Evolution of trading rules for the FX market
or
How to make money out of GP

Håkan Jonsson[1] Payam Madjidi[2] Mats G. Nordahl[1]

[1] Institute of Theoretical Physics
Chalmers University of Technology
S-412 96 Göteborg
Sweden

[2] Center for Parallel Computers
Royal Institute of Technology
S-100 44 Stockholm
Sweden

March 19, 1997

Abstract

Genetic programming is used to search for trading rules for the foreign exchange market using very high frequency data. Trading systems with retraining at regular intervals are studied with particular focus on the generalization from training to test data. These are found to generate higher returns than buy-and-hold and other algorithmically simple profitable strategies for these data.

1 Introduction

In this work we apply genetic programming [1] to the discovery of profitable trading rules for the foreign exchange market. Technical trading rules are quite commonly used by traders in the capital markets [2], and have been under much discussion, since their use contradicts the commonly accepted efficient market hypothesis (e.g., [3]).

In the academic literature, technical analysis has often been dismissed as futile, but a number of recent more careful statistical studies have shown

that these conclusions may need to be modified to some extent. In studies, e.g., by LeBaron [4], Taylor [5], and Levich and Thomas [6], trading rules indicating some degree of predictability for the foreign exchange market were found. One explanation for this could be the presence in the foreign exchange market of large actors (i.e., central banks) with other motives than profit. Some studies have attempted to explain at least part of the predictability by correlating it to periods of central bank intervention [7, 8]. The study of technical trading rules is also of interest in that it may reveal details about statistical time series properties of the data not captured by standard statistical tests, as pointed out, e.g., in [9, 10].

Several authors have tried to generate technical trading rules automatically using genetic algorithms or genetic programming (e.g., [11, 12]). Karjalainen [11] showed how genetic programming could be applied to find profitable and robust trading rules trading daily on the S&P500 stock market index, and used a statistical bootstrap approach to demonstrate forecastability. Pictet et al [12] applied genetic algorithms to the foreign exchange market, and in particular studied the effects of using various collective fitness sharing schemes.

One of the advantages of evolutionary methods in this context is their lack of bias in favor of certain easily comprehensible functions (such as comparisons between averages on different time scales) prefered by human traders. In this context it is reasonable to view the evolutionary algorithm more as a tool assisting a creative process [13] than an optimization algorithm, an aspect of genetic algorithms that in our opinion deserves more emphasis.

In section 2 of this paper, we describe and discuss the exchange rate data that was used; section 3 contains a description of our genetic programming implementation. Section 4 describes the results of experiments, in particular experiments where the population was retrained at regular intervals; and section 5 contains conclusions and suggestions for future work.

2 The data

In this work we use very high frequency data from the foreign exchange market, in the form of the data set HFDF-93 obtained from Olsen and Associates in Zürich. The data consists of bid and ask quotes for currencies collected from Reuter's FXFX page, and similar services from Telerate and Knight Ridder. Quotes are reported as often as every other second; for USD vs DEM, the data set contains 1472241 quotes during the period Oct 1 1992

at 0:00:00 GMT to Sep 30 1993 at 23:59:59 GMT.

Price quotes are published on the networks by market makers, but are not binding. Actual deals are made over the phone or over computer networks. Due to the decentralized nature of the market, it is essentially impossible to find data on actual transactions, which means that actual prices must be estimated from the published quotes.

The data may for example contain flaws in the form of delays and communication disruptions, so that the time ordering of the bids in the data can be inaccurate. Some quotes may also be misleading in the sense that they, possibly on purpose, deviate significantly from current transaction prices and do not correspond to any actual transactions (though erroneous quotes and obvious outliers were removed by O&A). This has to be taken into account when estimating prices.

An example of an actual time series of quotes is shown in Figure 1. For each market participant (excluding the least active ones), the bid prices have been connected together to visualize the behavior of different quoters. Prices should however still be regarded as valid only for a short period of time.

In our application, the price plays two different roles: the price that a strategy is allowed to trade at when it performs a transaction, and the series of prices a strategy uses as history information to determine its actions. We allow our strategies to trade at most once per minute. The price for a certain transaction (bid or ask — the bid price is the price when selling, the ask price corresponds to buying) is defined in terms of the logarithm of the bid (ask) price averaged over a certain time interval, or a certain number of transactions. This makes the price estimation less sensitive to occasional misleading data points and other errors. The interval chosen for averaging should be defined into the future, so that the strategies cannot evolve to take advantage of the price estimation mechanism.

In this way, transaction costs enter in a realistic fashion into the algorithm through the difference between bid and ask prices in the data. The spread varies significantly over time, as can be seen in figure 2. Note in particular peaks in the spread correlated to the opening and closing hours of major market centers.

