Evolving Trading Rule-Based Policies

Robert Bradley[1,2], Anthony Brabazon[1,2] and Michael O'Neill[1,3]

[1] Natural Computing Research and Applications Group
University College Dublin, Ireland.
[2] School of Business, University College Dublin, Ireland.
[3] School of Computer Science and Informatics, University College Dublin, Ireland.
robert.bradley@ucdconnect.ie anthony.brabazon@ucd.ie m.oneill@ucd.ie

Abstract. Trading-rule representation is an important factor to consider when designing a quantitative trading system. This study implements a trading strategy as a rule-based policy. The result is an intuitive human-readable format which allows for seamless integration of domain knowledge. The components of a policy are specified and represented as a set of rewrite rules in a context-free grammar. These rewrite rules define how the components can be legally assembled. Thus, strategies derived from the grammar are well-formed, domain-specific, solutions. A grammar-based Evolutionary Algorithm, Grammatical Evolution (GE), is then employed to automatically evolve intra-day trading strategies for the U.S. Stock Market. The GE methodology managed to discover profitable rules with realistic transaction costs included. The paper concludes with a number of suggestions for future work.

1 Introduction

A rule-based trading system typically relies on a series of conditional rules to make trading decisions. This is in contrast to a discretionary system which primarily relies on human judgement. Discretionary traders function without explicitly quantified rules and instead act on mental rules which are developed through experience. An obvious problem with this approach is that the decision-making process can be skewed by human emotions such as fear and greed [7]. Although there are successful traders who function exclusively using these techniques, there are many advantages to quantifying one's rules. Quantifying a strategy and representing the logic in computer code allows for system automation, speeding up the decision-making process, and for the statistical analysis of the trading strategy via backtesting. In addition, it facilitates the application of advanced machine learning techniques in order to optimise system parameters. In this study we employ an evolutionary algorithm called Grammatical Evolution (GE) [8] to automatically evolve profitable trading models. GE has previously been applied to for-ex trading [1] and to stock market trading [2] . These studies evolve expressions which evaluate to real-numbers, and rules are formulated by comparing these values to threshold values in order to generate a trading entry signal. The exit strategies employed are static which is a simplification of

the process of real-world trading. In contrast, this study structures a grammar such that a trading strategy is comprised of entry and exit rules which sensibly combine technical indictors resulting in a more intuitive and comprehensive representation of a trading strategy.

1.1 Structure of Paper

The remainder of this paper is structured as follows. Section 2 outlines the construction of a rule-based policy as applied to the problem of trading-rule design. Section 3 illustrates how a set of rewrite rules can govern the steps in creating a well-formed, domain-specific policy. In section 4, Grammatical Evolution is employed to heuristically navigate the search space of possible trading rules in order to find ones which are high-quality. The results of our experiments are presented in section 5. The last section outlines the conclusions of the study and suggests a number of avenues of future work.

2 Rule-Based Policies

Representation is a key factor when designing a trading strategy. One intuitive approach is to represent a strategy as a *rule-based policy*. This approach has previously been used to develop automated agents for a well known arcade game [6], and more generally provides a framework which encapsulates many real-world decision scenarios. A *rule-based policy* is a set of rules with the following structure

IF [Condition] holds, THEN do [Action]

The policy also includes logic to decide which rule in the set should be executed given a particular state of the environment. Rule-based policies are a particularly useful representation for financial trading rules as they are human readable and it is a straightforward task to embed domain knowledge. In applying these policies we need to specify four things:

1. What are the possible actions?
2. What are the possible conditions and how are they constructed from observations?
3. How to make rules from conditions and actions
4. How to combine the rules into policies

We will address each of these in turn.

For the purpose of our study we are limiting the system to five basic actions as outlined in Table 1 below: *EnterLong, ExitLong, EnterShort, ExitShort and DoNothing*. On the close of each time interval (one minute intervals in the case of our experiments), a trading agent is confronted with the problem of deciding which of these actions to take.

Action	Description
EnterLong	Open a long position
ExitLong	Close long position
EnterShort	Open a short position
ExitShort	Close short position
DoNothing	Take no action

Table 1. The above table shows the list of actions which are used in the construction of trading rules.

