# Geometric Semantic Genetic Programming for Financial Data

James McDermott[1,2(✉)], Alexandros Agapitos[1], Anthony Brabazon[1,3], and Michael O'Neill[1]

[1] Natural Computing Research and Applications Group, Complex and Adaptive Systems Lab, University College Dublin, Dublin, Ireland
jmmcd@jmmcd.net, {alexandros.agapitos,anthony.brabazon,m.oneill}@ucd.ie
[2] Management Information Systems, Lochlann Quinn School of Business, University College Dublin, Dublin, Ireland
[3] Accountancy, Lochlann Quinn School of Business, University College Dublin, Dublin, Ireland

**Abstract.** We cast financial trading as a symbolic regression problem on the lagged time series, and test a state of the art symbolic regression method on it. The system is geometric semantic genetic programming, which achieves good performance by converting the fitness landscape to a cone landscape which can be searched by hill-climbing. Two novel variants are introduced and tested also, as well as a standard hill-climbing genetic programming method. Baselines are provided by buy-and-hold and ARIMA. Results are promising for the novel methods, which produce smaller trees than the existing geometric semantic method. Results are also surprisingly good for standard genetic programming. New insights into the behaviour of geometric semantic genetic programming are also generated.

**Keywords:** Automated trading · Commodity · Exchange rate · Index · Genetic programming · Semantics · Fitness landscape · Hill-climbing

## 1 Introduction

Trading on financial markets is an important problem in its own right, and an interesting and difficult test problem for machine learning methods. It is a source of unending difficulty because of feedbacks between traders: each trader changes the environment for others, so no particular solution can win in the long run.

It is common to cast trading as a regression problem, where the goal is to predict the next price in a time series in terms of the current price and some lagged prices. Trading proceeds by interpreting negative predictions as short signals and positive predictions as buy signals.

A standard method of time-series modelling is ARIMA, the auto-regressive integrated moving average [11]. ARIMA and related methods use linear combinations of the lagged time-series. The autocorrelation is used to indicate which lags contain significant information, in the form of linear correlations with the


A.I. Esparcia-Alcázar et al. (Eds.): EvoApplications 2014, LNCS 8602, pp. 215–226, 2014.
DOI: 10.1007/978-3-662-45523-4_18

present value. If it is hypothesized that even lags which are not significantly correlated with the present value may contain information which can by taken advantage of by using them non-linearly and in combination, then there is a motivation for using other non-linear regression methods.

Genetic programming symbolic regression (GPSR) is an example. In comparison to a method like ARIMA, GPSR is more flexible, because it allows nonlinear combination of variables. However it is less reliable, due to its stochastic nature. For time series with simple structure, ARIMA will generally be faster and more reliable, and will produce a simpler and more readable model. For more complex time series, GPSR has the potential to out-perform ARIMA.

In recent years, several advances have been made in the state of the art in GPSR. One example is the *geometric semantic genetic programming* (GSGP) approach of Moraglio et al. [9], described in detail in Sect. 3.1. A geometric semantic mutation operator causes the fitness landscape to become a cone, easily searched using hill-climbing. However the GSGP operators bring about a very large increase in the size of the trees produced, so it is interesting to consider variations which avoid creating such large trees while retaining the geometric property. We propose two new mutation operators (see Sect. 3.2). The first, *one-tree* GSGP mutation, is very similar to the standard GSGP mutation operator, but adds less genetic material at each mutation step, helping to keep trees small. The other, *optimal-step* GSGP mutation, also uses this idea, and also chooses an optimal mutation step-size at each step: this may allow the search to approach the optimum much faster, requiring fewer steps, so that again the tree eventually produced has accumulated fewer nodes.

It is interesting to test GSGP in financial trading because it has not yet, to our knowledge, been tested on financial data or on any type of time-series modelling. In this paper we compare ARIMA, GSGP, and a standard GP hill-climber. We run our tests over 3 datasets of 1400 points each, derived from Gold, GBP/USD, and S&P500 markets.

