Cartesian Genetic Programming for Trading: A Preliminary Investigation

Michael Mayo

School of Computing and Mathematical Sciences
University of Waikato, Hamilton, New Zealand
Email: mmayo@waikato.ac.nz

Abstract

In this paper, a preliminary investigation of Cartesian Genetic Programming (CGP) for algorithmic intraday trading is conducted. CGP is a recent new variant of genetic programming that differs from traditional approaches in a number of ways, including being able to evolve programs with limited size and with multiple outputs. CGP is used to evolve a predictor for intraday price movements, and trading strategies using the evolved predictors are evaluated along three dimensions (return, maximum drawdown and recovery factor) and against four different financial datasets (the Euro/US dollar exchange rate and the Dow Jones Industrial Average during periods from 2006 and 2010). We show that CGP is capable in many instances of evolving programs that, when used as trading strategies, lead to modest positive returns.

Keywords: Cartesian Genetic Programming, Algorithmic Trading, Rule Learning

1 Introduction

Algorithmic trading is the problem of automating decisions to buy and sell financial assets such that, even after trading costs and losses are taken into account, the cumulative net return from the decision series is positive. The main tasks of these decision strategies are (i) market direction prediction and (ii) position sizing, risk management, and entry/exit management. The main problem with task (i), of course, is that markets are notoriously difficult to predict. In fact, there is a long history of debate about the efficient market hypothesis (Fama, 1970) and the issue of whether or not market price movements are essentially random walks (see, for example, Beechey et al. (2000) for a recent counter-analysis). In spite of this, past research efforts from computer scientists appear to show that pattern recognition techniques such as machine learning can make profits in the markets. Recent examples include the works of Contreras et al. (2012), Lean and Lai (2007), Liu and Xiu (2009), Ni and Yin (2009), Barbosa and Belo (2008), Hirabayashi et al. (2009), and Larkin and Ryan (2010).

Putting aside the debate for a moment, task (ii) mentioned above (which is concerned with the details about whether to act on a prediction and if so, how to act) is also not without its difficulties. For example, a market may be quiet one day and volatile the next. Therefore a strategy that that assigns a large position size to a trade on the quiet day (where the risk is low) may be in violation of its own risk management rules if it assigns the same position size the following day (where the risk is higher due to an increased likelihood of sharp price movements). Markets behaviours are well known to be non-stationary series (Sewell, 2011) and therefore methods and strategies that worked in the past cannot be expected to continue working. Furthermore, non-stationarity applies not just to prices but also to other important factors such as volatility and seasonal aspects (where seasonality includes not only properties that change with an annual cycle but also those that follow intraday, time-of-day-based cycles and weekly cycles). Non-stationarity probably explains why some technical strategies that traditionally worked in the past now may no-longer yield profits.

This paper takes the view that intraday market prices may be predictable to a small degree, although that "edge" may be very slim indeed. Financial engineering may be required to actually make such predictions profitable. We also take the stance that due to non-stationarity, a large amount of past training data is *not* required for learning trading strategies – in fact, too much data may lead to problems such as the learning of patterns that are now defunct. We therefore, in our experiments, use two months of intraday data to learn a trading strategy, and test it on the following month of intraday data.

The machine learning method of choice in this paper is Cartesian Genetic Programming (CGP) (Miller, 2011). We chose this method for a number of reasons. Firstly, CGP evolves programs that can have a fixed upper limit on size because they are represented as a fixed-size array. In contrast, traditional tree-based Genetic Programming methods have no limit on size and the problems of bloat are well known. Secondly, CGP can evolve programs with multiple outputs as well as multiple inputs. Although we do not use more than one output in the experiments presented here, in the future this would be advantageous for learning trading strategies because the multiple outputs can be used to emit different aspects of the strategy. For example one output may be a prediction of direction, and the second output may be a position size indicator (with a zero indicating "no trade"). A third output could possibly be a distance to a stop loss price.

The third and final reason for CGP being interesting from a trading perspective is that as learning proceeds over time (in generations), programs tend to reduce in complexity whenever fitness hits a plateau (Miller, 2011). That is, in the absence of further im-

provements, CGP programs tend to have less active nodes due to the genetic drift feature of CGP. For a trading strategy, this is a very desirable property because smaller programs are easier for humans to understand, making them more like "traditional" indicators. Furthermore, smaller programs are less prone to overfitting.