An action is taken if the condition associated with the executed rule holds true. The market position is updated accordingly, as per Table 2. In the case where conditions belonging to multiple rules are satisfied simultaneously, the policy must include logic to prioritize one action over another.

Action	DoNothing	EnterLong	ExitLong	EnterShort	ExitShort
Position	Flat	Long	Flat	Short	Flat

Table 2. This table shows the market position state changes resulting from a sequence of actions.

A trading rule condition is a Boolean expression of observations and comparison operators. The system designer decides on the set of observations, where members of the set could range from a simple closing price to a sophisticated statistical metric. In this study we have limited our observation set to four technical indicators (see Table 3). The length of this list is arbitrary, and can be extended with any number of variables, indicators, and analytics.

Analytic	Description
SMA	simple moving average which gauges momentum
WMA	weighted moving average which gauges momentum
STOC	Stochastic indicator which gauges overbought/oversold levels
ADX	ADX indicator gauges strength of trend

Table 3. This list of technical indicators serve as the building blocks in the construction of trading rule conditions.

A trading strategy is then constructed by logically combining one or more rules of the form IF [Condition] holds, THEN do [Action]. Each of the actions listed in Table 1 have a rule which decides whether or not the action should be taken based on a boolean condition. The policy includes logic which decides which rule should be executed given the current market position [9]. For example,

if we are already long then it makes no sense to execute the *ExitShort* rule. This feedback loop from the environment to the policy produces more logical trading decisions.

3 Grammatical Representation

A context-free grammar is a set of one or more rewrite rules of the form $NT- > T$, where NT is a nonterminal which maps to one or more terminals and/or nonterminals. This study employs a metasyntax called Backus Naur Form (BNF) to express the rewrite rules. We define a root nonterminal $< policy >$, see Fig. 1, which is mapped to the policy framework discussed in Section 2.

```
<Policy>      ::=          if(Long){
                                 if(<LongExitCondition>){
                                       ExitLong;
                                 }
                           }
                           else if(Short){
                                 if(ShortExitCondition){
                                       ExitShort;
                                 }
                           }
                           else if(Flat){
                                 if(<LongEntryCondition>){
                                       EnterLong;
                                 }
                                 else if(<ShortEntryCondition>){
                                       EnterShort;
                                 }
                           }
```

Fig. 1. The root nonterminal maps to the policy framework which includes terminals and non terminals.

Rewrite rules are also added to define how conditions may be constructed from observations and operators. This allows us to incorporate our domain knowledge into the grammar, resulting in a set of rules that governs how a well-formed policy can be assembled.
We embed our domain knowledge in the grammar by defining rewrite rules which ensure that sensible conditions are created. For example, it would not be sensible to directly compare a moving average and a stochastic indicator, which oscillates between 0 and 100. Thus, we create separate nonterminals for each type of indicator. This inhibits ill-formed conditions such as $(SMA(10) > STOC(65))$ being created. Fig. 2 shows the production rules used to build conditions. A more comprehensive set of observations might require additional domain knowledge to be included in the grammar.

Digit concatenation [3] can also be applied to the creation of constants within a strategy. Uses of constants in trading rules include for example, the parameterisation of technical indicators, and the creation of appropriate thresholds

```
<condition>        ::=  <ma><greatless><ma>
                      | <stochastic> <greatless> <threshold>
                      | (<condition>\&\&<condition>)
 <maindicator>     ::=  SMA(<number>)
                      | WMA(<number>)
<oscillator>       ::= STOC(<number>)
                      | ADX(<number>)
```

Fig. 2. A partial BNF grammar showing three production rules used to create well-formed conditions.

for (as an example) oscillator indicators. We define rewrite rules to control the range of values which a constant can take. The integer parameter passed to technical indicators specifies the number of intervals over which the indicator will be calculated. Domain knowledge is embedded in our rewrite rules so that this parameter is limited to an appropriate range of values. For example, in determining the number of periods over which a technical indication can 'look back' we need to consider the likely trading frequency. In this paper, we focus on high-frequency trading and we limit the range of this parameter to 2000, which is about 5 days of trading given a 1 minute frequency. Therefore, the production rules in Fig. 3 below allow for the generation of integers in the range [1,1999].

```
<number>        ::=   <1-9>
                    | <1-9><0-9>
                    | <1-9><0-9><0-9>
<1-9>           ::=   1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<0-9>           ::=   0 | <1-9>
```

Fig. 3. The production rules which define how digits can be concatenated to create an indicator look-back parameter constant.

