Large-Scale, Time-Constrained Symbolic Regression

Michael F. Korns

Investment Science Corporation, 1 Plum Hollow, Henderson, Nevada 89052 USA
mkorns@korns.com

Summary. This chapter gives a narrative of the problems we encountered using genetic programming to build a symbolic regression tool for large-scale, time-constrained regression problems. It describes in detail the problems encountered, the commonly held beliefs challenged, and the techniques required to achieve reasonable performance with large-scale, time-constrained regression. We discuss in some detail the selection of the compilation tools, the construction of the fitness function, the chosen system grammar (including internal functions and operators), and the chosen system architecture (including multiple island populations). Furthermore in order to achieve the level of performance reported here, of necessity, we borrowed a number of ideas from disparate schools of genetic programming and recombined them in ways not normally seen in the published literature. [1]

Key words: artificial intelligence, genetic programming, portfolio selection, data mining, formal grammars, quantitative portfolio management

[1] This work was completed with the support of Timothy May who pioneered the Strategy Central Project at Investment Science Corp.

1 Introduction

This is the story of the problems encountered by Investment Science Corporation in using genetic programming techniques to construct a large-scale, time-constrained symbolic regression tool.

Since any in-depth discussion of our financial methods is strictly forbidden by our corporate policy, we introduce our financial motivations very briefly and from these motivations quickly construct the requirements of a generic symbolic regression tool which could be used for our application and for many other large-scale, time-constrained data mining applications.

In the main, the published genetic programming literature addresses problems which are small-scale, without serious time constraints, and often complex. In this historical context, a number of commonly held beliefs and a number of disparate schools of thought have developed in the GP community. In our pursuit of industrial scale performance with large-scale, time-constrained data mining problems, time and again we were required to reexamine many commonly held beliefs and, of necessity, to borrow a number of ideas from disparate schools of genetic programming and recombine them in ways not normally seen in the published literature.

These commonly held beliefs were challenged and these disparate schools of genetic programming were combined into unusual hybrids:

- *Challenged:* "That standard GP parameters (such as population size, crossover rate, etc.) will work for large-scale, time-constrained problems"
- *Combined:* "Hybrid combination of grammar and tree-based GP"
- *Combined:* "Hybrid fitness measure combining value prediction with order prediction in GP symbolic regression"
- *Combined:* "Hybrid combination of multiple island populations and greedy search with tree-based GP"

This narrative follows the pattern of typical applied research projects at Investment Science involving man years of engineering effort and hundreds of experiments. Each of the challenges we faced are explained in detail, and the hybrid solutions required to overcome the challenges are also described in detail. The narrative is punctuated with descriptions of the seminal experiments leading up to the final experiment which produced a generic symbolic regression tool capable of processing one million row by twenty column data mining tables in less than fifty hours on a single workstation computer (specifically an Intel XPS 3.4Ghz Extreme Edition with 800Mhz front side bus and 2GB of 533MHz RAM, running our Analytic Information Server software generating Lisp agents that maximize the on-board Intel registers and on-chip vector processing capabilities).

1.1 Financial Motivation

We introduce our financial motivations only very briefly. From these motivations we construct the requirements for a generic symbolic regression tool which can be used not only for our financial application but for many other large-scale, time-constrained data mining applications.

Consider a quantitative (quant) trading system for the top 800 exchange-traded common stocks with the largest dollar-volume traded in the prior week (YCK04a and CB04a). These securities are so active that we will be able to move millions of dollars in and out of these investments without appreciably perturbing their prices. We will retrain the system weekly using a sliding training window of five years or 1,250 training days of historical data (YCK04a). This allows relatively frequent system retraining (weekly) while providing a relatively long retraining period of fifty hours (the weekend).

Our first issue is selecting a basket of 20 sample column data points from the over 500 available data points such as Open, High, Low, Close, Volume, EPS, Analyst Rating, etc. We solve this issue by implementing multiple independent trading systems. For instance one might have a value trading system with one set of 20 training points, a growth trading system with another set of training points, and a chartist trading system with yet another set of training points. If one has a farm of 100 workstations, each workstation could retrain each of 100 independent trading systems once per week.