2 Background

In this section, a brief review is given of the important concepts used in this paper. In particular, we describe the CGP approach used, and overview the important financial ideas.

2.1 Cartesian Genetic Programming

CGP is a relatively new field of genetic programming. It has found application in areas either where there is a significant amount of low-level data to be processed (e.g. in the evolution of image processing filters (Sekanina et al., 2011a)) or where the programs must adhere to significant physical constraints (e.g. the layout of circuits on a board (Sekanina et al., 2011b)). Financial applications are more related to the image processing scenario, because a trading strategy can be thought of as a "filter" on data that produces an output (that being signal to trade or not to trade), where the data is not 2D image data but is instead a stream of historical 1D price series data.

The canonical CGP algorithm (described more fully in (Miller, 2011)) is defined as follows. Firstly, a small number of fixed parameters must be specified. The first is the population size *popsize*, which in canonical CGP is set to 5. This very small population size is offset by the fact that it is customary for CGP to run for a very large number of generations, *maxgens*, which may have a value in the millions.

Further parameters describe the fixed features of each program, such as the number of inputs, n_{in}; the number of outputs, n_{out}; and the maximum number of function call instructions (or *nodes*) in a program, n_l. Typically there also needs to be a fixed *arity* parameter that specifies the number of inputs each function/node takes. We also in this research fix the number of point mutations per offspring to n_m, although in general this parameter need not be fixed.

Next, a table $Functions$ must be defined. A program in CGP is defined as a linear array of function calls of length n_l. Traditionally, basic numerical functions such as addition, subtraction, sin, cosine, square root, etc are used; alternatively, if the domain is logic circuits, low level AND and OR gates are sensible choices for functions. In our domain, we are interested in learning programs that resemble financial indicators, so the functions chosen are similar to the basic components used in those traditional indicators, such as comparison (greater than, less than, min, max), basic arithmetic operators, and the mean function. We also include functions that use none of their inputs at all, but instead have a fixed constant output such as 1 or -1. These resemble "bias" nodes from neural networks and often have an impact on the performance of CGP. The complete list of functions used in this paper are given in Table 1.

Once the parameters and $Functions$ table have been specified, the next step is to give the algorithm a value function $Fitness()$ with which to evaluate each individual program. The basic CGP algorithm can then proceed, and it does so as a simple $1 + \lambda$ evolutionary strategy (Miller, 2011) where $\lambda = 4$. In other words, the search starts from randomly generated programs, and proceeds generationally. In each generation, only the best program is retained and the others in the population are replaced by mutated offspring of the best program. There is no crossover in canonical CGP.

Table 1: Functions used to construct individual programs. Functions either take two inputs x and y, or they ignore the inputs and produce a constant value.

Function	Description
$+$	Returns $x + y$
$-$	Returns $x - y$
$\times$	Returns $x * y$
$/$	Returns x/y, or 1.0 on divide-by-zero error
$>$	Returns 1 if $x > y$, -1 otherwise
$<$	Returns 1 if $x < y$, -1 otherwise
MAX	Returns $max(x, y)$
MIN	Returns $min(x, y)$
$MEAN$	Returns $(x + y)/2$
C_1	Returns 1
C_{-1}	Returns -1
C_0	Returns 0

One interesting facet of the algorithm that differentiates CGP from other evolutionary algorithms is its method of selecting the parent for the next generation. In CGP, an offspring replaces its parent if its fitness is greater than *or equal to* its parent's fitness. That is, even if there is no improvement in fitness, the search algorithm can adopt a new "best program" and parent for the next generation as long as the offspring's fitness is at least equal to its parents. This mechanism permits neutral mutations that allow for genetic drift, a feature that adds random diversity into the population without any cost. It has been shown previously such diversity significantly improves the search performance of CGP.

An example of a program evolved using CGP in the experiments reported here is shown in Figure 1.

This example illustrates the phenomenon of non-coding regions in CGP quite well. Each node/function call in a program may be coding or non-coding. If a node is defined as coding, this means that it is connected to the inputs either directly or indirectly. In the case of indirect connection, the connection is via the outputs of another function. Furthermore, in order to be a coding node, the node's own output must *also* be used to compute the final outputs of the program. Any other function nodes are essentially useless and constitute "junk" regions of the genotype. In Figure 1, the example program has 7 coding and 23 non-coding function nodes. Note that CGP programs are directed acyclic graphs as opposed to trees or linear sequences of instructions.