When prices are used as historical information for a strategy, the average of bid and ask prices is used. The strategies have access to functions which compute, e.g., the average, maximum and minimum of the logarithm of the price over an arbitrary time interval. In this case the average of bid and ask prices is used (in future work we will also give strategies access to the

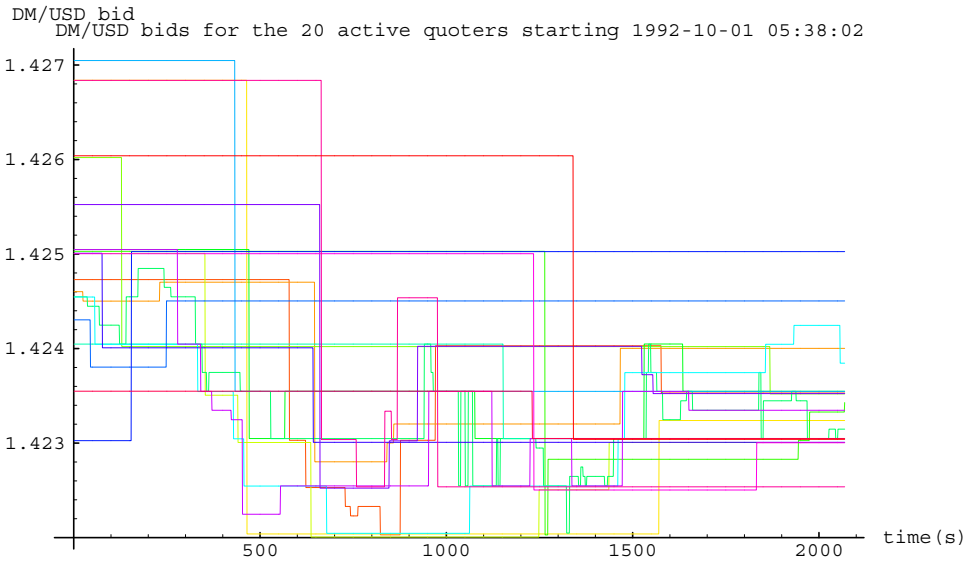


Figure 1: Quoted bid prices during a 35 minute interval of the data set. A bid is shown as a straight line until the next bid by the same quoter.

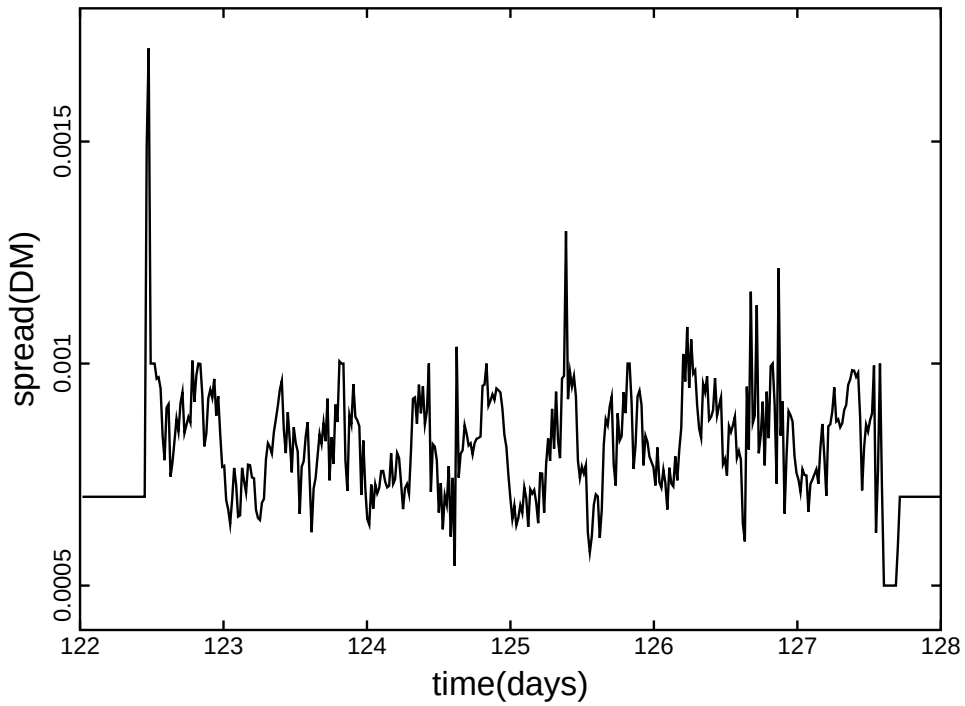


Figure 2: Price spread variation during a 6 day time period from the data set.

