produces above average securities investing profits, in most cases. *In fact, compared with GP-only prediction methods, adding particle swarm has increased accuracy significantly – while adding believability.*

18. GP with Differential Evolution

Testing the algorithm in (6) and limiting our basis functions to either sigmoid or CART sigmoid as in Section 13, each week we can construct a symbolic regression estimate for next 12Mo EPS for each of the stocks in our database, using the following five inputs as dependent variables: **brokerEPS**, **forwardEPS**, **projectEPS**, **forwardSEPS**, and **projectSEPS**.

Each week we train a symbolic regression model on approximately 375,000 training examples (250 weeks of backward historical data times approximately 1,500 stocks), and each week we use the newly trained symbolic regression model to predict the earnings per share of each stock in our database for the new week.

The per annum returns using this symbolic regression revenue prediction model buying the top five , twenty five, and fifty stocks with the highest ***(regression12MoEPS/currentPrice)*** produces above average securities investing profits as shown in Table 6.

Table 6: Returns for GP with Differential Evolution

Holding period	Decade	5 stocks	25 Stocks	50 Stocks
month	1990s	20.6%	26.8%	22.6%
month	2000s	7.4%	14.8%	18.6%
quarter	1990s	13.6%	29.0%	24.3%
quarter	2000s	9.6%	14.2%	18.8%
year	1990s	37.9%	27.4%	23.8%
year	2000s	5.3%	21.3%	21.5%
SP500	1990s	18%	18%	18%
SP500	2000s	(2%)	(2%)	(2%)

Note: *Per annum compound returns for each decade.*

Clearly using the GP with differential evolution symbolic regression revenue prediction model buying the top five , twenty five, and fifty stocks with the highest ***(regression12MoEPS/currentPrice)*** produces above average securities investing profits, in most cases. *However the GP with differential evolution algorithm does not yield a significant improvement over GP with particle swarm.*

19. GP with Bees Algorithm

Testing the algorithm in (7) and limiting our basis functions to either sigmoid or CART sigmoid as in Section 13, each week we can construct a symbolic regression estimate for next 12Mo EPS for each of the stocks in our database, using the following five inputs as dependent variables: **brokerEPS**, **forwardEPS**, **projectEPS**, **forwardSEPS**, and **projectSEPS**.

Each week we train a symbolic regression model on approximately 375,000 training examples (250 weeks of backward historical data times approximately 1,500 stocks), and each week we use the newly trained symbolic regression model to predict the earnings per share of each stock in our database for the new week.

The per annum returns using this symbolic regression revenue prediction model buying the top five , twenty five, and fifty stocks with the highest ***(regression12MoEPS/currentPrice)*** produces above average securities investing profits as shown in Table 7.

Table 7: Returns for GP with Bees Algorithm

Holding period	Decade	5 stocks	25 Stocks	50 Stocks
month	1990s	107.6%	66.7%	43.7%
month	2000s	9.8%	16.9%	19.3%
quarter	1990s	51.3%	37.9%	31.5%
quarter	2000s	10.5%	18.3%	19.4%
year	1990s	26.8%	30.0%	22.2%
year	2000s	15.4%	28.9%	24.0%
SP500	1990s	18%	18%	18%
SP500	2000s	(2%)	(2%)	(2%)

Note: *Per annum compound returns for each decade.*

Clearly using the GP with Bees Algorithm symbolic regression revenue prediction model buying the top five , twenty five, and fifty stocks with the highest ***(regression12MoEPS/currentPrice)*** produces above average securities investing profits, in most cases. *In fact, compared with all other prediction methods (*referring to fifty stock portfolios, which have less statistical variance than smaller portfolios*) adding the Bees algorithm has increased accuracy significantly over GP-only and is a slight improvement over GP with particle swarm and GP with differential evolution.* However, the Bees slight performance improvement over DE and PSO is not statistically significant under rigorous statistical analysis.

20. Summary

Having no population operators of its own which specialize in constant optimization, it is our contention that standard GP symbolic regression can benefit greatly when enhanced with swarm intelligence algorithms specializing in constant optimization. A method of integrating standard GP with swarm intelligence, Abstract Expression Grammars is introduced.

The importance of constants in symbolic regression is studied. It is shown that the size of the search space, for even simple financial applications, is very large and that a significant portion of that size is due to the presence of constants.

Several sample test problems, with embedded constants, are presented with standard GP symbolic regression unable to solve any of the problems while AEG enhanced SR is always able to solve each of the problems exactly. It made no difference which swarm algorithm was used – DE, Bees, or PSO. It was the presence of AEG integrated swarm intelligence which made the test problems tractable.

Theoretical, methodological, and regulatory issues applying standard GP symbolic regression to an important investment finance application are discussed. Symbolic regression is enhanced, using AEG, to be applicable to the prediction of forward 12 month earnings per share. A number of bloat and believability issues applying SR to predicting forward 12 month earnings are addressed and solved with AEG.

AEG enhanced symbolic regression is used to predict forward 12 month earnings per share on approximately 1500 stocks from 1990 to 2009. Three distinct swarm intelligence algorithms are compared: DE, Bees, and PSO. All three swarm algorithms perform well, providing earnings predictions in a format easily acceptable by portfolio managers and regulatory compliance officers.