2.2 Financial Concepts

The main financial trading concepts will be explained briefly in this subsection.

The first important concept to understand is the notion that assets can be *bought* and *sold* as well as *sold short*. Short selling is different from selling an asset that you already own, because rather than reducing your (positive) quantity of the asset by selling it, you actually sell your asset first (i.e. acquire a negative quantity of the asset) and gamble that the price will go down so that you can buy it back later

at a lower price, thus making a profit. Short selling is therefore the opposite of normal "long" buying and selling. In all of our experiments we assume that a trading strategy can both buy long and sell short, and that a trade (buying or selling) is closed with the opposite action.

Another important concept to understand is the way that trading strategies are evaluated. Whereas normal machine learning classifiers are evaluated via standard measures such as accuracy or ROC, in finance these concepts have very little relevance if the strategy's financial performance is also not considered. For example, a strategy with a 60% accuracy rate in picking direction will consistently lose money if its average loss per trade in dollar terms is twice its average win, even though the accuracy is greater than random.

We therefore utilise in this research the following three measures of a trading strategy's performance: *cumulative return*, i.e. the sum of the consecutive small wins and losses that a strategy makes over its testing period; *maximum drawdown*, which is defined as the maximum drop in cumulative return over the same period; and *recovery factor*, which is defined as the ratio of the first of these quantities to the second.

To illustrate, suppose that a strategy yields a profit of \$50 in the first week, but loses it all plus a further \$25 in the second week (yielding a balance of \$-25). In the third week, the strategy earns \$35 profit, thus ending the three weeks with a \$10 profit. The cumulative return in this case is \$10; the maximum drawdown is \$75; and the recovery factor is $\frac{\$10}{\$75}$ or 0.133.

Note that the recovery factor essentially normalises the return against maximum drawdown; strategies with both high returns and drawdowns should yield the same recovery factor as those with low returns but correspondingly low maximum drawdowns. A negative recovery factor indicates that the strategy made a loss, while a recovery factor of less than 1 indicates that the strategy's drawdown was greater than its eventual profit. Strategies with a recovery factors of 1 or more are therefore desirable.

3 Experimental Setup

In this section, the datasets used in the experiments are described. We then move on to outlining the way in which CGP programs were evaluated for fitness estimation purposes.

3.1 Datasets

Four datasets from two different major markets were utilised in our evaluation of CGP for trading strategy learning. The two markets selected were deliberately chosen because they are highly liquid, meaning that there is simply a larger number of traders. The "herding behaviour" of the crowd may therefore more easily become apparent in these markets. Smaller markets, on the other hand, are less liquid and therefore more prone to sudden large price movements arising from single trades and other such noise. The two markets that we chose are quite disparate in order to ensure that our approach was tested rigorously.

The chosen markets were (i) the market for US currency, as determined by the Euro/US dollar exchange rate, and (ii) the US share market, as measured by the Dow Jones Industrial Average. Both markets have quite different characteristics. We also chose two quite distinct time-periods from their market price series, namely pre-recession 2006 and post-recession 2010. The two time periods combined with the two markets yielded four datasets.

Each dataset consisted of three month's worth of data, of which the first two months were used for training and the last month for out-of-sample testing. The exact dates and details of the datasets are given in Table 2.

The data we used is available from a financial data firm, Pi Trading[1] and comes in the form of an EST time-stamped series of open, high, low and close prices for every minute that a market is open. There are no records for minute bars where there are no transactions (i.e. where the open, low, high and close values are identical), so the actual number of records in the dataset is less than the number of minutes that the markets were open for. For the exchange rate data, this amounts to about 80,000 minute records in both the 2006 and 2010 periods, and for the Dow Jones data (which is open during US business hours only) this comprises approximately 25,000 records.