Our second issue is the large volume of data which is fed to the system on every retraining period (1,250 historical daily samples for each of 800 common stocks is 1,000,000 rows of training data by 20 columns). The solution to this second issue is the subject of this paper as we pursue the construction of a generic regression tool which can perform a single 1,000,000 row by 20 column symbolic regression in less than 50 hours on a single workstation computer. Clearly such a tool would prove useful not only in our own financial application but in many other large-scale, time-constrained applications.

1.2 Experimental Setup

All our experiments are scored on testing data sets different from the data sets they were trained on. We have crafted nine separate test cases (formulas), from simple to complex. All of our test cases are trained on one million row by M column randomly generated training matrices (where M is either 1, 5, or 20). Then a separate randomly generated one million row by M column testing matrix is used for scoring. All of our nine test case formulas are shown below (generated with five columns).

Test Case Formulas

linear
```
y = 1.57 + (1.57*x0) - (39.34*x1) +
        (2.13*x2) + (46.59*x3) + (11.54*x4);
```

hidden
```
y = 1.57 + (2.13*sin(x2));
```

cubic
```
y = 1.57 + (1.57*x0*x0*x0) -
        (39.34*x1*x1*x1) + (2.13*x2*x2*x2) +
        (46.59*x3*x3*x3) + (11.54*x4*x4*x4);
```

elipse
```
y = 0.0 + (1.0*x0*x0) + (2.0*x1*x1) + (3.0*x2*x2) +
        (4.0*x3*x3) + (5.0*x4*x4);
```

hyper
```
y = 1.57 + (1.57*tanh(x0*x0*x0)) -
        (39.34*tanh(x1*x1*x1)) + (2.13*tanh(x2*x2*x2)) +
        (46.59*tanh(x3*x3*x3)) + (11.54*tanh(x4*x4*x4));
```

cyclic
```
y = 14.65 + (14.65*x0*sin(x0)) -
        (6.73*x1*cos(x0)) - (18.35*x2*tan(x0)) -
        (40.32*x3*sin(x0)) - (4.43*x4*cos(x0));
```

cross
```
y = -9.16 - (9.16*x0*x0*x0) -
        (19.56*x0*x1*x1) + (21.87*x0*x1*x2) -
        (17.48*x1*x2*x3) + (38.81*x2*x3*x4);
```

mixed
```
if ((mod(x0,4) == 0)
    {
    y = (1.57*log(.000001+abs(x0))) -
            (39.34*log(.000001+abs(x1))) +
            (2.13*log(.000001+abs(x2))) +
            (46.59*log(.000001+abs(x3))) +
            (11.54*log(.000001+abs(x4)));
    }
else
if ((mod(x0,4) == 1)
    {
    y = (1.57*x0*x0) - (39.34*x1*x1) +
            (2.13*x2*x2) + (46.59*x3*x3) +
            (11.54*x4*x4);
    }
else
if ((mod(x0,4) == 2)
    {
    y = (1.57*sin(x0)) - (39.34*sin(x1)) +
        (2.13*sin(x2)) + (46.59*sin(x3)) +
        (11.54*sin(x4));
    }
else
if ((mod(x0,4) == 3)
    {
    y = (1.57*x0) - (39.34*x1) +
        (2.13*x2) + (46.59*x3) +
        (11.54*x4);
    }
```

ratio

```
if ((mod(x0,4) == 0)
    {
    y = ((1.57*x0)/(39.34*x1)) +
            ((39.34*x1)/(2.13*x2)) +
            ((2.13*x2)/(46.59*x3)) +
            ((46.59*x3)/(11.54*x4));
    }
else
if ((mod(x0,4) == 1)
    {
    y = ((1.57*x0)
            ((39.34*x1)
            ((2.13*x2)
            ((46.59*x3)
    }
else
if ((mod(x0,4) == 2)
    y = ((1.57*sin(x0))/(39.34*tan(x1))) +
            ((39.34*sin(x1))/(2.13*tan(x2))) +
            ((2.13*sin(x2))/(46.59*tan(x3))) +
            ((46.59*sin(x3))/(11.54*tan(x4)));
    }
else
if ((mod(x0,4) == 3)
    y = 0.0 - (39.34* log(.000001+abs(x1))) +
            (2.13* log(.000001+abs(x2))) +
            (46.59*log(.000001+abs(x3))) +
            (11.54* log(.000001+abs(x4)));
    }
```

Our nine test cases vary from simple (linear) to complex (formulas with embedded if-then-else expressions). Finally, to add difficulty, we sometimes train and test our nine test cases with random noise added using the following formula.

```
;; Modify each y in Y or each ty in TY with random noise.
y = (y * .80) + (y * random(.40));
```

The addition of random noise makes each test case inexact and theoretically undiscoverable. Nevertheless, given our application, we need to test our symbolic regression tool against inexact data.