Incidentally, comparing t-statistics, f-statistics, variance, information ratio and p-values shows it made no difference when the Bees Algorithm was replaced with the Differential Evolution Algorithm or with the Particle Swarm Algorithm. The results of an AEG symbolic regression on predicting future 12Mo eps was statistically similar for all swarm algorithms compared. It was the integration with any of the three swarm algorithms which made symbolic regression effective for forward earning prediction.

Enhancing standard GP with Abstract Expression Grammar hybrid algorithms solves a number of regression accuracy, believability, and regulatory issues when using symbolic regression in financial applications. Based upon our experiments in this chapter, standard GP symbolic regression has serious issues when applied to financial applications; while, swarm enhanced SR shows real promise in the financial domain.

Furthermore using AEG to add swarm intelligence algorithms to SR significantly enhanced accuracy in future 12 month revenue prediction and produced above average securities investing profits in the historical period 1990 to 2009. Significantly this superior performance was undeterred by the bearish market environment of the 200 decade.

Directions for future research include investigating whether or not there are other swarm algorithms which would show real statistical significantly improved results over DE, Bees, and PSO? Is AEG the optimal GP SI integration approach to symbolic regression, or is there another integration approach which is superior?

References

[1] Graham, Benjamin, and David Dodd. 2008. Securities Analysis. New York, New York, USA. McGraw-Hill.

[2] Kennedy, J.; Eberhart, R. 1995. Particle Swarm Optimization. *Proceedings of IEEE International Conference on Neural Networks*. **IV**. pp. 1942–1948.

[3] Korns, Michael F. 2007. Large-Scale, Time-Constrained Symbolic Regression-Classification. In Riolo, Rick, L, Soule, Terrance, and Wortzel, Bill, editors, Genetic Programming Theory and Practice V, New York, New York, USA. Springer, pp. 299–314.

[4] Korns, Michael F., and Nunez, Loryfel, 2008. Profiling Symbolic Regression-Classification. In Riolo, Rick, L, Soule, Terrance, and Wortzel, Bill, editors, Genetic Programming Theory and Practice VI, New York, New York, USA. Springer, pp. 215–228.

[5] Korns, Michael F., 2009. Symbolic Regression of Conditional Target Expressions. In Riolo, Rick, L, Soule, Terrance, and Wortzel, Bill, editors, Genetic Programming Theory and Practice VII, New York, New York, USA. Springer, pp. 211–228.

[6] Korns, Michael F., 2010. Abstract Expression Grammar Symbolic Regression. In Riolo, Rick, L, Soule, Terrance, and Wortzel, Bill, editors, Genetic Programming Theory and Practice VIII, New York, New York, USA. Springer, pp. 109–128.

[7] Price, Kenneth, Storn, Rainer, Lampinen, Jouni 2009. Differential Evolution: A Practical Approach to Global Optimization. New York, New York, USA. Springer.

[8] Guido Smits, Ekaterina Vladislavleva, and Mark Kotanchek 2010, Scalable Symbolic Regression by Continuous Evolution with Very Small Populations, in Riolo, Rick, L, Soule, Terrance, and Wortzel, Bill, editors, *Genetic Programming Theory and Practice VIII*, New York, New York, USA. Springer, pp. 147–160.

[9] Flor Castillo, Arthur Kordon, and Carlos Villa 2010, Genetic Programming Transforms in Linear Regression Situations, in Riolo, Rick, L, Soule, Terrance, and Wortzel, Bill, editors, *Genetic Programming Theory and Practice VIII*, New York, New York, USA. Springer, pp. 175–194.

[10] Trent McConaghy, Pieter Palmers, Gao Peng, Michiel Steyaert, Goerges Gielen 2009, Variation-Aware Analog Structural Synthesis: A Computational Intelligence Approach. New York, New York, USA. Springer.

[11] J.A., Nelder, and R. W. Wedderburn, 1972, *Journal of the Royal Statistical Society, Series A, General*, 135:370-384.

[12] John R Koza 1992, Genetic Programming: On the Programming of Computers by Means of Natural Selection. Cambridge Massachusetts, The MIT Press.

[13] Korns, Michael F., 2011. Accuracy in Symbolic Regression. In Riolo, Rick, L, Soule, Terrance, and Wortzel, Bill, editors, Genetic Programming Theory and Practice IX, New York, New York, USA. Springer (*to be published in winter 2011*).

[14] Pham, D., Ghanbarzadeh, A., Koc, E., Otri, S., Rahim, S., and Zaidi, M. 2005. “The Bees Algorithm”. Technical Report Cardiff University.

[15] Parpinelli, R. S., and Lopes, H. S., 2011. New inspirations in swarm intelligence: a survey. *Int Journal of Bio-inspired Computation*. **Vol 3**. Number 1.

[16] Bernstein, J., 2001. Momentum Stock Selection: Using The Momentum Method for Maximum Profits. New York, New York, McGraw Hill

[17] Nicholas, J., 2000. Market-Neutral Investing: Long/Short Hedge Fund Strategies. New York, New York, Bloomberg Press.

[18] Poli, Riccardo, McPhee, Nicholas, Vanneshi, Leonardo, 2009. Analysis of the Effects of Elitism on Bloat in Linear and Tree-based Genetic Programming. In Riolo, Rick, L, Soule, Terrance, and Wortzel, Bill, editors, Genetic Programming Theory and Practice VI, New York, New York, USA. Springer, pp. 91–110.