3.2 Trading Simulation using CGP Programs

In order to evaluate a trading strategy with historical data, it must be simulated. However, a simulation of a trading strategy can only ever be a rough approximation, simply because real trading has many other factors that are beyond the scope of a simulation. For example, brokers usually charge transaction costs on trades, but the charging scheme may vary from broker to broker and across time. Likewise, live data may contain errors that are subsequently cleaned in historical datasets. Historical data also does not contain information about slippage and other order filling problems. In the simulations described here, we assume no transaction costs and that there are no complications with order filling such as slippage or incorrect prices.

Given the assumptions, each CGP program was evaluated in the following way. The data (either the in-sample split during learning or the out-of-sample split during testing) was divided into days. It was assumed that each strategy would make one trade per day, at the start of the day, and that the trade would remain open until the last minute of same day. At that point it would be closed and the cumulative return or loss of the strategy updated. We do not simulate position sizes in these experiments – instead, the cumulative return is measured in points, which are a standard unit for measuring market prices. In the Euro/US dollar market, the standard point size is 0.0001, whereas for the Dow it is 0.01. This method of recording performance is ideal because it is independent of the size of the trades, which depends on many other factors (such as whether the amount invested is fixed or compounding, etc).

How does the CGP program decide which action (buy or sell) to take? Refer again to Figure 1. Each program has a single output node for each program, which if positive indicates a buy or long position for the following day, and if negative, indicates a sell or short position. There are seven inputs for each program corresponding to the closing prices of minute bars during the day prior to the trade. The exact minute bars are -1 (i.e. the closing price of the immediately previous day), -60 (the price 60 minute bars ago), -120, -180, -240, -300, and -360. Note that we skip minutes bars for which there is no trading activity or price change. These closing prices are mapped onto the input variables for the program, namely i_1, i_2, etc, which Figure 1 depicts as an example. The

[1] http://pitrading.com/, data obtained 2011

Table 2: Datasets used in the experiments (EURUSD=Euro/US dollar exchange rate; INDU=Dow Jones Industrial Average).

Dataset	In-Sample Period	Out-of-Sample Period	Out-of-Sample Size
$EURUSD_1$	1/5/2006 - 31/6/2006	2/7/2006 - 31/7/2006	26 days
$EURUSD_2$	3/1/2010 - 28/2/2010	1/3/2010 - 30/3/2010	27 days
$INDU_1$	1/5/2006 - 31/6/2006	2/7/2006 - 31/7/2006	20 days
$INDU_2$	3/1/2010 - 28/2/2010	1/3/2010 - 30/3/2010	23 days

Table 3: Parameters used by the canonical CGP algorithm.

Parameter	Value
n_{in}	7
n_{out}	1
n_l	30
n_m	6
$popsize$	5
$maxgens$	100,000

Table 4: Out-of-Sample Returns for Simple Positive and Negative Strategies, expressed in market points (0.0001 for EURUSD and 0.01 for INDU).

Dataset	Rtn(Pos)	Rtn(Neg)
$EURUSD_1$	-0.0077	0.0077
$EURUSD_2$	-0.0192	0.0192
$INDU_1$	-58	58
$INDU_2$	280	-280

inputs are thus a sample of the prices that occurred during the day leading up to the trade.

Besides the number of inputs and outputs, CGP also has a number of other parameters that must be specified. During initial experiments, we discovered that setting *maxgens* to a very high value such as 10,000,000 (as suggested in some references) resulted in programs that grossly overfitted the training data and therefore performed poorly on out-of-sample data. We therefore reduced the number of generations to 100,000 and obtained far better results.

We also found that a relatively high mutation was effective. In our setup, the total number of alleles is 91 (that being 30 function nodes plus 2×30 inputs per node plus 1 output node specification). We set the mutation rate n_m=6, which corresponds to approximately 6.5% of the alleles. Although this is higher than the recommended mutation rate (Miller, 2011), it resulted in better performance than a lower mutation rate. A summary table of the CGP parameter settings used in our experiments are shown in Table 3.

Finally, because CGP is a randomised algorithm, it is insufficient to run CGP only once per training/testing dataset and expect the results to be significant statistically. Instead, we repeated each experiment 100 times (i.e. we perform 100 independent trials per train/test split) and calculated the average and standard error of each of the three performance measures. We then used these values to calculate the 99% upper and lower confidence limits for each measure.

4 Results

In this section, we report on the results of our experiments and examine the types of program that CGP evolves for trading.