1.3 Fitness Measure

- *Combined:* "Hybrid fitness measure combining value prediction with order prediction in GP symbolic regression compiler"

Standard regression techniques utilize least squares error as a fitness measure; however, we desire a high correlation between the order of our estimates, EY, and our actual values, Y. Let us define a new term *sequence* as follows: If our estimates, EY, *sequence* our actual values, Y, perfectly, then it is true that, if EY[1] <= EY[2] then Y[1] <= Y[2] for all ey in EY and all y in Y. If we produce a perfect regression, with a percent error of zero, then EY will always *sequence* Y perfectly. Unfortunately, if the regression percent error is less than perfect, EY may not sequence Y very well at all. Furthermore, two regression formulas, with equal percent error, may sequence Y differently. Clearly, considering our end application, we are interested in regression solutions which sequence Y as well as possible. In fact, even if the regression percent error is poor but the sequencing is good, we can still have an advantage, in the financial markets, with our symbolic regression tool.

We combined a normalized average percent error score with an order preservation score to produce an optimal fitness measure for our financial application.

```
(loop for n from 0 until N do
        (setq avgDifY (+ avgDifY (/ (abs (- Y[n] avgY)) N))))
(loop for n from 0 until N do
        (setq errPct (+ errPct (/ (abs (- EY[n] Y[n])) N avgDifY))))
```

We measure the sequencing of Y by EY by counting how many Y pairs are out of order when sorted by EY.

```
(setq sortedEY (sort EY < true))
(loop for n from 0 until N do
    (setq k1 sortedEY[n]) (setq k2 sortedEY[(+ n 1)])
    (if (> Y[k1] Y[k2]) then (setq seqErr (+ seqErr (/ 1 N)))))
```

We now have a sequencing measure (seqErr) which varies from 0% (perfect) to 100% (terrible) and a normalized average percent error (errPct). We construct our final fitness measure using both as follows.

```
(setq fitness (+ errPct (* .001 seqErr)))
```

1.4 Multiple Grammars

- *Combined:* "Hybrid combination of grammar and tree-based GP"

Recently, informal and formal grammars have been used in genetic programming (ONeil03) to enhance the representation and the efficiency of a number of applications including symbolic regression. Much of the published literature on grammar-based GP focuses on alternative genome representations and evolutionary operators. After extensive experimentation, in our application domain, we discovered that alternative genome representations and evolutionary operators provided less added value than the use of multiple grammars themselves.

Therefore we settled on a hybrid combination of tree-based GP and formal grammars where the head of each sublist is a grammar rule agent with polymorphic methods for mutation, crossover, etc. Different grammar rules communicate with each other by message passing (a staple of object-oriented and agent-oriented software engineering), described in more detail as follows.

Formal grammars are a staple of computer science and are widely used in industry (Aho86). We decided, from an engineering perspective, to allow our symbolic regression tool to use multiple grammars and even to mix grammars during the symbolic regression process.

There are two sides to every formal grammar: the recognition side, and the production side. For grammar recognition we used a feature-based compiler compiler which reads a set of feature-based grammar rules. We constructed our symbolic regression tool as a large agent complex, and embedded in the symbolic regression tool a formal grammar definition file (readable by the feature-based compiler-compiler agent). When the feature-based compiler-compiler agent is pointed at the embedded formal grammar definition rules, a child agent compiler for the specified grammars is automatically generated inside the symbolic regression tool. For the production side of each formal grammar, we generated grammar production agents embedded in the symbolic regression tool.