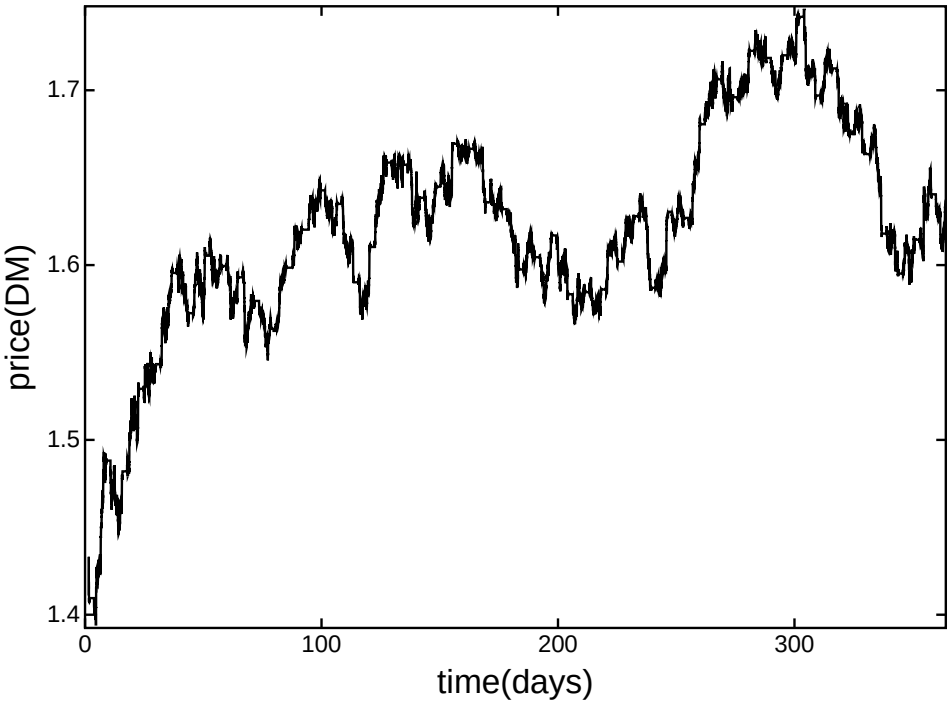


Figure 3: USD-DEM exchange rate during the time period from Oct 1 1992 to Sep 30 1993.

spread):

$$p(t_j) = (\log ask(t_j) + \log bid(t_j))/2 \tag{1}$$

Figure 3 shows a coarse grained view of the USD-DEM exchange rate for the entire time period that we are considering. We can see a significant increasing trend over this period, primarily during the initial month. This means that the algorithmically very simple buy-and-hold strategy performs very well, in particular when evaluated on the entire data set. We do not attempt to remove this trend, since we expect real exchange rate data to contain fluctuations on all scales, and any attempt at removing trends become an *ad hoc* procedure (note the difference to the stock market case, where it is reasonable to make comparisons with the interest rate of a risk-free bond). Instead strategies are compared to and required to outperform the buy-and-hold strategy.

3 Genetic programming

Genetic programming (GP) is a genetic algorithm (GA) [14] with hierarchical and non-fixed length genomes, which can be interpreted as programs. The term genetic programming was introduced by Koza [1]. The most common program representation used is that of trees, but other representations have been suggested (e.g., [15, 16]).

3.1 Strong typing

The genetic programming algorithm used in this paper is the classical GP paradigm extended with strong typing (STGP), first introduced by Montana [17]. The reason for using strong typing is the same as in ordinary programming: we want to represent our problem and its solutions in a natural way that reflects the structure of the problem. For example, comparing a time value with a price value is almost always meaningless. We also want to be able to interpret the solutions found, which can be very hard using standard GP. Strong typing can simplify this somewhat by preserving the semantics of the data processed by the functions.

To produce time values that depend on prices we then have to allow for multiple versions of functions and operators with different typings, e.g., the multiplication operator has one instance that multiplies time values and one that multiplies a price value with a time value.

Generating an initial population that is well diversified and at the same time contains a high proportion of genomes of reasonable fitness can be difficult. Keeping it that way during evolution is even harder. To avoid generating too many worthless rules we tune the selection of basis functions during generation by assigning suitable selection probabilities to the functions. Genetic operators preserving strong typing then at least ensure that syntactic constraints are maintained during evolution.

In our initial tests a large part of the randomly generated initial population consisted of worthless rules that either always evaluted to `TRUE` or `FALSE`, or that effectively performed that way by comparing quantities of very different magnitudes. In particular, exchange rates from the time series only vary between approximately 1.4 and 1.75 in the USD vs DEM time series, which has to be taken into account when making comparisons. To remedy this we (somewhat arbitrarily) introduced two types `Price` and `PriceConstant` to distinguish between values from the time series (or averages), and constants appearing in the functions. This allowed us to control that comparisons were only made between variables with comparable

ranges, and to significantly improve the fraction of viable functions in the initial populations and among mutants.

3.2 Function set

The function set used was the following, where `Price`, `PriceConstant` and `Time` are all real-valued types, P is a type variable that can be either a `Price` or a `PriceConstant` and t is the current time:

function	*type*	*description*
$<, >$	`Price` $\rightarrow P \rightarrow$ `Bool`	greater, less than
$*$	`Time` $\rightarrow P \rightarrow$ `Time`	multiplication
$*$	`Time` $\rightarrow P \rightarrow P$	multiplication
$+$	`Time` $\rightarrow$ `Time` $\rightarrow$ `Time`	addition
`norm` a b	`Price` $\rightarrow P \rightarrow P$	$\mid a - b \mid$
(`Time`)	`Price` $\rightarrow$ `Time`	
(`Price`)	`Time` $\rightarrow$ `Price`	
`and,or`	`Bool` $\rightarrow$ `Bool` $\rightarrow$ `Bool`	
`if` a `then` b `else` c	`Bool` $\rightarrow P \rightarrow P$	
`if` a `then` b `else` c	`Bool` $\rightarrow$ `Time` $\rightarrow$ `Time`	
`if` a `then` b `else` c	`Bool` $\rightarrow$ `Bool` $\rightarrow$ `Bool`	
`not`	`Bool` $\rightarrow$ `Bool`	
`price`	`Price`	current price
`avg` a b	`Time` $\rightarrow$ `Time` $\rightarrow$ `Price`	average price over $[t-b-a, t-b]$
`max` a b, `min` a b	`Time` $\rightarrow$ `Time` $\rightarrow$ `Price`	max/min price over $[t-b-a, t-b]$
`lag` a b	`Time` $\rightarrow$ `Time` $\rightarrow$ `Price`	value of a at $t-b$
`lag` a b	`Price` $\rightarrow$ `Time` $\rightarrow$ `Price`	