4.1 CGP Trading Strategy Performance

Before considering how strategies learned using CGP performed on the out-of-sample data, it is prudent to firstly consider how simplistic strategies perform. The most commonly used baseline method in trading strategies research is the buy and hold strategy; the equivalent of this in our context is a strategy that buys every day, which we refer to as a positive simple strategy. The opposite strategy to this is the sell everyday strategy, or negative simple strategy. We simulated these simple strategies and calculated the returns (in cumulative points) over the test period, which are given in Table 4.

Because these simplistic strategies are essentially opposites, one simplistic strategy is likely to make a profit and the other will make an equivalent loss, as the table demonstrates. The main problem in actually applying these simplistic strategies is deciding which one to take. As the table shows, if a unilateral decision were taken to follow the positive simple strategy (i.e. just buy every day), then a loss would have been incurred in three out of the four out-of-sample market periods.

Having covered the simple baselines, we now turn to the performance of CGP for trading strategy learning.

We assessed three different value/fitness measures. In each case, the objective of evolution was to find an individual that maximized the measure. The measures were: total cumulative return (i.e. net profit); negative maximum drawdown (negating drawdown makes small drawdowns more desirable); and the recovery factor.

CGP was run 100 times on each of the four in-sample datasets using one of each of the three different fitness measures just described. This yielded a total of $100 \times 4 \times 3 = 100 \times 12$ individual CGP runs. The in-sample best-of-run individual program was then tested on the corresponding out-of-sample data, and the average and standard error of the performance over 100 runs per combination of dataset/measure was calculated. We also calculated the 99% upper and lower confidence bounds for the average (which by definition is 2.58 standard errors above and below the sample mean). The results are given in Tables 5-8.

Examining these tables, we can make a number of observations.

Firstly, consider the recovery factor. Recovery factor is a ratio and therefore comparable across all markets despite their different units and different characteristics such as volatility. In every case, the average recovery factor is positive. Furthermore, in terms of statistical significance, the lower 99% confidence

Table 5: Out-of-Sample results for $EURUSD_1$ using three different in-sample optimization methods, 100 independent runs per method.

	Return Opt.			Negative MaxDD Opt.			Recovery Opt.		
	-MaxDD	Return	Rec.	-MaxDD	Return	Rec.	-MaxDD	Return	Rec
Avg	0.0216	0.0173	1.10	0.0263	0.0099	0.83	0.0226	0.0180	1.19
StdErr	0.0007	0.0016	0.13	0.0010	0.0023	0.17	0.0009	0.0018	0.17
Upper	0.0234	0.0215	1.43	0.0288	0.0158	1.26	0.0248	0.0227	1.64
Lower	0.0199	0.0130	0.78	0.0238	0.0039	0.41	0.0203	0.0133	0.75

Table 6: Out-of-Sample results for $EURUSD_2$ using three different in-sample optimization methods, 100 independent runs per method.

	Return Opt.			Negative MaxDD Opt.			Recovery Opt.		
	-MaxDD	Return	Rec.	-MaxDD	Return	Rec.	-MaxDD	Return	Rec
Avg	0.0374	0.0006	0.36	0.0344	0.0048	0.61	0.0263	0.0099	0.83
StdErr	0.0015	0.0028	0.10	0.0015	0.0031	0.13	0.0010	0.0023	0.17
Upper	0.0413	0.0078	0.63	0.0383	0.0128	0.96	0.0288	0.0158	1.26
Lower	0.0335	-0.0066	0.10	0.0304	-0.0032	0.27	0.0238	0.0039	0.41

bound on recovery factor, in all but one case, is also positive. This is a strong indication that the CGP method is effective.

However, the mean recovery factor is not always more than 1.0, which is desirable. For the 2006 EURUSD dataset, the average recovery factor is around 1.0, but it is much lower in the 2010 EURUSD dataset and the 2006 Dow Jones dataset. Surprisingly, the recovery factor is on average greater than 1.0 for the 2010 Dow Jones dataset.

The second result we will consider is the average return. Again, examining the tables, we see that while the returns are positive, they are often quite modest. For example, in the EURUSD 2006 dataset result shown in Table 5 the best return is 0.0031 or 31 points, which would only be significantly profitable if a significant investment was made (for a standard lot size of $100,000, this would amount to about $31 profit.) However, the simple negative strategy results shown in Table 4 are also quite modest at only 77 points, indicating that the market did not move far during the testing period.