There are three important grammars in the system. These are: a basic JavaScript-like numeric expression grammar for generating fast register and pointer integer and floating point computations (EXP); a standard regression grammar with a single s-expression genome (REG); and a multiple variable regression grammar with a multiple s-expression genome (MVL). The embedded feature-based grammar rules support recognition of each of these grammars even when mixed in the same well-formed formula (WFF). Grammar production is accomplished with embedded agent grammar complexes (one for each grammar), with matching polymorphic methods allowing each grammar to communicate with the other grammars for operations such as crossover.

We use standard Tree based genetic programming techniques with the exception that the head of every Lisp s-expression is a grammar production rule; also, the tail of each s-expression contains the arguments required by the grammar production rule. We use standard mutation and crossover operations (Koz92i) in our numeric expression EXP grammar. The REG grammar has only one s-expression in its chromosome so REG operates identically to the EXP grammar (using standard tree-based mutation and crossover). The multiple s-expression grammar (MVL) selects a random numeric s-expression chromosome, and performs standard tree-based mutation on the selected s-expression.

Details of each of the grammars are quite straight forward and can best be conveyed from the recognition side.

- *REG Grammar:* regress EXP ;
- *MVL Grammar:* mvlregress(EXP,EXP,...,EXP);

The weight-vectors, used as arguments to the ENN grammar rule, are standard numeric vectors whose length is determined by the number of columns in the regression problem. The numeric s-expressions, of the EXP grammar, are standard JavaScript-like numeric expressions with the variables *x0* through *xm* (where m is the number of columns in the regression problem), and with the following binary and unary operators + - / % * < <= == ! = >= > expt max min abs cos cosh cube exp log sin sinh sqroot square tan tanh. To these we add the ternary conditional expression operator *(...) ? ... : ... ;*

1.5 Overview of Symbolic Regression Tool

At Investment Science Corporation we have constructed a large agent complex for high volume symbolic regression applications consisting of one million rows and from five to twenty columns. Due to the heavy resources required to evaluate a candidate well-formed-formula (WFF) across one millions rows, we cannot afford to evaluate the same candidate twice. Therefore, every WFF which we have ever evaluated is saved during the course of a single training cycle. All WFF candidates are saved in a collection sorted by their fitness scores. A user option setting restricts the survivor WFF population to the "F" most fit WFF candidates. User option settings support single or multiple island populations and other potentially usefully clustering of candidate

WFFs. Parenthetically, all the tool's user option settings are available at run time; therefore, it might be possible for the tool to evolve itself, although we have not attempted anything of that nature.

Within the survivor population, mutation and crossover occur in the same fashion as with standard genetic programming. Each WFF survivor is visited once per each evolution. A user-option determines the probability of mutation and another determines the probability of crossover. If warranted, Crossover occurs between the visited individual and a randomly selected individual, from the survivor population, with less fitness than the visited individual. The tool supports multiple grammars in the same training cycle as described previously.

Standard genetic programming practice encourages the use of multiple independent training *runs*. Each run incorporates one initialization step and "G" generational steps during which evolutionary operators are applied. It is standard practice for the experimenter to perform multiple independent training runs of G generations each and then report the results of the fittest individual evolved across all runs (the champion individual).

Since our symbolic regression tool is to be used in a fully automated setting, their can be no human intervention to decide how many independent training runs to perform; therefore, the concept of automatic multiple independent training runs has been incorporated into the tool. A user-option determines the number of evolutions "without fitness improvement" which trigger the system to start a new independent training run. After each independent training run, the best-of-breed champions from the previous run are saved and the training cycle restarts from scratch. Thus a training cycle of "G" generations may involve more or fewer separate independent training runs depending on the occurrence of long gaps without fitness improvement.

In every experimental report, the following standard table of results will be provided for each of the nine test cases.