Sect. 2, next, describes some related work. The GSGP methods are described in Sect. 3. Experiments and results are given in Sect. 4; Sect. 5 analyses these results; and Sect. 6 gives conclusions and future work.

## 2 Related Work

Many authors have used evolutionary methods, and particularly GP, for financial trading: see [2,3] and references therein. Previous work has shown the potential benefit of exploring new GP representations in particular [1]. Out-performing a buy-and-hold strategy was found to be surprisingly difficult by several authors as described by Lohpetch and Corne [5]. They can more reliably out-perform buy-and-hold when trading at a monthly level, with less reliability when trading daily. They do not attempt 5-minute trading as in the current paper.

The GSGP method was developed by Moraglio et al. [9]. It is rooted in the unifying theory of geometric operators [8]. A geometric mutation operator produces new individuals distributed in a ball surrounding the original. A geometric

crossover operator produces children distributed in the line segment between the two parents. The radius of the ball, and the line segment, are defined using a suitable metric. In the case of geometric *semantic* GP, the metric is on the semantic space of programs. In this space, each element is a vector of the outputs from some program on the vector of fitness cases. The key achievement of Moraglio et al. [9] in relation to GPSR is to define mutation and crossover operators which are geometric according to Euclidean distance on the semantic space. Because symbolic regression fitness is equivalent to Euclidean distance from the target in the semantic space, the fitness landscape becomes a cone. This means that search will encounter no local optima and can proceed reliably and efficiently, an important step forward for GP.

In fact, the absence of local optima, and the consensus of previous results [9, 12], suggests that a hill-climber is a sufficient search algorithm: a population and a crossover operator are unnecessary. Standard GP, using the subtree mutation operator, also encounters no local optima. For this paper, then, we consider mutation and hill-climbing only.

GSGP has previously been used for symbolic regression on real-world data [12]. However, the results achieved there were no better than predicting a constant for all data points[1]. On the other hand, the fact that the landscape becomes a cone promises very good performance; and results on randomly-generated symbolic regression problems have been very good [9].

One disadvantage of the GSGP operators is that they bring about a very large increase in the size of the trees produced. The mutation operator adds two random trees plus four nodes to the tree at each step. The new variants proposed in the following sections aim to mitigate this.

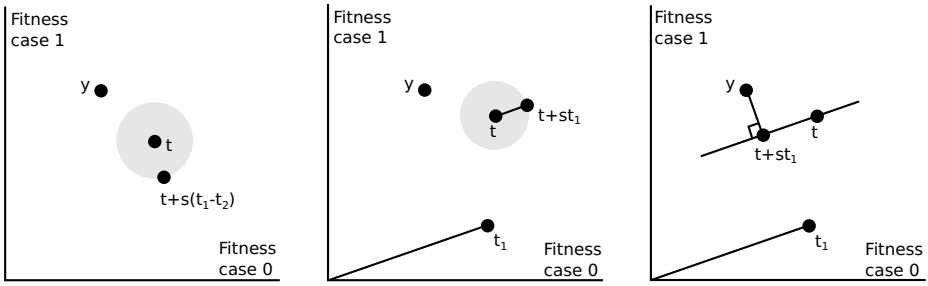
## 3 GP, GSGP, and Variations

A key concept in understanding GSGP is *semantic space*. Individuals' values on the vector of fitness cases give their position in semantic space. For example, consider a dataset of three input variables $x_0$, $x_1$, and $x_2$, and one output variable $y$, with 2 fitness cases:

| | $x_0$ | $x_1$ | $x_2$ | $y$ |
|---|---|---|---|---|
| Fitness case 0 | 3 | 4 | 1 | 10 |
| Fitness case 1 | 7 | 8 | 2 | 12 |

Consider an individual `(* x0 x1)`. Its values on the two fitness cases are $(12, 56)$. These two values give the individual's position in the 2-dimensional semantic space, which is depicted in Fig. 1. The target $y = (10, 12)$ is also a point in semantic space.

[1] For example, for a problem in predicting the *bioavailability* of certain drugs [12], the mean of the target values on all fitness cases is approximately 66.4%. Predicting this value for all fitness cases produces a fitness value of 30.4%.




