On the Genetic Programming of Time-Series Predictors for Supply Chain Management

Alexandros Agapitos, Matthew Dyson, Jenya Kovalchuk, and Simon M. Lucas
University of Essex
Department of Computing and Electronic Systems
Colchester, CO4 3SQ, United Kingdom
{aagapi,mdyson,yvkova,sml}@essex.ac.uk

ABSTRACT

Single and multi-step time-series predictors were evolved for forecasting minimum bidding prices in a simulated supply chain management scenario. Evolved programs were allowed to use primitives that facilitate the statistical analysis of historical data. An investigation of the relationships between the use of such primitives and the induction of both accurate and predictive solutions was made, with the statistics calculated based on three input data transformation methods: integral, differential, and rational. Results are presented showing which features work best for both single-step and multi-step predictions.

Categories and Subject Descriptors

I.2 [**ARTIFICIAL INTELLIGENCE**]: Automatic Programming

General Terms

Algorithms, Performance, Economics, Management

Keywords

Prediction/Forecasting, Statistical Time-Series Features, Single-Step Prediction, Iterated Single-Step Prediction

1. INTRODUCTION

In today's highly dynamic, time-constrained environment, developing effective and efficient decision support systems is a key challenge. In the domain of supply chain management (SCM) concerns include the planning and coordination of the activities of organizations from getting raw materials, manufacturing goods to delivering them to customers, supporting dynamic strategies is a major but unresolved issue. The ability to learn and adapt to new conditions in the environment is of paramount importance as the complex nature of the domain renders the application of analytical algorithms for decision-making problematic.

GECCO'08, July 12–16, 2008, Atlanta, Georgia, USA.

There is obviously a great need to simulate such dynamic environments and provide test-beds in order to evaluate the applicability and efficiency of various machine learning approaches to the problem. The TAC SCM (Trading Agent Competition - Supply Chain Management) game was introduced by Carnegie Mellon University and the Swedish Institute of Computer Science in 2003. Ever since, it became an annual event in which a number of teams from around the world compete against each other in a simulated supply chain domain. The game is now probably the best vehicle for testing SCM agents as it encapsulates many of the tradeoffs that could be found in real SCM environments (time-constraints, network latency, unpredictable opponents, etc.). All constituent parts of a supply chain are highly connected and interdependent. Being competent in one of them does not guarantee improvement of the overall performance. Many researchers follow a divide-and-conquer strategy by decomposing the problem and tackling each subtask using the most appropriate method with the expectation that the final integration will be able to reach an optimal or near-optimal performance.

The TAC SCM scenario is very briefly defined as follows (a detailed description can be found in http://www.sics.se/tac/): Six agents compete in the game, acting as PC manufacturers. They buy components from suppliers, assemble them in PCs and sell them to customers. The behaviour of both suppliers and customers are simulated by the TAC server. The aim of each participating manufacturer is to maximize their profit; the agent with the highest bank balance at the end of the game wins. The game lasts for 220 simulated days. On every day of the game each agent is faced with the following challenging tasks: (i) component procurement, (ii) product sales, (iii) production scheduling, and (iv) delivery scheduling. During procurement, an agent sends request-for-quotes (RFQs) for components and decides on which supplier offers to answer with order. On the sales side, an agent gets customers RFQs for PCs, decides on which RFQs to respond with an offer along with the offer price to set. To deal with customer orders, an agent makes decisions on which PCs to produce and which PCs to deliver. Potential expenses are attributed to component purchase, storage cost for keeping an inventory of components and PCs, penalties for late deliveries of customers' orders and bank overdrafts. Income sources consist of revenue from PCs sales and interest on positive bank balance. One of the most important phases of the SCM life cycle deals with submitting bidding prices during auctioning of customer offers. Each customer sends RFQs

for PC manufacturing and participating agents offer their bids. The lowest price proposed wins the auction and the respective agent is allocated the manufacturing task.

This paper deals with the problem of predicting winning bidding prices for customer offers. For this purpose, we have formulated the problem of price prediction into the problem of time-series modeling and forecasting based on historic data of auction winning prices in order to pursue profits increase. We treat the SCM environment as a dynamic black-box system that generates time-series in terms of winning bidding-price reports that reflect the variable macroeconomic factors influencing the market at a particular point in time. Genetic Programming (GP) [6] has been successfully applied to the induction of programs that perform time-series trend modeling tasks, and has, in many cases, shown to outperform other traditional techniques in this problem domain. Representative, recent studies include [6, 16, 14, 8, 5, 13, 2, 9, 15, 10, 4, 3, 17, 12, 11, 7]. An aspect of fundamental engineering significance is to structure the representation space in such a way so as to increase the evolvability of solutions in that space. In the GP paradigm, the careful design of language constructs can greatly increase the range and behavioural diversity of possible programs within a programming space, rendering their discovery possible as opposed to a programming space that does not get to include them in the first place. The majority of previous GP systems have been exploring a search space populated by programs representing non-linear combinations of elementary mathematical functions and time-series terminal data. However, notable exceptions tackled the time-series modeling task via the use of technical-analysis primitives [3] or high-order statistical features [14]. In this paper we attempt a further step towards better understanding the issues involved in the area of time-series analysis and modeling via statistical language constructs in GP. Furthermore, we are investigating their effectiveness in short-term (prediction of only the next value) and long-term (prediction of 2 or more steps) predictions. A significant issue in the development of GP models for financial data processing is how to prepare the raw input observables in order to facilitate their subsequent learning. Previous work [5] has shown that different program representations are sensible to different data preprocessing techniques. Intuitively, there is a strong coupling between the data processing toolbox available to an evolved program and the nature of data presented to it during fitness evaluation. We similarly raise this issue in the present study, and investigate the impact of data transformation techniques for extracting significant information from the observables to the performance of programs that are equipped with statistical processing capabilities. Finally, a literature review of the previous approaches to the task of bidding price prediction in the SCM TAC community has not revealed any work that exploits the use of GP. The contribution towards that direction is to study a different family of predictor representation and uncover its merits and deficiencies in that particular problem domain. There thus exist ample beneficiary potential from the current study.