Type conversion functions are allowed in the initial population with a small probability, so as not to exclude them entirely from the search space. The price time series is supplied as an implicit argument to the functions and does not appear explicitly in the function set. The output from the rules is a Boolean variable which if `TRUE` says that a trader's assets should be placed in DEM at the next time step, and otherwise in USD.

The terminal set consists of real values typed as `Price`, `PriceConstant` and `Time`. The maximum initial `Time` constants were typically set to 2592000.0 seconds (one month), and variables of type `PriceConstant` were given values between 0.0 and 2.0.

The function set above is similar to that used by Karjalainen [11], and is rather biased towards rules reminiscent of those used by human traders.

While this simple set is a good starting point, we consider it important to also investigate more diverse function sets.

3.3 Population dynamics

The selection mechanism used was ranked selection, where genomes are ranked according to their fitness values, and selected for reproduction according to their rank, with a distribution skewed linearly towards the highest rank according to a selection pressure parameter.

Reproduction was performed by selecting two parents, performing crossover or mutation, evaluating the fitness of the offspring and inserting it into the population. A total of N individual reproductions (with N the number of genomes in the population) is called a generation.

The crossover operator is the standard GP crossover constrained by strong typing: a crossover point is randomly selected in parent A and a crossover point of the same type is selected in parent B, and subtrees are then swapped. Only one of the offspring is kept. The mutation operator is defined in terms of crossover with a randomly generated tree.

3.4 Fitness measure

The fitness of a trading rule is measured as its return over a specified time, where return is defined as the ratio between accumulated capital and initial capital. The fitness depends on what prices rules are allowed to trade at, which as discussed above involves a price estimation.

In practice an automated trading system could receive information of a new bid, make a decision and complete a transaction in a matter of seconds. Several bids on the market are valid at the same time, but would usually be considered invalid after a few minutes. We have tried several different schemes for estimating prices, such as using averages over a certain time interval following the transaction, or an average over a certain number of bids. In the runs below, we used the simplest choice, namely to define the trading price as the price of the first bid following the time at which the decision to buy/sell was made.

4 Results

The main purpose of the experiments below was to investigate the effects of continuous retraining in a trading system. One would expect a temporary regularity discovered by an automated trading system to dissipate over time

as other market participants adjust their behavior, so that a rule that was profitable at some point would lose its value, and the system would need to readjust its behavior.

This could be adressed by retraining the system at regular intervals. In general, one could retrain at time intervals $T_{retrain}$, using historical data from a time interval $T_{history}$, which could be different from $T_{retrain}$.

The system could either be retrained from a random initial population each time, or the entire population (or a fraction of it) could be kept as an initial state for the training algorithm. In the runs below, the system was trained on a period of thirty days and then tested on the consecutive thirty days. This was done for eleven consecutive months of data. Retraining was done from scratch, to avoid problems due to lack of diversity in the population. These design choices clearly need further study.

A second issue in the design of a trading system is what rules to actually use given a population trained on some training set. The rule that is best adapted to the training set is not an obvious choice, since the population may contain some rules that are overfitted to the training data and generalize poorly. Lacking a statistical theory of learning in populations similar to the statistical learning theory for neural nets [18], we simply evaluate the system by considering average results for the n best rules in the population. Another approach, which may be preferable, would be to use a majority decision among the n best rules.

In the runs below, the strategies were allowed to trade every 120 seconds, which gives 21600 strategy evaluations each month. The population size was 500, the mutation probability 0.3 and 200 generations were run each month. The maximum number of nodes in a tree was 40, and the minimum depth was 20 nodes.

Figure 4 shows learning curves for each of the training months. In general, convergence is fairly rapid. The results for each month are shown in table 1. A comparison with the buy-and-hold strategy is also included.

A surprising feature is the correlation between the performance on training and test sets. The correlation coefficients in table 1 take the entire training and test sets into account; in figure 5 we show the distribution of results on training and test sets for individual months with the worst rules truncated. Since transaction costs are included, it is possible to invent trading rules that almost always perform poorly, such as rules which make a very large number of random trades. Rules with very low fitness both on training and test set will contribute to the correlation, but even with arbitrary restrictions to rules that perform reasonably well there is still a significant correlation. Figure 5 also shows that successful strategies typically perform