GSGP One-tree Optimal-step

**Fig. 1.** GSGP and variations. Individuals are shown in the semantic space. In a dataset of two fitness cases, an individual's values on the two cases give its position in this space. In GSGP, left, the tree resulting from a mutation is expected to lie in a ball surrounding the original (in semantic space). The same is true of *one-tree* GSGP, centre. In *one-tree optimal-step* GSGP, right, the optimal value of the mutation step size $s$ is found, in order to scale the effect of the new random tree and bring the resulting tree as close to the optimum as possible. The new random tree $t_1$ is seen as a vector with a fixed direction. Changing the scalar moves the resulting tree back and forth along a line through $t$ parallel with that vector. Choosing the optimal $s$ brings the new point as close as possible to the target $y$.

### 3.1 GSGP

The GSGP mutation operator for symbolic regression problems [9] works by taking the difference of two randomly-generated trees $t_1$ and $t_2$, scaled by a positive constant $s$ giving the step-size, and combining that with the original tree $t$, to give a new tree $t_{\text{new}}$ as shown in Fig 2.

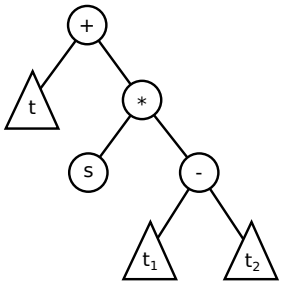



**Fig. 2.** The new tree produced by the GSGP mutation operator: $t_{\text{new}} = t + s(t_1 - t_2)$

In the semantic space, the new individual is distributed in a ball of radius $s$, because for each dimension of the semantic space, the added tree $(t_1 - t_2)$ has expected value 0. See Fig. 1. One disadvantage of the operator, mentioned in previous work [12], is that the resulting tree grows by $4 + |t_1| + |t_2|$ nodes per step, where $|\cdot|$ indicates the number of nodes in a tree. The 4 comes from the $s$, $+$, $-$, and $*$ nodes.

Running for many generations then results in very large trees, which have some disadvantages. They are for practical purposes unreadable. They require very large CPU and memory resources, though correct implementation can mitigate this issue: Moraglio et al. [9] avoid storing the trees themselves by restricting attention to a space of polynomials in which simplification of large trees is automatic; the implementation of Vanneschi et al. [12] uses pointers to previous results; our implementation[2] uses memoisation [7]. Large trees are often also associated with a decrease in generalisation ability (i.e. overfitting), but this is shown by Vanneschi et al. [12] to be bounded above.

### 3.2 Novel GSGP Variations

It is interesting to consider variations on the GSGP mutation operator, with the goal of avoiding the large trees it produces, but retaining the beneficial geometric property.

We propose a *one-tree* GSGP mutation operator which instead uses only a single new random tree and draws $s$ from a normal distribution centred at 0:

$$t_{\text{new}} = t + st_1$$

For each dimension, the tree $st_1$ has expected value 0, because of the distribution of $s$. This operator adds $3 + |t_1|$ nodes per step, so it approximately halves the number of nodes in the eventual result.

We also propose an *optimal-step one-tree* GSGP mutation operator, defined by the same equation as the one-tree operator, which again uses only one new random tree, and again adds $3 + |t_1|$ nodes per step. The difference is that instead of drawing $s$ from a normal distribution, it finds the *optimal* value for $s$ at each mutation event. The optimal value of $s$ is the positive or negative constant which minimises the distance of the resulting tree from the optimum. This value can be calculated by differentiation. In the following, $y$ is the target vector in semantic space, i.e. the vector of target values at the fitness cases; $t$ and $t_1$ are to be interpreted as the vectors of the corresponding tree's outputs at the fitness cases. Multiplication and other operators are to be interpreted element-wise.

The distance of the resulting tree from the optimum, which we wish to minimise, is $\text{RMSE}(y, t + st_1)$. Minimising RMSE is equivalent to minimising MSE.

$$\begin{aligned}\text{MSE}(y, t + st_1) &= \text{mean}((y - (t + st_1))^2) \\ &= \text{mean}(((y - t) - st_1)^2) \\ &= \text{mean}((y - t)^2 - 2(y - t)st_1 + s^2 t_1^2)\end{aligned}$$

[2] All code and data used in this study is available for download from https://github.com/jmmcd/PODI.