2. TIME-SERIES MODEL LEARNING

In time-series prediction the task is to learn a model that consists of the best possible approximation of the stochastic system that could have generated an observed time-series. Given *delayed vectors* v, the aim is to induce a model f that maps the vector v to the value x_{t+1}. That is,

$$x_{t+1} = f(v) = f(x_{t-(m-1)\tau}, x_{t-(m-2)\tau}, \ldots, x_t) \quad (1)$$

where m is embedding dimension and τ is delay time. The embedding specifies on which historical data in the series the current time value depends. This empirical study uses delay vectors with parameters $m = 20$ and $\tau = 1$. Additionally, we are interested in testing the scalability of our approach using both short-term and long term predictions. *Single-Step Prediction* (SSP) is used to predict one value x_{t+1} of the time series when all inputs $x_{t-m}, \ldots, x_{t-2}, x_{t-1}, x_t$ are given. *Iterated Single-Step Prediction* (ISSP) is employed to forecast further than one step in the future. Each predicted output is fed back as input for the next prediction while all other inputs are shifted back one place. As a result, the input consists partially of predicted values as opposed to observables from the original time series. That is,

$$\begin{aligned} x'_{t+1} &= f(x_{t-m}, \ldots, x_{t-1}, x_t); m < t \\ x'_{t+2} &= f(x_{t-m+1}, \ldots, x_t, x'_{t+1}); m < t \\ &\vdots \\ x'_{t+k} &= f(x_{t-m+k-1}, \ldots, x'_{t+k-2}, x'_{t+k-1}); m < t, k \geq 1 \end{aligned} \quad (2)$$

where k is the prediction step. By applying ISSP to the evolved predictor we can predict time series values further than one step into the future. Here, k is set to 10. Long-term predictions involve a substantially more challenging task than short-term ones. The fact that each newly predicted value is partially dependent on previously generated predictions creates a reflexive relationship among program outputs, often resulting in inaccuracy propagation and an associated rapid fitness decrease with each additional fitness-case evaluation. It has been shown [10, 9] that long-term predictors are sensitive in their initial output values and that inaccuracies of initial predictions are quickly magnified with each subsequent fitness evaluation iteration.

3. METHODS

3.1 Data transformation

Data transformation deals with techniques for extracting significant information from the observables, rendering them more amenable to training a learner. Three techniques are considered in this paper and compared in terms of their influence in learning efficient time-series predictors. Prior to each transformation, normalisation is performed to mitigate effects due to the magnitudes of time-series values. We simply scaled each value by dividing it with the maximum value in the time-series, thus converting the input values in the $0 < x \leq 1$ range. Statistical stationarity has been shown to be of great importance in the domain of time-series modeling. A stationary time series is one whose statistical properties such as mean, variance, autocorrelation, etc. are constant over time. Most statistical forecasting methods are based on the assumption that the time series can be rendered approximately stationary through the use of mathematical transformations. A stationarised series is relatively easy to predict: you simply predict that its statistical properties will be the same in the future as they

have been in the past. The predictions for the stationarised series can then be "untransformed" by reversing whatever mathematical transformations were previously used, to obtain predictions for the original series. The transformations and their respective reversals are presented below:

Differential Transformation. Differencing is a simple operation that involves calculating successive changes in the values of time-series ($y_t \leftarrow x_t - x_{t-1}$). It results in a stationary series. We consider a transformation by which each value is substituted by its variance from the mean of its neighboring data within a predefined interval l [5].

$$x_d = x_t - \frac{1}{l}\sum_{i=t-l}^{t-1} x_i \tag{3}$$

where x_d is the difference obtained from its initial value x_t minus the average of the time-series fragment that includes l previous neighbors. The advantage of such differencing is that it makes the series smoother than using ($y_t \leftarrow x_t - x_{t-1}$), thus relaxing the requirement of closely fitting the trend. The reverse transformation for obtaining predictions for the original series is as follows:

$$Rev_d = P + \frac{1}{l}\sum_{i=1}^{l} x_i \tag{4}$$

where P is the prediction value using equation 3, l is the number of neighbours and x_i is the i^{th} element of the delayed vector.

Rational Transformation. The effect of this transformation is similar to this of equation 3 but results in series with weaker oscillations by smoothing large magnitudes in the differences of consecutive values [5].

$$x_r = ln\left(\frac{x_t}{x_{t-1}}\right) \tag{5}$$

where x_t and x_{t-1} are consecutive data values in the series. The reverse transformation is as follows:

$$Rev_r = e^{[P+ln(x_1)]} \tag{6}$$

where P is the prediction value using equation 5 and x_1 is the first element of the delayed vector.

