Stateful Program Representations for Evolving Technical Trading Rules

Alexandros Agapitos, Michael O'Neill, and Anthony Brabazon
Financial Mathematics and Computation Research Cluster
Natural Computing Research and Applications Group
University College Dublin, Ireland
{alexandros.agapitos, m.oneill, anthony.brabazon}@ucd.ie

ABSTRACT

A family of stateful program representations in grammar-based Genetic Programming are being compared against their stateless counterpart in the problem of binary classification of sequences of daily prices of a financial asset. Empirical results suggest that stateful classifiers learn as fast as stateless ones but generalise better to unseen data, rendering this form of program representation strongly appealing to the automatic programming of technical trading rules.

Categories and Subject Descriptors

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

General Terms

Algorithms, Performance, Experimentation

Keywords

Stateful Program Representations, Memory in Genetic Programming, Evolution of Technical Trading Rules, Classification of Financial Time-series

1. INTRODUCTION

This paper presents an empirical study of sequence classification via stateful Genetic Programming (GP). The binary classification task considers the issuing of *go-long* or *go-short* commands in a financial trading system. Technical analysis is employed to extract features [3] from the raw time-series of daily prices of a financial asset. These indicators are used as building blocks during the evolutionary synthesis, by means of grammar-based GP, of binary classification decision-trees for the financial time-series. Time-series processing is being performed using a single-step moving-window approach, and a classification is obtained for each time-step. The program space, where GP operates, contains programs that are allowed to maintain state information [4, 2, 5, 1] in-between the sequential program execution, given input that is obtained through a number of differing realisations of the traditional moving-window approach.

GECCO'11, July 12–16, 2011, Dublin, Ireland.
ACM 978-1-4503-0690-4/11/07.

2. INDEXED MEMORY UTILISATION

Stateful-A is the general case of a state-aware sequence processing program. Under this formulation of memory usage, scalar strongly-typed state variables are initialised to default values (zero in the case of real-valued variables) prior to program execution. In each time-step, features are extracted from the raw time-series and these are fed as input to the program that returns a binary classification as output. During each execution, a program is allowed to access and modify a number of state variables. The process essentially utilises a moving time-window of size one that is being rolled over until the entire time-series has been exhausted.

Stateful-B is a more constrained approach to sequence processing. A moving time-window (outer window) is being utilised to produce each classification. This outer window of recent time-steps is being itself traversed by a moving time-window of size one (inner window). Prior to a classification, state variables are being initialised. Hence, the inner window is used to iteratively feed the features to each successive program execution, which has the ability to inspect and modify state information. A classification is being produced at the time-step that signifies the end of the outer window, the state variables are being initialised, and the outer window slides one time-step to the right. This process continues until the whole time-series have been exhausted.

Stateful-C is a memory utilisation method that attempts to explicitly bias the impact of more recent information on classification output, while still allowing the program to capture long-term time dependencies. The method has been designed to achieve this by performing an independent single-step traversal over the time-steps of a recent time-window in order to issue a classification output, while allowing for the state information to be maintained in-between subsequent classification outputs. The process resembles that of Stateful-B, with the difference that the state variables are only initialised in the beginning of the sequence processing, and are maintained throughout subsequent program executions until the whole time-series is exhausted.

Finally, **Stateless** is the case of a stateless sequence processing program. Here, a single time-step moving window is utilised to traverse the time-series and extract features that are used to output a classification decision. No state variables are defined.

3. EVOLVING TRADING RULES

The GP system evolves technical trading rules in the form of decision-trees. Each expression-tree is collection of `if-then-else` rules that are represented as a disjunction of

Table 1: Experimental results. Bold face indicates best performance. Δ denotes the mean percentage change in ADR between *Best* and *Final*, and quantifies the magnitude of overfitting.

Window	Train set Best	Train set Gen.	Test set Best	Test set Gen.	Test set Final	Δ
Memory utilisation: Stateless						
n/a	0.31	48.32	0.07	10.08	-0.02	-148%
Memory utilisation: Stateful-A						
n/a	0.30	47.58	0.07	11.72	-0.02	-136%
Memory utilisation: Stateful-B						
20	0.27	47.94	0.09	15.08	0.02	-80%
40	0.26	47.72	0.08	10.28	0.01	-262%
60	0.28	47.73	**0.09**	15.63	0.03	-69%
80	0.28	47.42	0.09	12.60	0.02	-83%
100	0.29	48.19	0.07	9.73	-0.01	-157%
Memory utilisation: Stateful-C						
20	0.27	47.76	0.09	11.58	0.03	-73%
40	0.27	48.89	0.08	16.49	0.02	-89%
60	0.27	48.36	0.09	11.59	0.02	-84%
80	0.28	47.97	**0.10**	12.14	0.03	-75%
100	0.29	47.85	0.07	11.33	-0.05	-212%

conjunctions of constraints on the values of technical indicators [3]. The technical indicators that are used in this experiment are: (a) **simple moving average** (MA), (b) **trade break out** (TBO), (c) **filter** (FIL), (d) **volatility** (VOL), and **momentum** (MOM). These are parameterised with lag periods, which is the number of past time-steps that each operator is based on. Currently, we allow periods from 5 to 200 trading days, with a step of 5 days. We also include the closing price of the asset at each trading day.