To find the optimal $s$, we differentiate with respect to $s$:

$$\begin{aligned} d(\text{MSE})/ds &= \text{mean } (-2(y-t)t_1 + 2st_1^2) \\ &= -2 \text{ mean}((y-t)t_1) + 2s \text{ mean}(t_1^2) \end{aligned}$$

This is zero when:

$$2 \text{ mean}((y-t)t_1) = 2s \text{ mean}(t_1^2)$$

Therefore the optimum value for $s$ is:

$$s = \text{mean}((y-t)t_1)/\text{mean}(t_1^2)$$

All the values $t$ and $t_1$ are known; in the symbolic regression setting, the values $y$ are also known. Therefore, the optimal value of $s$ can be calculated. This results in using the new random tree $t_1$ to always step in the direction of the target vector in semantic space (not guaranteed using the other mutation operators); and to take the step of precisely the right length, to minimise the new distance to the target vector. The process is visualised in Fig. 1.

A GP method using a standard GP mutation operator, or the GSGP or GSGP-one-tree operators, is *black-box*: it requires only the ability to call the fitness function. In contrast, GSGP-optimal-ms requires knowledge of the values $y$, and so is not a fully black-box method.

### 3.3 Standard GP

As a control we also used a GP operator which is not geometric in the semantic space: the subtree mutation of standard GP. In our implementation, subtree mutation cuts at any node (even the root), and replaces with a new subtree created using the *grow* method. Again, we use a mutation-only hill-climber. Due to the ability to cut even at the root, the subtree mutation operator can, by itself, reach any point in the search space; and it induces a landscape with no local optima. GP hill-climbing is known to perform surprisingly well [10]. It provides a direct comparison with the GSGP hill-climbers.

## 4 Experiments and Results

### 4.1 Trading Strategy

We cast trading on time series as a symbolic regression problem. GP attempts to predict the time series as a function of the lagged variables. More precisely, we use the log-returns time series $L_t = \log(v_t/v_{t-1})$. The goal is to estimate a function $f(x) = L_t, x = (L_{t-19}, L_{t-18}, \ldots, L_{t-1})$.

The predictor is operationalised as a trader with a simple strategy: at each time-step either a long or short position is opened, depending on the sign of the predicted log-return. It is closed at the next time-step and returns are collected. The returns consist of the simple return $r_t = (v_t - v_{t-1})/v_{t-1}$ if the open position

was long, or the negative of the simple return if the open position was short. This is summarised by saying the returns are $\text{sgn}(\widehat{L_t})r_t$. The returns are accumulated over time. Such a model is useful only for testing, since it ignores real-world issues such as trading costs and interest.

We also define the classification accuracy $\text{CA} = \#(\text{sgn}(\widehat{L_t}) = \text{sgn}(L_t))/N$, i.e. the proportion of time-steps on which the predicted sign is correct.

### 4.2 Experimental Setup

Three price histories were used: Gold (GOLD), GBP/USD (GU), and the Standard & Poor 500 index (SP500), each taken at 5-minute intervals for 1400 time-steps. (1-hour data was also considered, but discarded after pilot experiments on the theory that the structure in the time series being exploited by GP was rather short-term.) The data is available for download: see https://github.com/jmmcd/PODI. The log-return at time-step $t$ was calculated as $\log(v_t/v_{t-1})$. The data was split into training and test data (418 test points), omitting the first 19 points for use as lags.

Two baselines were used. For each dataset we calculate an ARIMA model using the R function `auto.arima`, available in the `forecast` package. It automatically chooses the model order to minimise the AIC (Akaike information criterion). For our datasets, it chose ARIMA models as follows: GOLD (3, 0, 3), GU (4, 0, 1), and SP500 (2, 0, 2). The first integer indicates the auto-regression order, the second the degree of differencing, and the third the moving average order. In all cases the degree of differencing is zero, as expected because the log-return time series is stationary. Having chosen these models, it then fits the model using the training sets. Accumulated returns are calculated over the testing set. For each dataset we also calculate the returns accumulated using a buy-and-hold strategy over the testing set.