- *Test:* The name of the test case
- *Gens:* The number of generations used for training
- *Minutes:* The number of minutes required for training
- *Train-Error:* The average percent error score for the training data
- *Test-Error:* The average percent error score for the testing data
- *Sorting:* The sequencing score for the testing data

1.6 Standard GP using the REG Grammar

- *Challenged:* "That standard GP parameters (such as population size, crossover rate, etc.) will work for large-scale, time-constrained problems"

Our first experiment was to use the REG grammar, with its very fast training time, for a straight forward approach to large scale symbolic regression. Our basic algorithmic components are WFFs, a memo cache of WFFs, the survivor population of the fittest WFFs, and a list of champion WFFs. We used the *GPSR* option settings which are almost directly in line with (Koz92i) and are as follows. At the initialization step of every training "run", exactly 5000 randomly generated WFFs, in the REG grammar, are evaluated. All evaluated WFFs are memoized (saved in a memo cache) and also saved in sorted order by fitness score (in the survivor population). The top twenty-five WFFs participate in the genetic operations of mutation and crossover (Koz92i) during each incremental generation. The probability of mutation is 10%, and the probability of crossover is 100%. When a WFF is chosen for crossover, its mate is chosen at random from the WFFs of lower fitness within the top twenty-five fittest individuals (in the survivor population). Crossover is always performed twice with the same parents and always produces two children which are evaluated, memoized, and saved in sorted order by fitness score (in the survivor population). The maximum number of generations before training halts is provided at the start of training time. If ten generations pass with no change in the fittest WFF, then system saves the fittest WFF in its list of champions, clears all WFFs in the survivor population (but not the memo cache) and evaluates 5000 randomly generated WFFs, starting a new "run". Any new "run" does not reset the generation count. Training always proceeds until the maximum number of generations have been reached. If G represents the maximum number of generations allowed for a fully automated training cycle, then the maximum number of independent "runs" is (G/10). Depending upon the progress in training, there may only be a single "run" during the entire training process. At the completion of training, the fittest champion WFF (the fittest WFF ever seen) is chosen as the result of the training process. The results of training on the nine test cases, using the GPSR option settings over 1 million rows and from one to five columns, returned near perfect results. The test results, with no random noise, demonstrated that more or less standard GP parameters are a great choice for one million row by one column problems even given the meager training time allowed.

Unfortunately when we increase the number of columns from one to twenty, the test results do not show the same level of excellence. We are doing very well on the simpler problems; but, the more complicated test cases are exceeding the allocated fifty hour maximum training time in only 100 generations. Clearly the hope that more or less standard GP settings will work for all large-scale, time-constrained problems has been challenged.

Thinking about our next step, we wonder could the problem be that we need more generations to train on the more complicated test cases? We decide that our next step will be to find some method of increasing the speed of the training process to allow 100 generations for training within the allocated maximum of fifty hours.

1.7 Vertical Slicing

Let us define a new procedure *Vertical slicing* as follows. First, the rows in the training matrix X are sorted in ascending order by the dependent values, Y. Then the rows in X are subdivided into S *vertical slices* by simply selecting every Sth row to be in each vertical slice. Thus the first vertical slice is the set of training rows as follows X[0], X[S], X[2*S], ... Each vertical slice makes no assumptions about the underlying probability distribution and each vertical slice contains evenly distributed training examples, in X, across the entire range of *ascending) dependent values, in Y.*

After several months of study and experimentation, including using support vector machines to select training example subsets, we selected vertical slicing as the most expedient method of decreasing training time. Vertical slicing works well because it is extremely simple, makes no assumptions about the underlying probability distribution function, and is extremely fast.

Vertical slicing reduces training time by subdividing the training data into "vertical slices" each of which is representative of the whole training data set over ascending values of Y. We may subdivide the training data into as many vertical slices as practicable. The larger the number of slices, the fewer training examples are in each slice.

Our next step is to randomly select one of the vertical training data slices as our "sample" training slice; furthermore, we modify each agent WFF and the memo cache to record the sample-fitness. There are now two different fitness scores for each WFF: sample-fitness and fitness. During evaluation, each WFF is first scored on the "sample" training slice and its sample-fitness is recorded. Next the sample-fitness of the WFF is compared to the sample-fitness of the least-fit WFF in the survivor population. If the sample-fitness of the WFF is greater than or equal to the sample-fitness of the least-fit WFF in the survivor population, the WFF is then scored against the entire training data and its true fitness is recorded. This approach will produce false negatives but no false positives.