Integral Transformation. This technique eliminates noise from the series (but does not stationarise the series) replacing each value by its moving average computed with a smoothing period of l neighbours according to [5]:

$$x_i = \frac{1}{l}\sum_{k=t-(l-1)/2}^{t+(l-1)/2} x_k \tag{7}$$

where l is the number of neighbours that are considered in the moving average filtering and x_k is the k^{th} series value. The reverse transformation takes the following form:

$$Rev_i = \frac{1}{l - \lceil \frac{l}{2} \rceil}\left(P \cdot l \cdot \sum_{k}^{l-\lceil \frac{l}{2} \rceil} x_k\right) \tag{8}$$

where P is the prediction value using equation 7, l is the number of neighbours, and x_k is the k^{th} element of the delayed vector.

3.2 Program Representation Language

Evolvable individuals employ an expression-tree representation. The primitive language is depicted in Table 1. Time-series data is presented to the program in two forms. Firstly, as scalar values representing the previous 20 data points. Secondly as a list (indexed from zero) of 20 values where the first data element (in index 0) represents the series value at time t, the second element represents the series value at time $t-1$ and so on up to time $t-19$. The implementation of language constructs for time-series statistical analysis require information about the data points these statistics should be calculated on. Statistical primitives accept 3 arguments, the first being the time-series and the remaining two define the left *start* (left) and *end* (right) bounds of the fragment of the time-series to consider in the statistical function. *Start* specifies how many steps back from the prediction (at time $t+1$) the statistics should be calculated from, while *end* specifies the number of data points to include in the statistics, given by: $datapoints = end - start + 1$. If the start and stop positions specify a negative fragment then the opposite interpretation is taken, also, if they are out of time-series length bounds, their values are induced by taking the modulus to the time-series length. The statistical function `diff` is the difference between the average values of two halves of a time-series fragment:

$$Diff(L,s,e) = \left(\frac{1}{\frac{e-s}{2}}\sum_{k=s}^{\frac{e-s}{2}+s} L_k\right) - \left(\frac{1}{\frac{e-s}{2}}\sum_{k=\frac{e-s}{2}+s}^{e} L_k\right) \tag{9}$$

where L is the input list representing the time-series and s, e are the start and end indices respectively. First order moment `pdm1` and second order moment `pdm2` are *position dependent* statistics that measure how the high-valued series points are distributed away from the center of a time-series fragment. The first order moment is the average of the time-series values weighted by their absolute distance from the middle point of the fragment and the second order moment is the variance of these values:

$$pdm1(L,s,e) = \frac{1}{e-s}\sum_{k=s}^{e}\left[L_k \cdot \left|k - \left(\frac{e-s}{2} + s\right)\right|\right] \tag{10}$$

$$pdm2(L,s,e) = \frac{1}{e-s}\sum_{k=s}^{e}\left\{\left[L_k \cdot \left|k - \left(\frac{e-s}{2} + s\right)\right|\right] - pdm1\right\}^2 \tag{11}$$

where L_k is the k^{th} input list element and s, e are the start and end indices respectively.

3.3 Fitness functions

The fitness function used during training of single-step predictors takes the form of *Mean Squared Error* (MSE) defined as:

$$MSE = \frac{1}{N}\sum_{k=1}^{N}(x_k - x'_k)^2 \tag{12}$$

Table 1: Language for evolving Predictors

Method	Argument(s)	Return
+, −, ∗, /	double, double	double
exp, ln, sqrt, sin, cos	double	double
mean, std.dev., skewness kurtosis, min, max, diff pdm1, pdm2	List, double, double	double
Terminal	**Value**	**Type**
Constant	25 rnd. in [0,100] 20 previous values	double
Parameter	1 time-series	List

Table 2: Summary of prediction performance

	Training		Testing		
	MSE	RSE	MSE	HIT	APC
	Single-Step Prediction				
Rational	.0035	.4088	.0045	.9274	.1896
Differential	.0009	.7650	.0051	.9200	.1889
Integral	.0004	.9928	.0081	.9128	.1871
	Iterated Single-Step Prediction				
Rational	.0375	.4231	.0401	.4686	.0136
Differential	.0110	.7606	.0223	.5063	.0156
Integral	.0314	.9385	.0469	.5021	.0155

where x_k and x'_k are the actual and predicted values respectively, and N is the number of training cases. For iterated single-step predictions we used an augmented version of mean squared error which rewards programs that make more accurate predictions in the far future. The rationale for this selection pressure is that forecasts in the near future are easier than in the long term. *Augmented Mean Squared Error* (AMSE) takes the following form:

$$AMSE = \frac{1}{N}\sum_{i=1}^{N}(1 + P \cdot A) \cdot (x_k - x'_k)^2 \qquad (13)$$

where x_k and x'_k are the actual and predicted values respectively, N is the number of training cases, P is the prediction step and A is an augmentor coefficient which is set to 0.05.