The GP alphabet consists of one variable for each of 19 lags, the constants -1, -0.1, 0.1, and 1, and the functions +, -, *, /, sin, sqrt, and square. A fitness evaluation budget of 20,000 was used, with 40 generations of 500 individuals each. At each generation a single best individual was selected as the parent of the next generation. For GP subtree mutation, a maximum depth of 12 was used. For GSGP, the mutation step was $s = 0.001$, as used by Moraglio et al. [9] and found to perform well by Vanneschi et al. [12].

Previous work [9,12] has not reported the algorithm or parameters used to generate the trees $t_1$ and $t_2$, but it is likely that non-trivial trees are being generated. We use the *grow* algorithm. Pilot experiments found that using a maximum depth of 3 offered no advantage over a maximum depth of 2, so 2 is used in all experiments to be reported (a tree of a single node is counted as depth 0, so maximum depth 2 allows a tree of up to 7 nodes).

The hypotheses to be tested are:

- Can any GP/GSGP methods out-perform the buy-and-hold and ARIMA baselines in trading on test data?
- Which of the GP/GSGP methods performs the best?

### 4.3 Results

Table 1 shows the main results. For each dataset, the ARIMA and buy-and-hold performance are shown first. For each type of mutation (GP, GSGP, GSGP-one-tree, GSGP-optimal-ms), the best run out of 30 (chosen by classification accuracy on the training set) is then considered. Its classification accuracy on training and test sets is shown. Finally, a 0 or 1 indicates whether its accumulated returns after 50, 100, and then all 418 time-steps of the test data have out-performed *both* ARIMA and buy-and-hold.

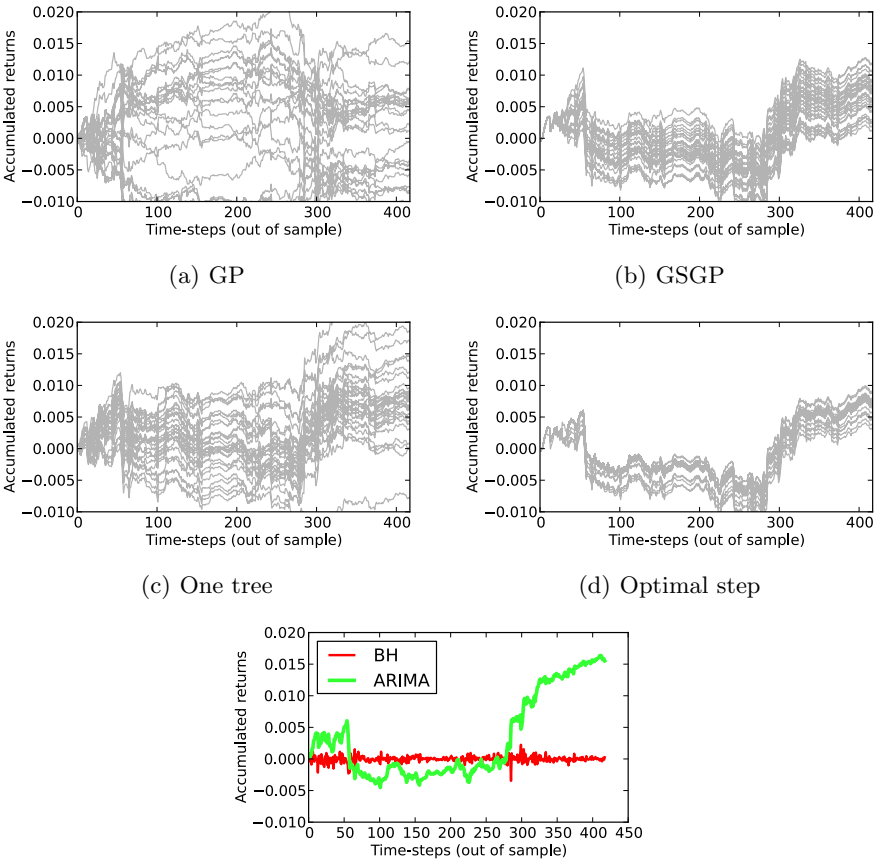
**Table 1.** Results. ARIMA and buy-and-hold performance are shown for each market. For GP, GSGP, and variants, the best result out of 30 runs, as measured by classification accuracy on the training set, is shown. Its classification accuracy on the training set and test set are shown (CA train and CA test), followed by a 0 or 1 indicating whether its returns were better than both ARIMA and buy-and-hold after 50, 100, or all 418 time-steps of the test data.