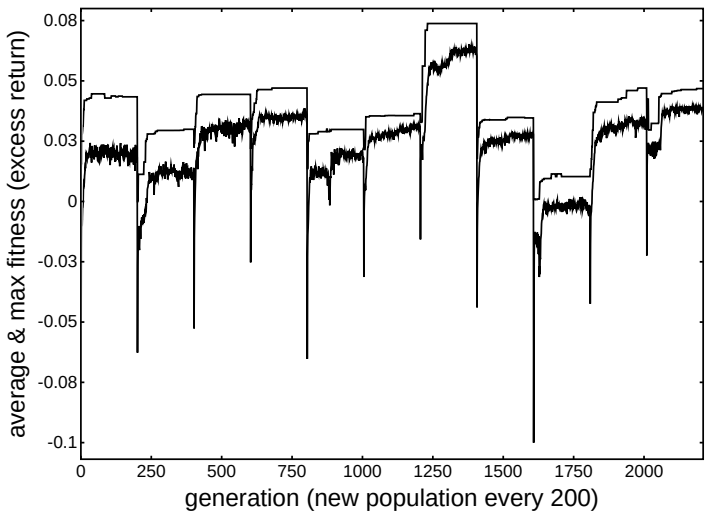
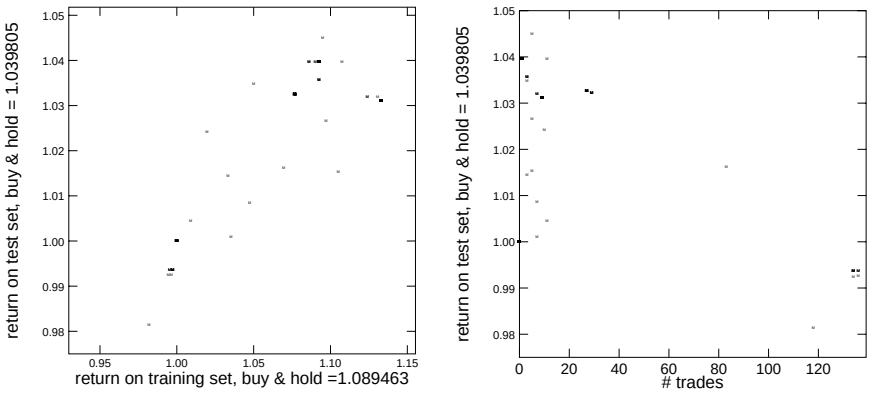


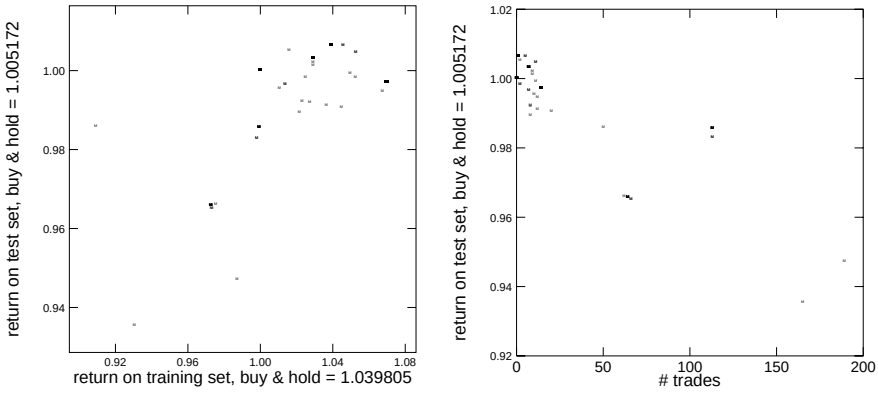
Figure 4: Evolutionary progress during a run where the system was retrained every month using one month of data. The graphs shows the success on the training set.

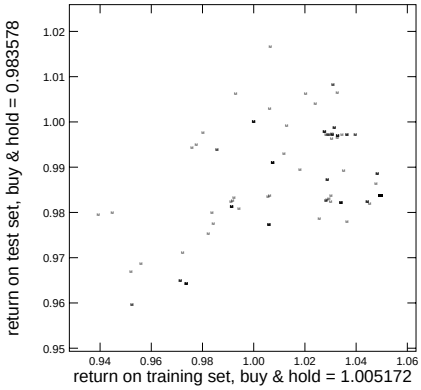
only a small number of trades per month.