Our next experiment was to use the REG grammar, with the RGPSR *option settings which are exactly like the* GPSR *option settings (discussed in the previous experiment) except that the vertical slicing option is turned on. The number of vertical slices is set to 100. The results of training on the nine test cases, using the RGPSR option settings over 1 million rows and 20 columns show that the vertical slicing improves the training speed of standard GP but does not improve the regression accuracy. Now that we have the training time within our maximum limit, how can we improve accuracy?*

1.8 Using the MVL Grammar

After several additional months of study and experimentation we decided to replace the REG grammar with the MVL grammar, and to modify the standard crossover operator to include some non-standard hill-climbing. The REG grammar, being a single expression regression, can be computed and optimized in the single pass required to compute the average percentage error. Unfortunately, the MVL grammar requires significant additional computational effort. Given an MVL grammar candidate, mvlregress(exp1,exp2,...,expM)*, a multivariate regression must be performed in order to both provide optimized coefficients and to compute the average percentage error. In preparation for the multivariate regression we create an intermediate N row by M column transform matrix, 'X, where each expression, exp1 thru expM, is the transform creating the 'x0 thru 'xm transform variables. The transform matrix, 'X, and the dependent variable vector, Y, are then used to represent a system of linear equations,* C*'X = Y*, which we use the* Gaussian elimination *algorithm (Sedge88) to compute the optimized coefficients and the error.*

We experimented with both the standard s-expression crossover operator in (Koz92i) as well as the linear string-based codon style crossover operator of (ONeil03). It is possible algorithmically to linearize any tree-based grammar rule s-expression into a linear string-based codon style genome and vice-versa. Primarily because we are employing grammar based s-expressions, with a distinct separation between genotype and phenotype, we found each style of crossover operator to be different, yet to be relatively equivalent, with respect to performance in our high speed, high volume application. However, we found that adding a small amount of hill-climbing mutation *to our crossover operator improved our regression performance. Given two parent grammar rule s-expression WFFs,*

'(ruleAdd (ruleSub x0 34.5) ***(ruleCos x4)****),*

and

'(ruleMul ***(ruleDiv x0 x5)*** *(ruleAdd x4 x1)),*

the standard crossover operator would use substitution to swap the selected subexpressions as follows:

'(ruleAdd (ruleSub x0 34.5) ***(ruleDiv x0 x5)****)*

We found that we could improve our performance by, with a probability of 20%, averaging the two selected subexpressions as follows:

'(ruleAdd (ruleSub x0 34.5) ***(ruleAvg (ruleCos x4) (ruleDiv x0 x5))****)*

In addition to the ruleAvg *grammar operator, we also used, with equal probability, the following grammar rules to combine the two selected subexpressions:* ruleMax*,* ruleMin*,* ruleAdd*,* ruleSub*,* ruleMul*, and* ruleDiv*. Once* *again, our hill-climbing crossover operates as a standard crossover operator with the exception that in 20% of the cases we average the two selected subexpression as shown above.*

Our next experiment was to use the MVL grammar, with the RMVL *option settings which are exactly like the* RGPSR *option settings (discussed in the previous experiment) except that the MVL grammar is substituted for the REG grammar and our enhanced hill-climbing crossover operator is substituted for the standard crossover operator. The results of training on the nine test cases, using the RMVL option settings over 1 million rows and 20 columns, showed an increase in performance in both average percent error and data sequencing. Unfortunately, we suffer an over all increase in computation time to the point that many of our test cases exceed the maximum allocated 3000 minutes (50 hours). Clearly we need to make additional enhancements to reduce computation time in cases with twenty columns.*

1.9 Island GP using Multiple Grammars

- Combined: *"Hybrid combination of multiple island populations and greedy search with tree-based GP"*

After several additional months of study and experimentation we decided to make the following additional enhancements to increase performance and reduce computation time in cases with twenty columns. We introduced "Double Vertical Slicing" and multiple island populations using the REG and MVL Grammars.

Our first alteration induces genetic diversity via multiple independent island populations. There is a correlation between the fittest individuals in a training run and genetic diversity in the population (Almal06). Furthermore the fittest individuals in a training run tend to cluster around a set of common root expressions (Daida05) and (HS04a). Capitalizing on these observations, we set our symbolic regression tool's options to create additional independent island populations, one for each column in the problem.