| Market | Method | CA (train) | CA (test) | R@50 | R@100 | R@End |
|---|---|---|---|---|---|---|
| GOLD5m | Buy and hold | n/a | n/a | -0.00070 | -0.00071 | 0.00026 |
| | ARIMA | 0.54 | 0.54 | 0.00432 | -0.00464 | 0.01367 |
| | GP | 0.61 | 0.59 | 0 | 0 | 0 |
| | GSGP | 0.58 | 0.57 | 0 | 0 | 0 |
| | GSGP-one-tree | 0.57 | 0.58 | 1 | 0 | 0 |
| | GSGP-optimal-ms | 0.58 | 0.58 | 1 | 0 | 0 |
| GU5m | Buy and hold | n/a | n/a | -0.00040 | -0.00005 | 0.00016 |
| | ARIMA | 0.50 | 0.50 | -0.00177 | -0.00200 | 0.00095 |
| | GP | 0.56 | 0.59 | 0 | 0 | 1 |
| | GSGP | 0.55 | 0.59 | 1 | 1 | 1 |
| | GSGP-one-tree | 0.57 | 0.54 | 1 | 0 | 1 |
| | GSGP-optimal-ms | 0.58 | 0.55 | 1 | 1 | 1 |
| SP5005m | Buy and hold | n/a | n/a | 0.00000 | 0.00044 | -0.00076 |
| | ARIMA | 0.64 | 0.65 | -0.00051 | -0.00145 | -0.00089 |
| | GP | 0.78 | 0.80 | 1 | 0 | 1 |
| | GSGP | 0.65 | 0.65 | 0 | 1 | 0 |
| | GSGP-one-tree | 0.67 | 0.62 | 1 | 0 | 0 |
| | GSGP-optimal-ms | 0.65 | 0.64 | 0 | 0 | 0 |

The results show that GP and the GSGP variants can perform well. Classification accuracy is 54-65%, with an exceptional 80%, on the test data: enough to accumulate positive returns in trading and out-perform the classification accuracy achieved by ARIMA. Note that each of GP and the GSGP variants are represented by a single individual here, hence no statistical test is carried out.

However the returns accumulated by the ARIMA method can be quite good, in particular on the GOLD dataset. Its performance near the end of the test data is unbeatable using GP or GSGP variants. The trading performance on the GOLD dataset is shown in Fig. 3. However, in other cases both ARIMA and buy-and-hold can be beaten (indicated by a 1 in the final three columns).

Accumulated returns using the GP/GSGP methods are particularly strong, and more reliable, in the short term – up to about 50 time-steps. This suggests that a good strategy is to retrain the model frequently with up-to-date data. This tends to confirm the previously-stated theory that the 1-hour data is less amenable to GP/GSGP learning.




(a) GP (b) GSGP

(c) One tree (d) Optimal step

(e) ARIMA/BH

**Fig. 3.** Returns on the Gold 5-minute data with ARIMA and buy-and-hold shown for comparison. Many of the GP/GSGP variants do well early on, but ARIMA's performance near the end of the trading period is very good.

Fig. 3 shows the accumulated returns on the Gold data only. As shown, ARIMA does well in the first 50 time-steps, then quite badly before achieving large gains near the end. The buy-and-hold strategy does not do well with these datasets, because there is no consistent upward trend. Neither is there a consistent downward trend, so a "sell-and-hold" strategy would not perform well

either. Buy-and-hold and "sell-and-hold" are equivalent to predicting constant True and constant False, respectively.