Similarly, a set of rewrite rules are defined to govern how digits may be concatenated to create the threshold constant. This threshold is compared to the oscillator indicators, for example ($STOC(55) > 80$). Domain knowledge is embedded in these production rules to limit this constant to be in the range [0,95] in increments of 5. A more complex grammar might define rewrite rules for a number of other constants which are used in a policy, with domain-specific knowledge applied to each constant if necessary.

4 Evolution of Trading Policies

Rule-based trading policies derived from our context-free grammar are guaranteed to be well-formed solutions. However, we are still faced with the challenge of deriving a successful trading strategy. A huge number of different strategies can be derived depending on the derivation sequence executed, and hence we

need to traverse this search space in an efficient manner. To do this we adopt an evolutionary approach.

The GE algorithm was inspired by the genotype to phenotype mapping process in biology. This process involves the mapping of DNA to proteins. In the case of GE, integer strings drive the selection of rewrite rules from our grammar which results in a mapped policy.

4.1 Data Review

The dataset, see Fig. 4, used in this study is comprised of 200 trading days of 1 minute bars for the CAT stock which is listed on the NYSE. Normal trading hours on the NYSE are 9.30 to 16:00 EST, resulting in 390 minutes of trading per session, producing a dataset of 78,000 samples. Each bar contains a price for the open, high, low, and closing trades for that minute. The dataset is partitioned to produce an in-sample section from Monday 2007-01-22 to Friday 2007-04-20, and an out-of-sample section which ranges from Monday 2007-04-23 to Friday 2007-07-20. The first partition is used to train a population of policies, and the second to test the best individuals out-of-sample.

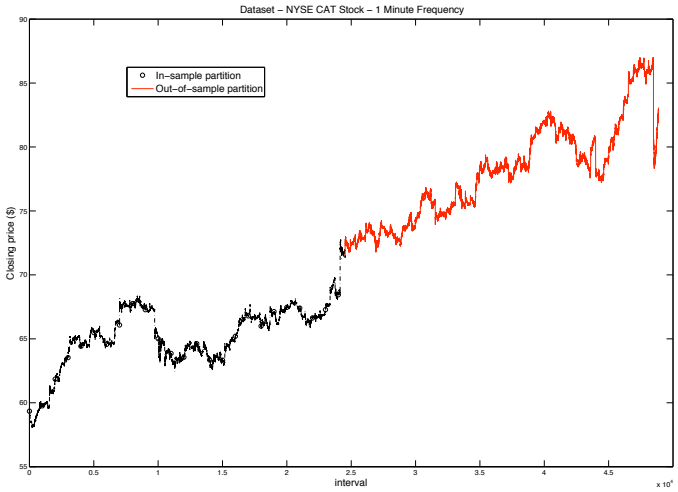

Fig. 4. Dataset of CAT high frequency data.

4.2 Methodology

The results for the data-mining of trading policies using GE were averaged over 30 separately-seeded runs in order to allow us to assess the statistical significance

of the performance metrics. A single run involves an in-sample training phase and an out-of-sample testing phase. Phase one trains a GE population of 300 individuals for 50 generations. On each generation of the algorithm, each integer string in the population is mapped to a trading policy. The in-sample dataset is then iterated from start to end. The trading policy is executed at the close of each interval and a trading action from the set listed in Table 1 is signaled. This action is then applied to the open of the next interval. For example if an *EnterLong* signal is generated on the close of interval 2007-01-22 09:56 a new trade is initiated on the open of the next interval 2007-01-22 09:57. Signals are only accepted up to and including the second last interval of the day. On the last interval of each day any open position (long or short) is closed and the market position is set to flat. This ensures no positions are held overnight. The running return of the strategy is stored at each interval.