3.4 Evolutionary Algorithm, Variation operators, and Run Parameters

For evolutionary algorithm we used a panmictic, generational genetic algorithm combined with elitism (0.5%). The algorithm uses tournament selection with a tournament size of 4. The evolutionary run proceeds for 50 generations and the population size is set to 500 individuals. Evolution halts when all of 50 generations have elapsed. Ramped-half-and-half tree creation with a maximum depth of 7 is used to perform a random sampling of program space during the initial generation. During the run, expression trees are allowed to grow up to depth of 17. Our search employs a mixture of mutation-based variation operators [1]: (i) **All Nodes Mutation**, (ii) **Macro Mutation**, (iii) **Point Mutation**, (iv) **Swap Mutation**, (v) **Grow Mutation**, (vi) **Truncation Mutation**, (vii) **Gaussian Mutation**. These variation operators are applied in the following way: A sample S from a Poisson random variable with a mean of 2 was generated. S random mutation operators were uniformly picked (with replacement) from the set of available operators and were applied in sequence using a *pipe-and-filter* pattern (i.e. `Mutant=(Swap(Grow(Parent)))`). Let this type of mutation be called *Variation-Bundle*. In order to account for the exploration-exploitation trade-off we allow for the selection of either a *Variation-Bundle* or a single point-mutation (each node is being mutated with a probability of 15%) using an adaptive probability that is induced as follows: $Prob_{single-mut} = k * (gen_{current}/gen_{max})$, where $gen_{current}$ and gen_{max} are the current and maximum number of generations in the evolutionary run respectively, and k is a discount coefficient which is set to 0.55.

3.5 Training/Test data sets

Training sets have been generated within the TAC SCM simulation. Information available during the game includes the lowest and highest order prices for each PC type from the previous day, and also the request volume, order volume, and average order price during the reporting period. Price reports are available to all competing agents on a daily basis. Information on bidding prices of competitors is not available to TAC agents. Using the price reports we built the time series of the lowest prices and these were used during the training phase. Thus, the training set consisted of 65 price series for each PC type, collected in 65 independent SCM simulation runs. Testing involved utilising evolved predictors in the TAC SCM simulation live (against the same set of competitor agents). Results are based on 20 games. Datasets have been derived from log files generated while playing against agent distributions published by participating teams and can be available upon request.

3.6 Prediction performance measures

We evaluated the evolved predictors in terms of MSE (depicted in equation 12), *Ratio of Squared Errors* (RSE) [5], *Hit Percentage* (HIT) [5] and *Average Percentage Change* [14] (APC).

Like MSE, RSE is used to evaluate the level of fitting, but in addition, it is able to show an improvement over a random walk model [5]. It is defined as:

$$RSE = \frac{\sum_{i=1}^{N}(x_i - x'_i)^2}{\sum_{i=2}^{N}(x_i - x_{i-1})^2} \qquad (14)$$

where x and x' are the actual and predicted values respectively, and N is the number of training cases.

HIT provides an indication of the accuracy that the evolved model has followed the trend of the observables. It is defined as [5]:

$$HIT = \frac{1}{N-1}\sum_{i=2}^{N}\begin{cases} 1 & \text{if } \Delta x_i > 0 \wedge \Delta x'_i > 0 \\ & \vee\ \Delta x_i < 0 \wedge \Delta x'_i < 0 \\ 0 & \text{if } \Delta x_i > 0 \wedge \Delta x'_i < 0 \\ & \vee\ \Delta x_i < 0 \wedge \Delta x'_i > 0 \end{cases} \qquad (15)$$

where N is the number of points in the time-series and $\Delta x_i = x_i - x_{i-1}$ is defined as the difference of two consecutive points. Finally, APC is a performance measure that takes into consideration the magnitude of the incorrect pre-