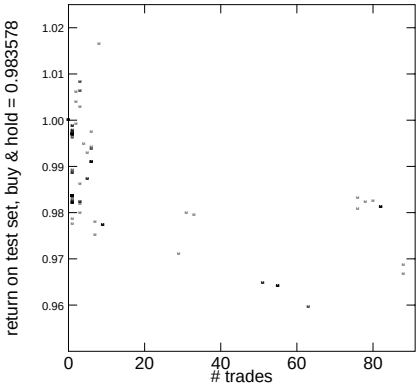


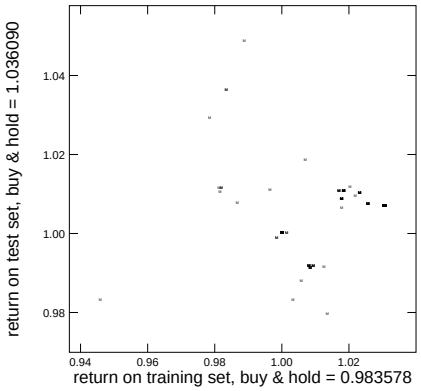
month	*correlation*	*max return*	*mean return*	*max excess return*	*mean excess return*
1	0.88428	1.044998	1.02546	0.005193	-0.014341
2	0.85155	1.006547	0.99563	0.001375	-0.009541
3	0.90121	1.016482	0.98270	0.032904	-0.000882
4	0.91319	1.048793	1.00343	0.012703	-0.032666
5	0.93553	1.022292	0.99935	0.034017	0.011075
6	0.90094	1.000000	0.98209	0.029305	0.011395
7	0.86631	1.025961	1.01185	0.022798	0.008683
8	0.89437	1.078831	1.06168	0.001598	-0.015551
9	0.82117	1.022616	1.00013	0.019290	-0.003196
10	0.88877	1.004899	0.98139	0.034550	0.011034
11	0.92877	1.007727	0.99812	0.028086	0.018477

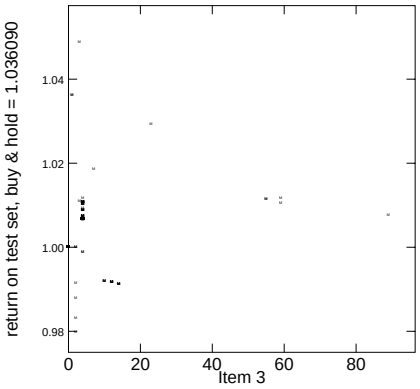
Table 1: Results on eleven months of test data.

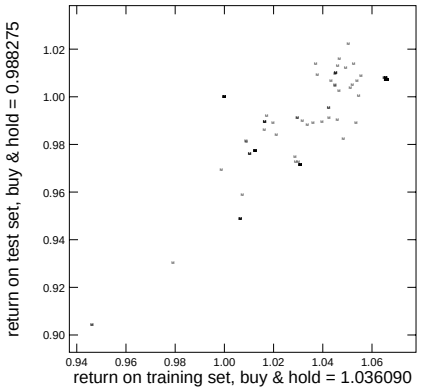


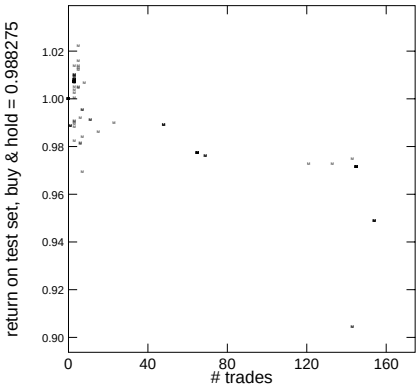

return on test set, buy & hold = 0.983578
return on training set, buy & hold = 1.005172