*During the initialization step in every training run we generate the first 5000 possible root expressions sequentially in every independent island population. We sequentially examine every possible root expression with relevance to each column. A root expression has relevance to a column if and only if the column name appears, at least once, in the root expression. For example, the WFF "*x0*" has relevance to column zero (hence independent island population zero), while the WFF "*x4*sin(x8)*" has relevance to column four and column eight.*

In each independent island population using the REG grammar, we generate 5000 root expressions (ignoring real constants and the trinary conditional expression), such that in each island population each root expression is guaranteed to have at least one reference to the name for that column x0 *thru* xm.

This forces some genetic diversity and focuses each island population around the island's designated column.

Since the sequentially generated root WFFs are sorted in each island population by fitness, we have performed a defacto greedy search and made some guesses, by column, as to which grammar root expressions will be the most indicative of future evolutionary fitness. The above initialization step we call "root initialization" and it places 5000 sequentially generated REG grammar WFFs in each of M independent island populations sorted by fitness (the same number of islands as columns). It is performed once during the initialization step of every training run.

Our second alteration induces genetic diversity in the main population by borrowing information from each multiple independent island population. In the initialization step of every training "run", after the root initialization, exactly 5000 randomly generated WFFs, using the MVL grammar, are evaluated in the main population. Each of the M numeric expressions in the MVL grammar mvlregress(exp0,...,expM) *are taken at random from the 25 fittest individuals in the respective independent island populations. These are the fittest root expressions generated in each independent island population during "root initialization". This modified initialization process induces genetic diversity in the main population and guarantees that each of the MVL grammar's M numeric expressions will have at least one reference to the designated terminal for that column x0 thru xm. All evaluated WFFs are memoized and saved in sorted order by fitness score. The top twenty-five WFFs, in the main population, participate in the genetic operations of mutation and crossover (Koz92i) during each incremental generation. The probability of mutation is 10%, and the probability of crossover is 100%. The hill-climbing modifications to the crossover operator, introduced in the RMVL option settings, are also employed here.*

Our third alteration decreases training time by enhancing the vertical slicing algorithm used to measure the fitness of each WFF. In the RMVL option settings, we selected a vertical sample data set for sampling and then trained on the full data set. Here we introduce "double vertical slicing" by selecting a vertical slice for sampling and another vertical slice for training. By not training against the full training data set we further reduce training time. The number of vertical slices is set to 100 for the sample set and is set to 10 for the training set. Scoring each individual WFF's fitness is always done on the entire training data set (defacto out-of-sample scoring). This approach will produce false negatives but no false positives.

Our fourth alteration introduces a "tournament of champions". In the RMVL option settings, any time there are 10 generations with no fitness improvement a new training run is started. In the "tournament of champions" enhancement, every 5 training runs, migration occurs between the pool of champions (from all previous training runs) and the main population after all initialization. This allows previous champions to compete in the normal evolutionary process but only once in every few training runs. This approach *will produce greater accuracy but without sacrificing the independence of most of the training runs.*

Our final experiment was to use the REG and MVL grammars, with the RGMVL option settings but we introduce random noise of 40%. The results of training on the nine test cases, using the RGMVL option settings over 1 million rows and twenty columns with 40% random noise, are shown in the table below.

Table 1. RGMVL Options 1M rows by 20 columns Random Noise

Test	*Gens*	*Minutes*	*Train-Error*	*Test-Error*	*Sorting*
cross	100	1950	0.831	0.901	0.396
cubic	100	702	0.399	0.392	0.005
hyper	100	2077	0.852	0.896	0.111
elipse	100	910	0.705	0.662	0.156
hidden	100	828	0.110	0.015	0.112
linear	100	708	0.102	0.016	0.001
mixed	100	896	0.676	1.210	0.037
ratio	100	758	0.309	0.942	0.145
cyclic	100	738	0.433	0.511	0.884

Description of table headings:

- Test: *The name of the test case*
- Gens: *The number of generations used for training*
- Minutes: *The number of minutes required for training*
- Train-Error: *The average percent error score for the training data*
- Test-Error: *The average percent error score for the testing data*
- Sorting: *The sequencing score for the testing data*