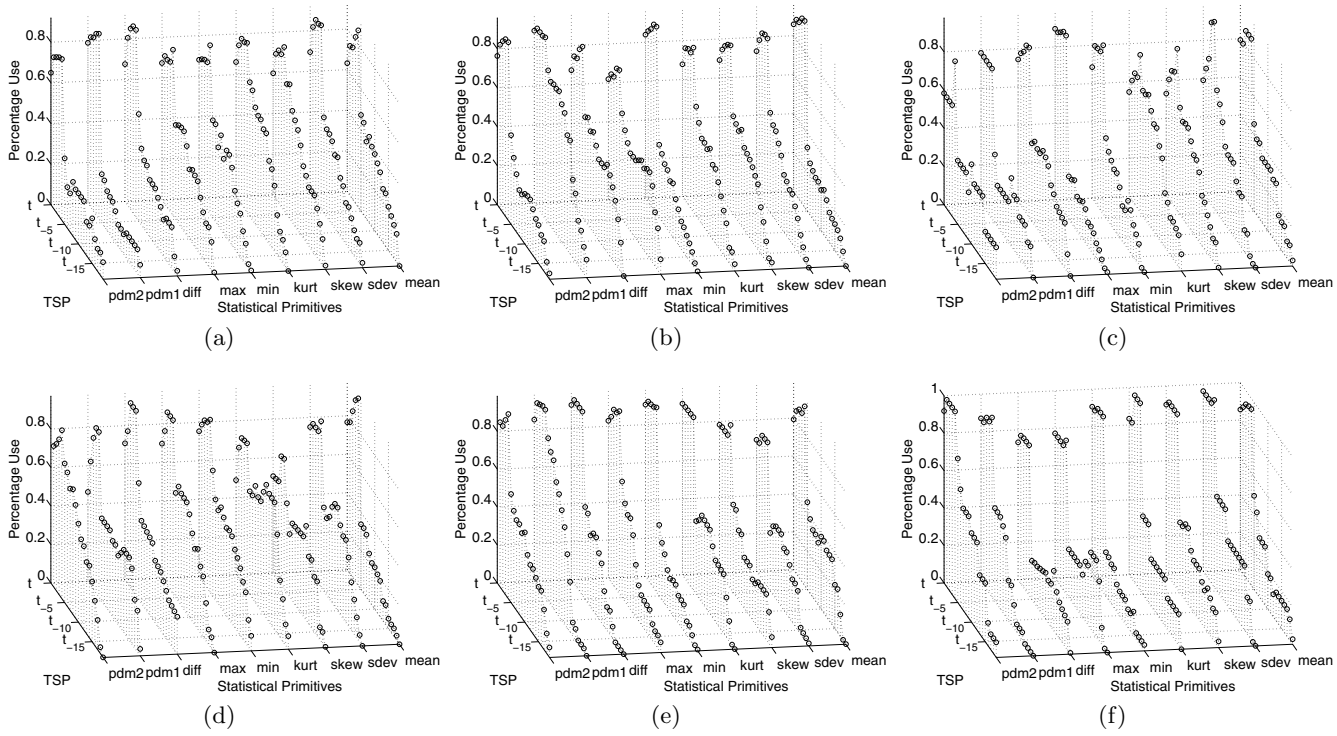

Figure 1: Average percentage of coverage of time-series points (TSP) by different statistical primitives. (a) Differential-Single; (b) Rational-Single; (c) Integral-Single; (d) Differential-Iterated; (e) Rational-Iterated; (f) Integral-Iterated.

Table 3: Average depth of best-of-run predictors (std. deviation in parentheses)

	Single-step	Iterated Single-step
Rational	9.49 (4.13)	8.69 (4.32)
Integral	6.55 (4.49)	5.12 (4.67)
Differential	7.64 (5.11)	6.05 (4.52)

diction. It is defined as:

$$APC = \frac{1}{N-1}\sum_{i=2}^{N}\begin{cases} +\left(\frac{|x_i - x_{i-1}|}{x_{i-1}}\right) & \text{if} \quad \Delta x_i > 0 \wedge \Delta x'_i > 0 \\ & \vee \quad \Delta x_i < 0 \wedge \Delta x'_i < 0 \\ -\left(\frac{|x_i - x_{i-1}|}{x_{i-1}}\right) & \text{if} \quad \Delta x_i > 0 \wedge \Delta x'_i < 0 \\ & \vee \quad \Delta x_i < 0 \wedge \Delta x'_i > 0 \\ 0 & \text{if} \quad x_{i-1} = 0 \end{cases} \tag{16}$$

where N and Δx_i are defined as in equation 15. The bigger the value of this metric, the better the predictor's performance.

3.7 Context of genotypic analysis

Evolved predictors are subjected to genotypic analysis in order to infer relationships between the properties of their constituent expression-tree nodes and prediction performance. These quantitative properties concern:

(i) *Average use of statistical primitives* (U) within the expression-tree structure. This is simply the count of distinct statistical tree-nodes in the whole population divided by the number of individuals. We calculate one average per statistical primitive included in the primitive language.

(ii) *Percentage of time-series coverage* (C). During program execution, the invocation of a statistical function requires that the start and end bounds of the list (representing the time-series) should be specified. We monitor every such function invocation during program evaluation and calculate both the percentage of individual point usage, and the coverage (in terms of the number of points used) of the time-series as a whole. Averages are maintained for the coverage for each statistical primitive separately.

(iii) *Skewness of distribution of temporal position of series values* (S). The `List` representing the time-series stores the value of the series x_t in index 0, while the value at time x_{t-i} in index i. Every time a point is being accessed we record its index in the `List`. The skewness of the distribution of the values of these list indices gives an indication of the historic information that the prediction is based on. A positive skewness indicates that more series values are being accessed from the beginning of the `List` while a negative one indicates that the prediction is based more on values of the distant past. The skewness is calculated for every evolved predictor based on each different statistical primitive (note that in order to calculate the skewness we include the extremities in the sample, which are the values 0 and 19 – the `List` bounds). Subsequently, averages are induced. Let us illustrate this with an example. Suppose a statistical primitive is applied to a time-series covering values with indices 0,1,2,3,4. The skewness of list {0,0,1,2,3,4,19} is 2.38. On