Next, the different GP mutation types were compared. Two out-of-sample criteria were used: the classification accuracy on the test set (higher is better) and the returns after 50 time-steps of the test set (higher is better). Mann-Whitney $U$ tests were used, to avoid requiring an assumption of normality. The significance threshold was $\alpha = 0.05$. For each dataset, 6 pairwise tests were performed (GP v GSGP, GP v GSGP-one-tree, GP v GSGP-optimal-ms, etc.) A Bonferroni correction was applied, in other words $p$-values were multiplied by 6 to compensate for the multiple tests. Results are shown in Table 2.

**Table 2.** Comparison of GP mutation types. The ordering of the median value is shown as $<$ if the difference is non-significant and as $<<<$ if significant. GSGP-one-tree is notated as "1t" and GSGP-optimal-ms as "Opt".

| Data | Criterion | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| GOLD | CA | Opt | $<$ | 1t | $<$ | GSGP | $<<<$ | GP |
| GOLD | R@50 | GP | $<<<$ | GSGP | $<$ | Opt | $<$ | 1t |
| GU | CA | GP | $<$ | 1t | $<<<$ | Opt | $<<<$ | GSGP |
| GU | R@50 | 1t | $<$ | Opt | $<<<$ | GSGP | $<$ | GP |
| SP500 | CA | Opt | $<$ | GSGP | $<$ | 1t | $<<<$ | GP |
| SP500 | R@50 | Opt | $<<<$ | GSGP | $<$ | GP | $<$ | 1t |

The two criteria (classification accuracy and Returns@50) often disagreed in the ordering of the values. In fact, results are very mixed: all four mutation types "won" at least once, counting cases where two "winners" tied with non-significant differences. However, in summary it seems that GP has performed quite well, certainly holding its own overall against the GSGP variants; whereas the GSGP-optimal-ms method has not demonstrated any great advantage, at least using these two out-of-sample criteria.

One possible interpretation of the results is that the time series contain a limited amount of structure to be exploited by learning methods, and that both ARIMA and standard GP are sufficient to capture most of this structure. Hence the extra modelling ability of GSGP, seen in previous work, is unneeded.

The improvement in fitness over the generations (not shown for lack of space) is relatively slight for all GP/GSGP methods. Again, this suggests that whatever structure is present is being exploited easily in the early generations, and that longer evolution is not needed.

The computational time required for the different methods was not recorded. However, all GP methods are roughly comparable in this, and are far slower than deterministic methods such as ARIMA.

## 5 Discussion

One interesting effect of the GSGP variants was observed in pilot experiments. GSGP is "greedy": once added, a subtree cannot be deleted from a GSGP

individual. If a subtree displays a division by zero or other "pathological" behaviour known to occur in GP [4], it may be difficult for evolution to counteract, and performance on the entire run may be affected. Usually, such subtrees are simply not selected, so the problem does not arise. However, GSGP and variants are vulnerable to a poor choice of initial individual. If it is chosen randomly, there may be a substantial probability of choosing a subtree of pathological behaviour. Instead, it is safer to select the initial individual from an initial population. Experiments to measure the size of this effect are ongoing.

Previous work has not exhibited any solution trees produced using GSGP methods. The trees produced by [12] were too large to reconstruct on computer, never mind in print. The individuals evolved by [9] used the functions +, -, and * only, so implicit simplification into polynomials was possible, so the trees' "true form" (i.e. the form produced prior to simplification) could be avoided. That simplification is not possible with our alphabet. So, it is interesting to look at an example tree produced using the GSGP-one-tree variant after just three steps (random constants have been rounded off):

```
(+ (+ (+ (* (/ x3 0.1) (sin x0)) (* 1.346 (* (/ x0 x5) (sin -0.1))))
(* -0.0506 (+ (+ x6 x1) x1))) (* 0.165 (sin (sin x3)))).
```

It is a linear combination of random subtrees, with both positive and negative coefficients. GSGP-optimal-ms produces trees of similar form. The original GSGP method also produces a linear combination of random subtrees, though using both addition and subtraction, and with all coefficents equal to 0.001. In fact, it is useful to see GSGP and variants as ad-hoc approaches to generalised linear models (GLMs). This relationship has not been explored in previous work. It suggests that previous research into using GP in a GLM context is of relevance, in particular the Fast Function Extraction system of [6].