On reaching the end of the training set the vector of returns is analyzed to determine the performance of the individual. The performance metric used in this paper is an information ratio where the average daily return minus transaction costs is divided by the standard deviation. This metric is a risk-adjusted measure which favors strategies with stable daily returns. The calculated ratio is the fitness of the strategy being evaluated. On completion of a generation, each strategy in the population has a fitness score as calculated above. Roulette wheel selection is employed with a steady-state replacement strategy, see Goldberg [5]. The population is evolved for 50 generations and the best trading policy is then tested on the out-of-sample dataset to assess the robustness of the strategy on unseen data. The experimental parameters used for this study are listed in Table 4.

Parameter	**Value**
Pop size	300
Mutation	.001
Crossover	.9
Generations	50
Runs	30
Selection	Roulette wheel
Replacement	Steady-state

Table 4. Experimental Parameters

5 Results

The results of our experiments are now presented. A population of 300 individuals was trained on partition one of the dataset discussed in Section 4.1 for 50 generations. The best individual from the population was then tested out-of-sample on partition two. Thirty separately seeded runs were carried out.

	In-sample			Out-of-sample		
	Mean	Std Dev	Info-ratio	Mean	Std Dev	Info-ratio
Best policy	57.83	32.85	1.76	13.85	30.96	0.45
Best policy - costs	51.56	29.55	1.75	9.41	30.96	0.30
Random agent	-6.07	35.32	-0.17	2.27	30.23	0.08
Random agent - costs	-354.21	35.13	-10.08	-287.05	29.37	-9.77
Buy only	0.19	NA	NA	19.94	NA	NA
Buy only - costs	-0.28	NA	NA	19.13	NA	NA

Table 5. Statistics derived from the annualized percentage return of the best policy against a number of benchmarks, averaged over 30 runs.

Table 5 shows the performance of the best individual against a number of benchmarks, averaged over 30 runs. The average best policy, not including transaction costs, returned 57.83% annualized in-sample. The standard deviation of this level over the runs was 32.85%. The mean return dropped to 13.85% out-of-sample. When transaction costs were included the in-sample mean return dropped to 51.56%, while the out-of-sample performance fell to 9.41%.

Two benchmarks were employed. The first, a zero-intelligence agent, executes an action from the set{*Buy, Sell, Do Nothing*} with equal probability on the close of each interval. Due to the uniformity of the sampling distribution this random strategy will complete a trade every 4 intervals on average. This results in a large volume of trades making transaction costs a significant factor. The average best policy fails to significantly out-perform the random agent when costs are not included, however with transactions costs of 1 cent per share included the performance of the zero-intelligence strategy is drastically reduced. We note that a zero-intelligence agent with a high trading frequency (about 100 trades a day) might not be a realistic benchmark when compared to an evolved policy trading at a potentially much lower frequency.

The second benchmark in our experiment is an intra-day buy-and-hold strategy. Each day this agent initiates a long position at the market open and goes flat at the market close. The best evolved individual significantly outperforms this benchmark in-sample. The opposite is true out-of-sample. The benchmark's superior performance over the second partition is due to the aggressively bullish trend in the second half of our dataset, see Fig. 4. Although the benchmark has superior performance the risk profile is very different to that of the evolved strategies. The buy and hold strategy is exposed to the systematic risk of the market 390 minutes a day. A typical evolved policy is in the market a lot less than this. One such policy is described below.

5.1 Example Evolved Policy

This section takes a closer look at a profitable policy evolved using Grammatical Evolution. Fig. 1 shows the root nonterminal in our BNF grammar which maps to a basic template. The policy template is comprised of a number of rules, and

the rules' conditions are represented by nonterminals in our grammar. GE is used to evolve these conditions using the rewrite rules defined in Fig. 2. An example evolved trading policy is shown in Fig. 5. The result is human-readable, and can be easily visualized in any standard technical indicator charting software.

```
if(Long){
      if(((STOCHFD_TA(3,46)>40)&&((STOCHFD_TA(5,92)>55)&&(ADX_TA(92)<40)))){
            ExitLong;
      }
else if(Short){
      if(((ADX_TA(48)>85)&&(ADX_TA(719)>85))){
            ExitShort
      }
else if(Flat){
      if((((((STOCHFD_TA(63,69)<60)&&((STOCHFD_TA(1,321)>5)&&(ADX_TA(21)<80)))&&
      (((STOCHFD_TA(5,50)>5)&&(ADX_TA(95)<65))&&(STOCHFD_TA(759,8)<30)))&&
      ((WMA_TA(74)<SMA_TA(28))&&(SMA_TA(581)>SMA_TA(1))))&&(STOCHFD_TA(2,2)<30))){
            EnterLong;
      }
      else if((ADX_TA(578)>65)) {
            EnterShort;
      }
}
```

Fig. 5. Example policy evolved by Grammatical Evolution.