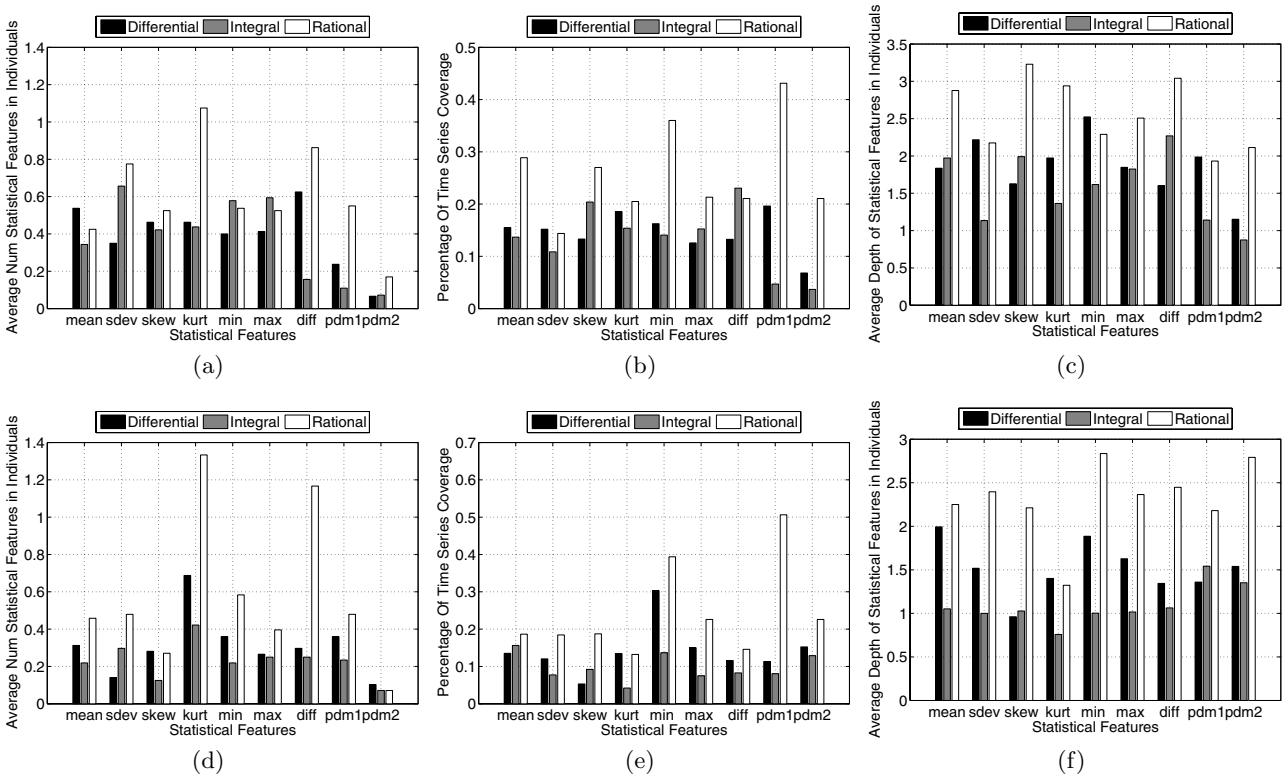

Figure 2: Average usage of statistical features. (a) Single; (d) Iterated. Average percentage of coverage of time series points. (b) Single; (e) Iterated. Average depth of statistical features within the expression-tree structures. (c) Single; (f) Iterated.

the other hand if the primitive is applied to values residing in indices 10,11,12,13, the skewness of list {0,10,11,12,13,19} is −0.91. This indicates how the skewness of the distribution of series points covered can reveal temporal information of historical data used for prediction.

(iv) *Average depth of statistical primitives* (D) within the expression-trees. The hierarchical nature of a tree data structure imposes a concomitant effect on the level of impact of various tree-nodes in the overall program fitness. It is generally true that, in the absence of side-effecting primitives, tree-nodes near the root contribute to a greater degree than tree-nodes in lower tree-levels. When combined with the average primitive usage, this genotypic property captures information about the effectiveness of specific statistical primitives within the current evaluation context, which in turn provides an indication of their overall appropriateness to the prediction task. Average statistical primitive depths are recorded both in individual and population levels.

4. RESULTS

We performed 80 independent experimental runs for each single-step and iterated single-step prediction (with a maximum step of 10) tasks under every data transformation technique. As already mentioned (section 3.5), learning was based on 65 time-series of minimum bidding prices. For testing, we used the best evolved predictors of each type and ran 20 additional simulations in order to collect performance statistics. The genotypic analysis (discussed in section 3.7) is based on 80 evolved programs for each prediction type.

Table 2 illustrates training and testing performance results. An initial, general observation is that both single- and multi-step predictors achieved competent results under all three data transformation techniques. During the training phase of simple-step predictors, an initial observation concerns potential overfitting under differential and integral transformations as this is evidenced by the very low mean square error. Despite this observation, their evolved solutions show a similar, good generalisation capability as this is measured by HIT and APC. Finally, we observe that single-step predictor training with the rational series yielded an overall better performance in the test data. In the case of multi-step prediction the differential series appear to achieve better generalisation results despite the low training error possibly indicative of overfitting. As before, the magnitude of differences is small, rendering all three data transformation techniques amenable for use in learning multiple-step predictors.

Figure 1 shows the average percentage of time-series points used by statistical primitives as measured during genotypic analysis. The first observation is that in both prediction types, under all transformations, statistical operations are being applied the most to series values closer to the present. It appears that all cases make heavy use of the last 5 days. We also calculated the skewness of the distribution of the used points' temporal characteristic (as defined

Table 4: Pearson correlation coefficients between genotypic features and prediction performance