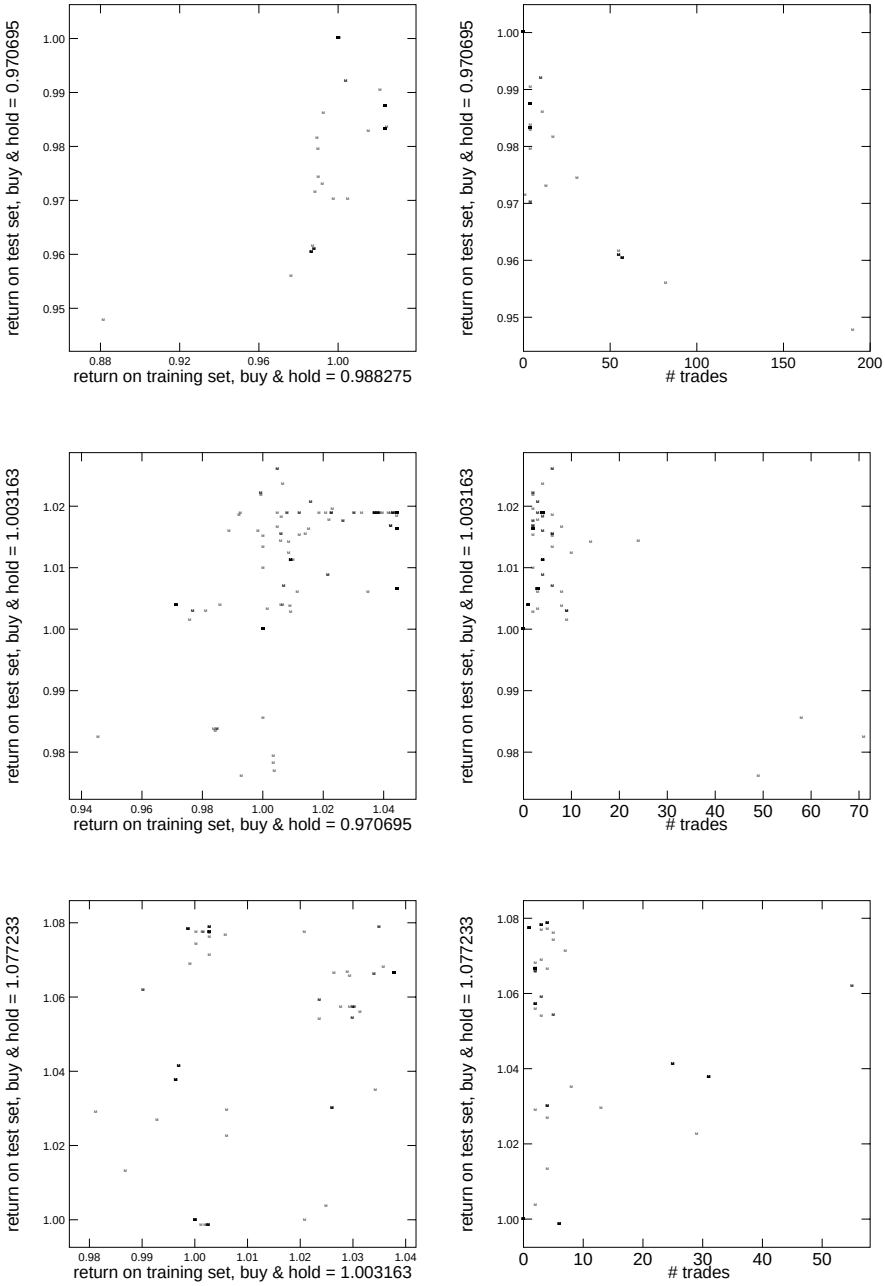

return on test set, buy & hold = 0.983578
trades

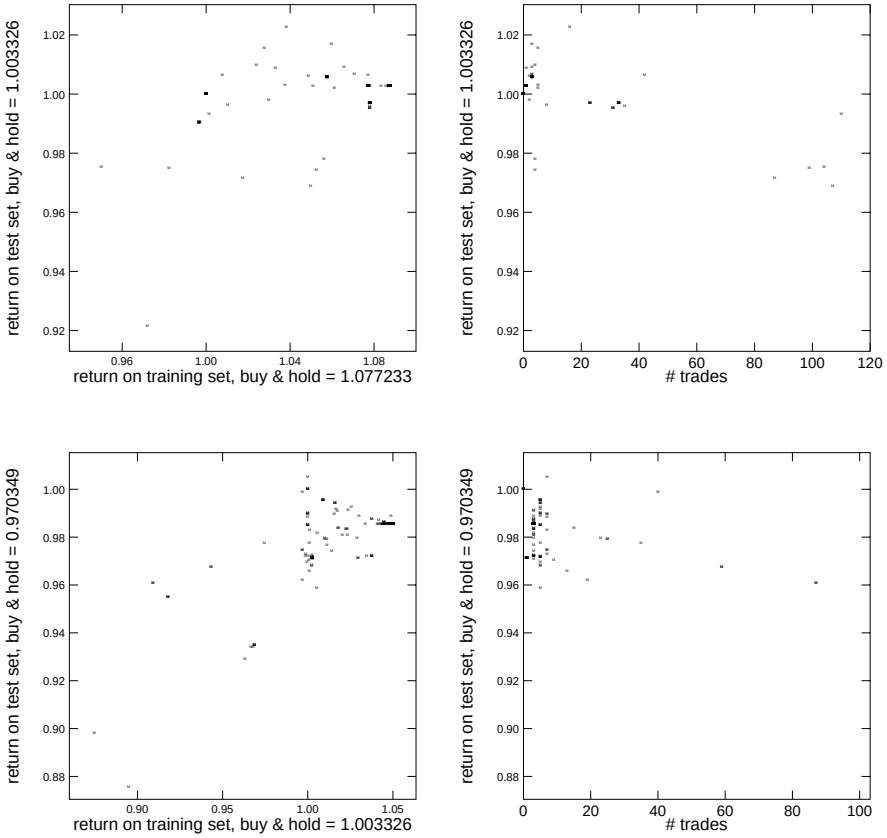

return on test set, buy & hold = 1.036090
return on training set, buy & hold = 0.983578


return on test set, buy & hold = 1.036090
Item 3


return on test set, buy & hold = 0.988275
return on training set, buy & hold = 1.036090


return on test set, buy & hold = 0.988275
trades


return on test set, buy & hold = 0.970695
return on training set, buy & hold = 0.988275
trades
return on test set, buy & hold = 1.003163
return on training set, buy & hold = 0.970695
trades
return on test set, buy & hold = 1.077233
return on training set, buy & hold = 1.003163
trades


return on test set, buy & hold = 1.003326
return on training set, buy & hold = 1.077233
trades
return on test set, buy & hold = 0.970349
return on training set, buy & hold = 1.003326
trades

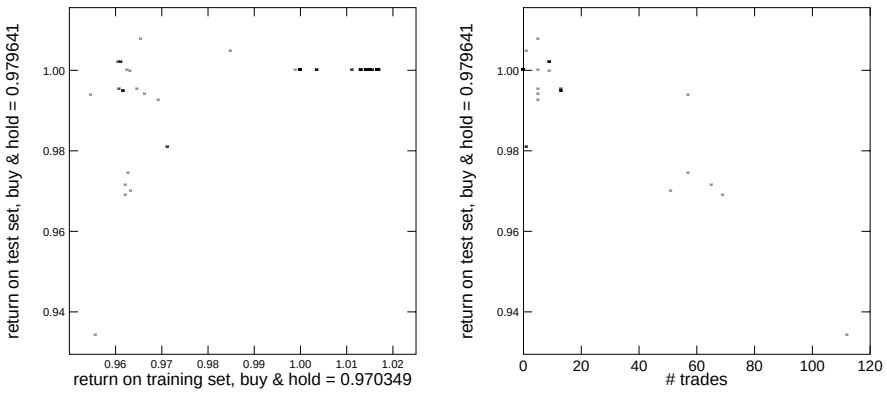


Figure 5: Relation between results on training and test sets for 11 consecutive pairs of months (the diagrams have been truncated to exclude rules of very low fitness).