The form of trees produced by GSGP and variants may be limiting. They cannot use non-linear behaviour at the root. Although non-linearities arise in the random subtrees, these are crucially never subject to gradual improvement, only re-weighting.

## 6 Conclusions and Future Work

Although somewhat mixed, our results are perhaps the first positive results using GSGP on real-world data. We have shown that GP, GSGP, and variants can perform well out-of-sample over short time horizons. The novel GSGP variants produce smaller trees, relative to the original GSGP. It is clear that for each setup, some runs (out of 30) are far more successful than others. Therefore, future work will consist of using a validation dataset to pick out the individuals created during the most successful runs, and then trade on the test set only using those. In the meantime, using the best classification accuracy on the training set to choose the best runs seems to work well.

## References


1. Agapitos, A., O'Neill, M., Brabazon, A.: Stateful program representations for evolving technical trading rules. In: Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation GECCO, pp. 199–200. ACM (2011)
2. Brabazon, A., O'Neill, M.: Biologically inspired algorithms for financial modelling. Springer, Berlin (2006)
3. Brabazon, A., O'Neill, M.: Natural computing in computational finance, vol. 1-3. Springer (2008)
4. Keijzer, M.: Improving symbolic regression with interval arithmetic and linear scaling. In: Ryan, C. et al. (eds.): EuroGP 2003. LNCS, vol. 2610, pp. 70–82. Springer Heidelberg (2003)
5. Lohpetch, D., Corne, D.: Outperforming Buy-and-Hold with Evolved Technical Trading Rules: Daily, Weekly and Monthly Trading. In: Di Chio, C., Brabazon, A., Di Caro, G.A., Ebner, M., Farooq, M., Fink, A., Grahl, J., Greenfield, G., Machado, P., O'Neill, M., Tarantino, E., Urquhart, N. (eds.) EvoApplications 2010, Part II. LNCS, vol. 6025, pp. 171–181. Springer, Heidelberg (2010)
6. McConaghy, T.: FFX: Fast, scalable, deterministic symbolic regression technology. In: Genetic Programming Theory and Practice IX, pp. 235–260. Springer (2011)
7. Michie, D.: Memo functions and machine learning. Nature **218**(5136), 19–22 (1968)
8. Moraglio, A.: Towards a geometric unification of evolutionary algorithms. Ph.D. thesis, University of Essex (November 2007). http://eden.dei.uc.pt/~moraglio/
9. Moraglio, A., Krawiec, K., Johnson, C.G.: Geometric Semantic Genetic Programming. In: Coello, C.A.C., Cutello, V., Deb, K., Forrest, S., Nicosia, G., Pavone, M. (eds.) PPSN 2012, Part I. LNCS, vol. 7491, pp. 21–31. Springer, Heidelberg (2012)
10. O'Reilly, U.M., Oppacher, F.: Program search with a hierarchical variable length representation: Genetic programming, simulated annealing and hill climbing. In: Davidor, Y., Schwefel, H.P., Manner, R. (eds.) Parallel Problem Solving from Nature - PPSN III. LNCS, vol. 866, pp. 397–406. Springer, Jerusalem (1994). http://www.springer.de/cgi-bin/search_book.pl?isbn=3-540-58484-6
11. Tsay, R.S.: Analysis of financial time series, 3rd edn. Wiley, Hoboken (2010)
12. Vanneschi, L., Castelli, M., Manzoni, L., Silva, S.: A New Implementation of Geometric Semantic GP and Its Application to Problems in Pharmacokinetics. In: Krawiec, K., Moraglio, A., Hu, T., Etaner-Uyar, A.Ş., Hu, B. (eds.) EuroGP 2013. LNCS, vol. 7831, pp. 205–216. Springer, Heidelberg (2013)