SINGLE-STEP PREDICTORS (*sample size in parentheses*)									
Mean Squared Error (MSE)									
Rational	S `diff` -.461 (30)	C `pdm1` -.241 (57)	U `pdm1` -.158 (57)	C `sdev` .167 (38)	D `sdev` .253 (38)	S `sdev` -.333 (38)	S `skew` -.392 (47)	S `min` -.303 (29)	S `mean` -.202 (29)
Integral	U `min` -.465 (28)	S `sdev` -.456 (40)	D `min` -.380 (28)	S `pdm1` -.372 (23)	C `kurt` -.316 (28)	C `sdev` .213 (40)	S `kurt` .306 (23)	S `pdm2` .395 (23)	C `pdm1` .730 (23)
Different.	S `diff` -562 (25)	S `pdm2` -551 (21)	S `min` -511 (54)	U `diff` -240 (25)	U `min` -230 (54)	U `pdm2` -228 (21)	D `min` -213 (54)	C `sdev` .630 (22)	C `kurt` .481 (21)
Hit Percentage (HIT)									
Rational	C `sdev` -.260 (47)	D `sdev` -.196 (47)	S `diff` -.172 (29)	S `sdev` .138 (47)	S `pmd1` .120 (30)	U `kurt` .121 (29)	S `mean` .153 (46)	U `mean` .244 (29)	C `kurt` .290 (57)
Integral	D `pdm1` -.180 (17)	D `sdev` -.104 (17)	C `min` .359 (40)	S `min` .366 (40)	U `min` .441 (40)	C `kurt` .520 (18)	S `sdev` .581 (17)	S `pdm1` .677 (17)	U `pdm1` .880 (17)
Different.	S `kurt` -.26 (20)	D `sdev` -.325 (20)	D `kurt` -.144 (20)	S `sdev` -.112 (20)	S `kurt` .197 (20)	S `pdm2` .243 (19)	C `pdm2` .313 (19)	S `diff` .389 (25)	S `min` .471 (21)
ITERATED SINGLE-STEP PREDICTORS (*sample size in parentheses*)									
Mean Squared Error (MSE)									
Rational	S `pdm1` -.269 (39)	S `diff` -.231 (38)	C `sdev` -.230 (18)	S `max` -.226 (18)	U `pdm1` -.208 (39)	U `sdev` -.181 (18)	U `kurt` .241 (38)	D `kurt` .434 (38)	S `sdev` .455 (18)
Integral	U `kurt` -.567 (13)	C `kurt` -.489 (13)	U `pdm2` -.472 (16)	U `skew` -.466 (13)	D `kurt` -.205 (13)	D `skew` .328 (13)	S `pdm2` .330 (16)	S `min` .337 (27)	S `skew` .408 (13)
Different.	U `kurt` -.528 (31)	D `kurt` -.407 (31)	S `kurt` -.377 (31)	U `sdev` -.353 (17)	C `diff` -.278 (20)	C `sdev` -.228 (17)	D `diff` .202 (20)	D `sdev` .258 (17)	S `diff` .263 (20)
Hit Percentage (HIT)									
Rational	S `sdev` -.318 (38)	D `diff` -.254 (18)	D `sdev` -.246 (38)	S `diff` -.173 (18)	S `kurt` -.159 (19)	U `kurt` .302 (19)	C `sdev` .334 (38)	S `pdm1` .341 (19)	S `max` .462 (18)
Integral	D `skew` -.353 (16)	D `pdm2` -.167 (16)	D `min` -.149 (13)	C `pdm2` -.142 (16)	S `kurt` .148 (27)	S `min` .163 (13)	U `min` -.388 (13)	S `skew` -.404 (16)	S `kurt` .487 (27)
Different.	D `sdev` -.312 (31)	S `diff` -.182 (31)	C `kurt` -.157 (14)	D `diff` -.154 (31)	S `kurt` -.105 (14)	S `sdev` .165 (31)	U `kurt` .174 (14)	S `diff` .487 (31)	C `diff` -.480 (31)

by the `List` index in which they reside), we found that for the multi-step predictors the distribution is more positively skewed (avg. of 0.69) as opposed to the single-step predictors (avg. of 0.63). This indicates that the multi-step the predictors are using more series values towards the present (`List` index 0). This can be particularly observed in Figure 1(f), leaving us with the impression that predictors rely heavily on a very short 5-day-time-window.

A key issue addressed in this paper is the appropriateness and impact of including statistical primitives when tackling the problem of evolving programs for forecasting. Table 4 presents the most interesting relationships that were revealed between best-of-run predictors' genotypic characteristics (discussed in section 3.7) and MSE/HIT metrics used during testing. We calculated Pearson correlation coefficients to identify linear relationships using sample sizes that are indicated in the parentheses next to respective coefficients. An important item to note is that when using correlation we do not infer causation rather an *"is seen with"* relationship.

The case of single-step predictors under the rational data transformation showed a negative correlation between C (percentage of time-series coverage, -0.167), S (skewness of distribution of temporal position of series values, -0.333) of `std.dev.` statistical primitive and MSE. This suggests that low MSE (good predictor) is seen with high skewness (prediction is based on first indices of the input `List`) and high coverage of the input `List` size. This is also illustrated in Figure 2(a) which shows that `std.dev` is frequently used in the population and that its frequent appearance is justified by its impact on the overall prediction performance. The fact that a good predictor bases its future guesses on a time-window which is placed in the first indices of series values is also supported by the positive correlation between S and HIT (0.138), suggesting that high HIT is seen with high S. This is also illustrated in Figure 2(b) which shows that under the rational transformation `std.dev` is being applied in a small portion of the input list (15% coverage). Another interesting correlation of negative sign is observed between D (average depth of statistical primitives) of `std.dev.` and HIT (-0.196), which suggests that high coverage is seen with low depth of `std.dev.` tree node. This is also apparent in the positive correlation between D and MSE (0.253). As one might expect, this claim needs to be made from within a context identifying the average depth of `std.dev.` tree-node within the evolved expression-trees. Figure 2(c) illustrates that the average depth of `std.dev` nodes is 2.2 which further suggests that such nodes reside in the upper tree levels and they are more likely to have a significant impact in performance.

Single-step predictors under the integral data transformation showed, among others, an overall relationship with the use of `min` and `std.dev.` language primitives. The average use U of `min` is negatively correlated with MSE (-0.465) and at the same time, positively correlated with HIT (0.441). So, low MSE and high HIT is seen with high use of `min`.