Where the market did move a significant amount (for example, the Dow Jones 2010 dataset where the simple positive strategy records a $280 profit), the CGP strategies capture a significant chunk of that movement – a little over half of it with a net return of $169.35 on average.

Which of the three optimization measure is optimal in the experiments? An examination of the results shows that for the Euro/US dollar datasets, it is optimization of recovery factor that leads to the best on-average cumulative returns (those being 0.0180 and 0.099 for the 2006 and 2010 datasets respectively). For the Dow Jones datasets, optimizing negative maximum drawdown leads to the best cumulative returns.

Interestingly, in none of the four experimental datasets does direct optimization for in-sample return lead to the best out-of-sample return. Additionally, optimization for return is the only strategy that leads to a negative out-of-sample return, that being $-31.03 for the 2006 Dow Jones dataset. The lesson to be learned here seems to be that it is better to optimize for minimal drawdowns than it is to optimize directly for maximum return.

4.2 Analysis of Programs

In addition to the performance of CGP-based programs as trading strategies, we were also interested in the composition of the programs that were evolved. Figure 1 gives one specific example of a program that was evolved. To perform a more general analysis, we examined, for each of four datasets, the 100 best-of-run programs that were tested out-of-sample. Our analysis primarily concerned the frequency with which individual functions appeared in these programs. These frequencies are given in Table 9.

An examination of this table shows that that by far the most frequently selected operator is the subtraction $-$ operator, followed closely the comparison $<$ and $>$ operators, and then the $MEAN$ and MIN operators. These are indeed the type of operators that one would expect to see if designing an indicator-based trading system. Interestingly, functions representing constant outputs (C_1, C_{-1}, and C_0) are used very infrequently.

We also computed the average size of each best-of-run program for all four of the datasets. Those averages, in terms of the number of active nodes, are 6.16 and 6.21 for the EURUSD datasets and 4.58 and 5.34 for the INDU datasets. This shows that whereas programs of length up to 30 could have evolved, that many functions were not required and that the resulting programs were actually reasonably simple.

5 Conclusion

To conclude, an investigation of Cartesian Genetic Programming (CGP) with different objective functions for the purpose of learning trading strategies has been undertaken. CGP has been shown to be effective at learning strategies that often make a modest but significant net positive returns on data from two different markets and two different time periods. Furthermore, the method produces rules that are relatively simple, containing on average 5-6 functions per rule.

References

Barbosa R., Belo O. (2008)Autonomous Forex Trading Agents, in *Proc. 2008 International Conference*

Table 7: Out-of-Sample results for $INDU_1$ using three different in-sample optimization methods, 100 independent runs per method.

	Return Opt.			Negative MaxDD Opt.			Recovery Opt.		
	-MaxDD	Return	Rec.	-MaxDD	Return	Rec.	-MaxDD	Return	Rec
Avg	494.32	-31.03	0.29	354.85	232.30	1.01	440.02	56.71	0.35
StdErr	18.30	33.85	0.14	11.58	39.42	0.14	13.46	26.78	0.10
Upper	541.54	56.29	0.65	384.73	334.00	1.39	474.75	125.80	0.61
Lower	447.10	-118.36	-0.07	324.97	130.60	0.64	405.30	-12.39	0.08

Table 8: Out-of-Sample results for $INDU_2$ using three different in-sample optimization methods, 100 independent runs per method.

	Return Opt.			Negative MaxDD Opt.			Recovery Opt.		
	-MaxDD	Return	Rec.	-MaxDD	Return	Rec.	-MaxDD	Return	Rec
Avg	148.49	118.63	1.18	142.37	169.35	1.59	146.27	140.30	1.36
StdErr	5.55	13.68	0.14	4.32	11.09	0.21	5.67	14.03	0.16
Upper	162.80	153.92	1.55	153.52	197.95	2.14	160.89	176.49	1.76
Lower	134.18	83.34	0.81	131.22	140.74	1.04	131.66	104.11	0.96

on Data Mining, ICDM 2008, P. Perner Ed., LNAI 5077, pp. 389-403.