The logic of the policy framework favors long positions over short as the *EnterLong* condition is checked before the *EnterShort* condition. The complete dataset used in this study is relatively bullish and the rule in Fig. 5 has essentially switched off short trades by evolving a condition for the *EnterShort* rule which will always evaluate to false as the ADX indicator is not likely to breach the 65 level. With the *EnterShort* rule out of the picture the *EnterLong* condition is checked at each interval to find a good entry point. This policy made 67 trades in-sample, and 44 out-of-sample, see Table 6. The average trade duration was approximately 60 minutes across the two partitions with a standard deviation of 65 minutes. The table also shows the mean return and the standard deviation of the return on these trades in basis points.

	In-sample		**Out-of-sample**	
	Mean	Std Dev	Mean	Std DEv
Trade duration (mins)	45	52	75	80
Return (bps)	10.32	23.47	19.82	36.78
Return with costs (bps)	8.78	23.47	18.50	36.78

Table 6. Trade duration (in minutes) and return (in basis points) statistics for the example evolved policy.

6 Conclusion and Future Work

In this study Grammatical Evolution was used to evolve well-formed trading rule-based policies for a large-cap U.S. stock. A policy is derived from a grammatical representation of the components which make up a policy. Despite the fact that we limited the system to a very simple set of building blocks, GE managed to uncover some profitable rules when realistic transaction costs were included.

In spite of the promise of these results, no absolutely definitive conclusions can be drawn from a set of experiments based on a single stock over a single time period. We intend to pursue a number of avenues to extend this work. For example, in this study the best policy from the population trained over a 3 month period is traded out-of-sample for the next 3 months. This is a conservative approach as it is ambitious to expect a simple technical trading rule to yield robust results over such a large window. A moving window approach, where the training set is incremented periodically and the population is trained for a number of generations at each increment, has been shown to yield superior returns [4] over a static approach, like the one adopted in this study. We intend to investigate this approach in more detail. We also intend to analyze the phenotypic characteristics of the system during evolution to give greater transparency into the distribution of intelligence inherent in the population.

References

1. A. Brabazon and M. O'Neill. Evolving technical trading rules for spot foreign-exchange markets using grammatical evolution. *Computational Management Science*, 1(3):311–327, October 2004.
2. A. Brabazon and M. O'Neill. Intra-day trading using grammatical evolution. In Anthony Brabazon and Michael O'Neill, editors, *Biologically Inspired Algorithms for Financial Modelling*, pages 203–210. Springer-Verlag, Berlin, 2006. chapter in biologically inspired algorithms book.
3. I. Dempsey, M. O'Neill, and A. Brabazon. Grammatical constant creation. *Genetic And Evolutionary Computation Gecco 2004 , Pt 2, Proceedings*, 3103:447–458, 2004.
4. I. Dempsey, M. O'Neill, and A. Brabazon. Adaptive trading with grammatical evolution. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pages 9137–9142, Vancouver, 6-21 July 2006. IEEE Press.
5. D. E. Goldberg. *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Kluwer Academic Publishers, 2002.
6. S. Istvan and L. András. Learning to play using low-complexity rule-based policies: Illustrations through ms. pac-man. *J. Artif. Intell. Res. (JAIR)*, 30:659–684, 2007.
7. A. W. Lo. The adaptive markets hypothesis: Market efficiency from an evolutionary perspective. *Journal of Portfolio Management, Forthcoming*, 2004.
8. M. O'Neill and C. Ryan. *Grammatical Evolution*. Kluwer, 2003.
9. P. Saks and D. Maringer. *Applications of Evolutionary Computing*, chapter Evolutionary Money Management, pages 162–171. Springer Berlin / Heidelberg, 2009.