In the case of single-step predictors under the the differential transformation we distinguish the impact of the use of `kurtosis` which shows a quite strong positive correlation

(0.481) C and MSE as well as a positive correlation S and HIT (0.197) suggesting its application to a small interval of series points near the first indices of the input `List`. The negative correlation between D and HIT (-0.144), and the low average depth depicted in Figure 2(c) supports the argument of its impact in performance.

Following a similar train of thought, the genotypic analysis of multi-step predictors distinguished:
(i)`std.dev.` applied under the rational transformation with high time series coverage (correlation between C and HIT is 0.334) towards the final indices of the `List` (correlation between S and MSE is 0.455). The correlation between D and HIT is 0.246. Also, `diff` plays an important role with a negative correlation between D and HIT (-0.254).
(ii)`min` applied under the integral transformation shows a negative correlation between D and HIT (-0.149), and a negative correlation between U and HIT (-0.388) which suggests that high average usage of this primitive yields a high hit percentage. The positive correlation between S and HIT suggests that this primitive accesses series elements residing in small indices. In addition, `skew` appears to be promising with a positive correlation between D and MSE (0.328) that is supported by the negative correlation between D and HIT (-0.353) as well as the average depth of this statistical primitive in the population (0.94). The positive correlation between S and MSE (0.408) which is supported by the negative correlation between S and HIT (-0.404) suggests that the application of such a statistical function considers series values in high (towards the end of the `List`) indices.
(iii)`std.dev.` appears valuable when applied under the differential transformation showing a positive correlation between D and MSE (0.258) as well as a negative correlation between D and HIT (-0.312). The positive correlation between S and HIT (0.165) suggests that high hit percentage is seen with the application of this statistical operation in small indices of the input `List`.

5. CONCLUSIONS

This paper presented an empirical study on the automatic induction of both single- and multi-step time-series predictors by means of genetic programming. Constructs for performing statistical operations on the input series where included in the primitive language and their effectiveness has been demonstrated by the evolution of both accurate (low testing MSE) and predictive (high HIT/APC) models. Three data transformation techniques were assessed in terms of their impact on the learnability of the task, all yielding quite similar results, distinguishing rational and differential transformations for the evolution of single- and multi-step prediction models respectively. Genotypic analysis of evolved predictors addressed various aspects of the use of such statistical primitives and revealed relationships with observed performance.

6. REFERENCES

[1] K. Chellapilla. Evolving computer programs without subtree crossover. *IEEE Transactions on Evolutionary Computation*, 1997.

[2] S.-H. Chen. *Genetic Algorithms and Genetic Programming in Computational Finance*. Kluwer Academic Publishers, 2002.

[3] A. L. Garcia-Almanza and E. P. K. Tsang. Forecasting stock prices using genetic programming and chance discovery. In *12th International Conference On Computing In Economics And Finance*, 2006.

[4] A. Hui. Using genetic programming to perform time-series forecasting of stock prices. In *Genetic Algorithms and Genetic Programming at Stanford*. 2003.

[5] H. Iba and N. Nikolaev. Genetic programming polynomial models of financial data series. In *Proceedings of the IEEE 2000 Congress on Evolutionary Computation*, 2000.

[6] J. Koza. *Genetic Programming: on the programming of computers by means of natural selection*. MIT Press, Cambridge, MA, (1992).

[7] G. Y. Lee. Genetic recursive regression for modeling and forecasting real-world chaotic time series. In *Advances in Genetic Programming 3*. 1999.

[8] N. Y. Nikolaev and H. Iba. Genetic programming of polynomial models for financial forecasting. In *Genetic Algorithms and Genetic Programming in Computational Finance*. 2002.

[9] H. Oakley. Two scientific applications of genetic programming: Stack filters and non-linear equation fitting to chaotic data. In *Advances in Genetic Programming*. 1994.

[10] W. Panyaworayan and G. Wuetschner. Time series prediction using a recursive algorithm of a combination of genetic programming and constant optimization. *Electronics and Energetics*, 2002.

[11] D. Rivero, J. R. Rabunal, J. Dorado, and A. Pazos. Time series forecast with anticipation using genetic programming. In *Computational Intelligence and Bioinspired Systems, 8th International Work-Conference on Artificial Neural Networks*, 2005.

[12] K. Rodriguez-Vazquez and P. J. Fleming. Genetic programming for dynamic chaotic systems modelling. In *Proceedings of the IEEE CEC*, 1999.

[13] M. Santini and A. Tettamanzi. Genetic programming for financial time series prediction. In *Genetic Programming, Proceedings of EuroGP'2001*, 2001.

[14] R. Schwaerzel and T. Bylander. Predicting currency exchange rates by genetic programming with trigonometric functions and high-order statistics. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, 2006.

[15] I. Yoshihara, T. Aoyama, and M. Yasunaga. GP-based modeling method for time series prediction with parameter optimization and node alternation. In *Proceedings of the 2000 IEEE Congress on Evolutionary Computation*, 2000.

[16] T. Yu and S.-H. Chen. Using genetic programming with lambda abstraction to find technical trading rules. In *Computing in Economics and Finance*, 2004.

[17] W. Zhang, Z. ming Wu, and G. ke Yang. Genetic programming-based chaotic time series modeling. *Journal of Zhejiang University Science*, 2004.