In order to allow for a program space that enables the representation of stateful programs, standard `read` and `write` operators have been defined. `read` returns the value stored in memory index `address`. `write` sets the value at memory index `address`, and returns the value of this memory location that has just been overwritten. In addition to those standard memory-manipulation operators, soft-assignment operators have been defined. These are `updateAdd` and `updateMul` that are semantically equal to `i=i+value`, and `i=i*value` respectively. Once the memory update has been performed, the operators return the value at `address` that has just been overwritten. The indexed memory is set to size 10. In the case of stateless program representation, no indexed memory is being utilised, and the grammar is being modified to exclude the memory-manipulating primitives.

The GP algorithm employs a panmictic, generational, elitist genetic algorithm. The selection scheme, run initialisation, variation operators, and search size are similar to the experimental setup in [3]. The maximisation problem uses a fitness function that takes the form of the **Information Ratio** of daily returns. This is defined as the ratio of average daily return (ADR) generated by the rule's trading signals over a training period, divided by the standard deviation of these daily returns. The dataset used is a time-series of daily prices of the Nikkei 225 index for the period of 01/01/1990 to 31/03/2010. The first $2,500$ trading days are used for training, whereas the remaining $2,729$ compose the out-of-sample test set.

4. RESULTS AND CONCLUSION

Stateful representations performed better than their stateless counterpart. A statistically significant difference (unpaired t-test, $p < 0.0001$, $df = 98$) was found between stateless (mean ADR of 0.07, mean annualised return of 14%) and Stateful-C-80 (mean ADR of 0.10, mean annualised return of 20%). Another statistically significant difference (unpaired t-test, $p < 0.0003$, $df = 98$) was found between stateless and Stateful-B-60 (mean ADR of 0.09, mean annualised return of 18%). Furthermore, no statistically significant differences were found between the number of generations required to evolve the best-generalising programs in both stateful and stateless cases. Results suggest that the best-generalising individuals are discovered relatively early in the evolutionary process, on average before generation 20. After that point, overiffting becomes apparent in both stateful and stateless program representations. The level of overfitting, accruing from continued training till the end of the evolutionary run, becomes more pronounced in the case of stateless programs. Results show that the average percentage decrease in ADR for the case of stateless programs reaches 148%, resulting to a final mean ADR of -0.02. For the same dataset, stateful representations Stateful-B-60 and Stateful-C-80 are attaining a performance decrease of only 69% and 75% respectively, resulting in final mean ADR of 0.03 in both cases. In conclusion, stateful program representations in this problem domain produced more profitable trading rules, and suffered less from the problem of overfitting, compared to the stateless program representation. No differences were found relative to the speed of evolutionary learning between stateful and stateless families of program representation.

Acknowledgements

This publication has emanated from research conducted with the financial support of Science Foundation Ireland under Grant Number 08/SRC/FM1389.

5. REFERENCES

[1] A. Agapitos, M. Dyson, S. M. Lucas, and F. Sepulveda. Learning to recognise mental activities: genetic programming of stateful classifiers for brain-computer interfacing. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, 2008.

[2] A. Agapitos and S. M. Lucas. Evolving a statistics class using object oriented evolutionary programming. In *Proc. of 10th European Conference on Genetic Programming*, volume 4445, pages 291–300, 2007.

[3] A. Agapitos, M. O'Neill, and A. Brabazon. Evolutionary learning of technical trading rules without data-mining bias. In R. Schaefer, C. Cotta, J. Kolodziej, and G. Rudolph, editors, *PPSN 2010 11th International Conference on Parallel Problem Solving From Nature*, volume 6238 of *Lecture Notes in Computer Science*, pages 294–303, Krakow, Poland, 11-15 Sept. 2010. Springer.

[4] A. Agapitos, J. Togelius, and S. M. Lucas. Evolving controllers for simulated car racing using object oriented genetic programming. In *Proceedings of the Genetic and Evolutionay Computation Conference*, 2007.

[5] A. Agapitos, J. Togelius, and S. M. Lucas. Multiobjective techniques for the use of state in genetic programming applied to simulated car racing. In *Proc. of IEEE CEC*, pages 1562–1569, 2007.