Fortunately training time is well within our 3000 minute (50 hour) limit. In general average percent error performance is poor with the linear *and* hidden *problems showing the best performance. Extreme differences between training error and testing error in the* mixed *and* ratio *problems suggest over fitting. Surprisingly, order prediction is fairly robust in most cases with the exception of the* cyclic *and* cross *test cases. Furthermore order prediction is fairly robust even on many test cases where average percent error is poor.*

1.10 Summary

Genetic Programming, from a rigorous corporate perspective, is not yet ready for industrial use on large scale, time constrained symbolic regression problems. However, adapting the latest research results, has created a symbolic regression tool whose promise is exciting. Academic interest in the field is growing while pure research continues at an aggressive pace. Further applied research in this field is absolutely warranted.

We favor the following research areas for further experimental study. Experiments using standard statistical analysis to increase genetic diversity and information flow between independent island populations and the main population. Experiments automating Pareto front analysis to improve selection of promising WFF root grammar candidates. Additional experimentation with grammar expressions to increase the speed of evolutionary training and to develop a better understanding of hill-climbing operators from a root grammar viewpoint.

References

1. Sedge88 *"Algorithms", Robert Sedgewick; Addison-Wesley Publishing Company; 1988.*
2. Aho86 *"Compiler Principles, Techniques, Tools", Alfred V Aho, Ravi Sethi, Jeffery D. Ullman; Addison-Wesley Publishing; 1986.*
3. Daida05 *"Considering the Roles of Structure in Problem Solving by a Computer", in Genetic Programming Theory and Practice II, J Daida; Springer, New York; 2005.*
4. Almal06 *"Content Diversity in Genetic Programming and Its Correlation with Fittness", in Genetic Programming Theory and Practice III, A Almal, WP Worzel, E A Wollesen, C D MacLean; Springer, New York; 2006.*
5. WL00c *"Datamining using Grammar based Genetic Programming and Applications" Man Leung Wong, Kwong Sak Leung; Kluwer Academic Publishers, Dordrecht Netherlands; 2000.*
6. YCK04a *"Discovering Financial Technical Trading Rules Using Genetic Programming with Lambda Extraction" in Genetic Programming Theory and Practice II, Tina Yu, Shu-Heng Chen, Tzu-Wen Kuo; Springer, New York; 2005.*
7. HS04a *"Does Genetic Programming Inherently Adopt Structured Design Techniques?" in Genetic Programming Theory and Practice III, John Hall, Terence Soule; Springer, New York; 2006.*
8. Chen02 *"Genetic Algorithms and Genetic Programming in Computational Finance" edited by Shu-Heng Chen; Kluwer Academic Publishers, Dordrecht Netherlands; 2002.*
9. Koz92i *"Genetic Programming: On the Programming of Computers by Means of Natural Selection" John R. Koza; The MIT Press, Cambridge Massachusetts; 1992.*
10. Koz94i *"Genetic Programming II: Automatic Discovery of Reusable Programs" John R Koza; The MIT Press, Cambridge Massachusetts; 1994.*
11. Koz99i *"Genetic Programming III: Darwinian Invention and Problem Solving" John R Koza, Forrest H Bennett III, David Andre, Martin A Keane; Morgan Kaufmann Publishers, San Francisco, California; 1999.*
12. Koz03i *"Genetic Programming IV: Routine Human-Competitive Machine Intelligence" John R Koza, Martin A Keane, Mathew J Streeter, William Mydlowec, Jessen Yu, Guido Lanza; Kluwer Academic Publishers, Dordrecht Netherlands; 2003.*
13. ONeil03 *"Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language" Michael O'Neill, Conor Ryan; Kluwer Academic Publishers, Dordrecht Netherlands; 2003.*
14. CB04a *"Lessons Learned Using Genetic Programming in a Stock Picking Context" in Genetic Programming Theory and Practice II, Michael Caplan, Ying Becker; Springer, New York; 2005.*
15. Kle67 *"Mathematical Logic" Stephen Klenne; John Wiley and Sons, New York; 1967.*
16. SK04b *"Pareto-Front Exploitation in Symbolic Regression" in Genetic Programming Theory and Practice II, Guido Smits, Mark Kotanchek; Springer, New York; 2005.*