Beechey M., Gruen D., Vickrey J. (2000). *The Efficient Markets Hypothesis: A Survey*. Reserve Bank of Australia.

Contreras I., Hidalgo J., Nunez-Letamendia L. (2012), A GA combining technical and fundamental analysis for trading the stock marking. In *EvoApplications 2012*, Springer, pp.. 174-183.

Fama, E. (1970), Efficient Capital Markets: A Review of Theory and Empirical Work. *Journal of Finance* 25 (2): 383417. doi:10.2307/2325486. JSTOR 2325486.

Hirabayashi A., Aranha C., Iba H. (2009), Optimization of the Trading Rule in Foreign Exchange using Genetic Algorithm, in *Proc. GECCO'09*, pp. 1529-1536.

Larkin F. and Ryan C. (2010), Modesty is the Best Policy: Automatic Discovery of Viable Forecasting Goals in Financial Data. In *Proc. EvoApplications 2010*, Part II, pp. 202-211.

Lean Y., Lai K. (2007), *Foreign Exchange Rate Forecasting with Artificial Neural Networks*. Springer-Verlag.

Liu Z., Xiu D. (2009), An automated trading system with multi-indicator fusion based on D-S evidence theory in forex market, in *Proc. Sixth International Conference on Fuzzy Systems and Knowledge Discovery*, IEEE, pp. 239-243.

Miller J., ed. (2011), *Cartesian Genetic Programming*. Springer.

Ni H., Yin H. (2009), Exchange rate prediction using hybrid neural networks and trading indicators, *Neurocomputing*72:2815-2832.

Sekanina L., Harding S., Banzhaf W., Kowaliw T. (2011), Image Processing and CGP. In Miller (2011), pp. 181-216.

Sekanina L., Walker J, Kaufmann P, Platzner M. (2011), Evolution of Electronic Circuits. In Miller (2011), pp. 125-180.

Sewell, M. (2011), *Characterization of Financial Time Series*. Research Note RN/11/01, Dept. of Computer Science UCL.

Table 9: Percentage probability of a function being selected for a node in a best-of-run individual by dataset, over 100 independent runs per dataset.

Function	$EURUSD_1$	$EURUSD_2$	$INDU_1$	$INDU_2$
$+$	6.15%	4.94%	9.29%	6.31%
$-$	15.05%	10.37%	15.55%	11.87%
$\times$	8.58%	7.97%	6.91%	7.61%
$/$	11.33%	8.13%	8.86%	11.50%
$>$	11.97%	16.91%	12.96%	11.32%
$<$	12.30%	13.24%	12.53%	12.80%
MAX	10.36%	9.89%	5.40%	8.16%
MIN	8.25%	12.92%	8.86%	14.66%
$MEAN$	12.14%	7.97%	13.61%	10.39%
C_1	1.46%	1.91%	2.38%	2.97%
C_{-1}	1.29%	2.71%	1.94%	1.30%
C_0	1.13%	3.03%	1.73%	1.11%

Figure 1: An example of a program evolved using CGP. Function node 17 is the output node. Each function call has two inputs. Inputs may be input data (denoted by i_0, i_1, etc) or the output of another function node (denoted by an integer identifying the node). In the array representation in figure (a), active nodes are marked by marked by *. Figure (b) is the corresponding evaluation graph.

Node	F	Inputs	Node	F	Inputs	Node	F	Inputs
0*	$+$	i_5,i_6	10	$-$	6,i_1	20	MAX	i_4,i_7
1*	$-$	i_7,0	11	MIN	2,6	21	$\times$	3,5
2*	MAX	i_2,i_6	12*	$/$	1,i_1	22	MIN	i_3,10
3	$+$	i_7,0	13	$-$	10,10	23	$MEAN$	4,3
4	C_1	i_5,i_4	14	MAX	5,i_6	24	$MEAN$	3,i_5
5	$\times$	i_5,2	15*	$+$	i_5,9	25	$<$	8,15
6	MAX	0,4	16	$/$	i_1,2	26	$\times$	0,4
7	$\times$	i_7,0	**17***	$/$	15,12	27	$>$	i_1,24
8	C_1	1,2	18	$MEAN$	i_3,11	28	$+$	8,11
9*	MIN	i_5,2	19	$>$	8,3	29	$+$	4,2