For trading, we would select a certain number of rules from the trained population. If we consider averages among the 10 best rules in the poulation, we obtain a mean excess return was 0.002600, and the mean return 1.007812. If the rules had been used to trade on actual data and had kept their capital from month to month, the accumulated return would have been 1.086773. This can be compared to the buy and hold strategy, which yields an accumulated return of 1.053119, with an average return of 1.005212 per month. Another algorithmically simple strategy which the system easily could discover would be that which each month invests in the currency that performed best the previous month. This rule gives an accumulated return of 1.0655, and is also outperformed by the system. An example of an evolved rule is shown in figure 6.

4.1 Bootstrap tests

In this section, we discuss evaluation of the evolved trading rules through their performance on resampled data [19], where comparison time series are generated by scrambling the increments of the original series at random. A more detailed study will appear elsewhere; in particular tests against other statistical hypotheses than a random walk, such as GARCH, will also be performed.

To begin with, a random choice of the 10 best rules from each month

month	*buy and hold*	*return*	*excess return*
1	1.039805	1.034031	-0.005774
2	1.005172	0.998728	-0.006444
3	0.983578	0.987353	0.003775
4	1.036090	1.007253	-0.028837
5	0.988275	1.004813	0.016538
6	0.970695	0.990618	0.019923
7	1.003163	1.017387	0.014224
8	1.077233	1.059586	-0.017647
9	1.003326	1.001941	-0.0013853
10	0.970349	0.984217	0.013868
11	0.979641	1.000000	0.020359

Table 2: Results on eleven months of test data with 10 rules selected for trading.

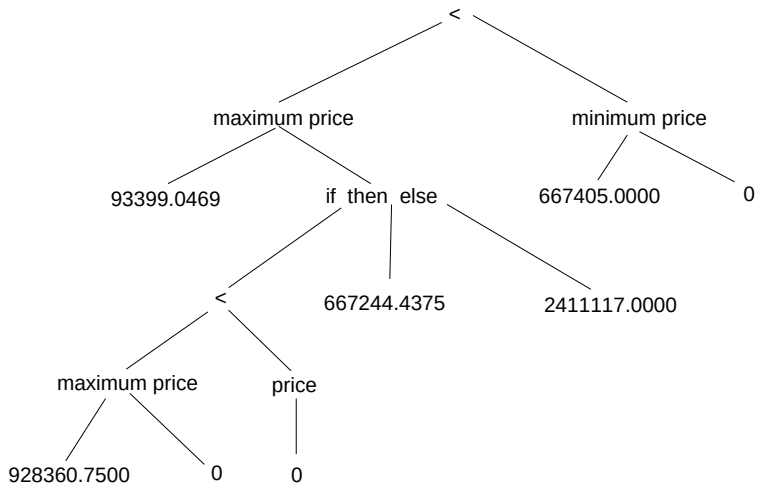


Figure 6: An example of a successful trading rule.

was tested over the entire year against data scrambled individually for each month (one could also have considered scrambling the entire time series, which would have been a less stringent test).

An interesting issue is how to estimate transaction costs in a reasonable way, since scrambling the data will remove the obvious structure present in the spread. We have chosen to take the trading cost in the tests equal to the minimum trading cost available in the subset of the price series under consideration (which introduces a bias against our system).

We first show results from testing the system over an entire year, with trading rules selected at random from the 10 best rules each month. Figure 7 shows the distribution of annual returns for 10^6 realizations of this procedure. The bizarre peaks at lower returns are due to the limited diversity of the population for certain months (as discussed below), which effectively may lead to a random choice between a couple of rules with very different fitness for that month. Of the resamplings considered, only a fraction of 0.08 generate a higher return than the annual return (1.086773) of our system, a satisfactory result.

The performance distribution for the random walk data is also shown for each month individually, where the 10 best rules from each month were tested on 100 cases of resampled data. Here we can clearly see that the performance of the learning algorithm varies quite a lot between months. In some case, no individual in the population performs any trades at all on the scrambled data. In these cases an undesirable degree of convergence in the population was present, which indicates that the performance could be improved by better tuning of parameters to ensure greater diversity.

5 Conclusions and discussion

The main result of the paper is the surprisingly high correlation between the performance on training set and test set. Since transaction costs are included, it is reasonably easy to come up with trading rules that almost always perform poorly, for example by making a very large number of random transactions, which is a partial explanation for this correlation. We also find that our strategies (or more precisely an average over the 10 best evolved rules each month) outperform the buy-and-hold strategy and the strategy which invest in the currency that performed best the previous month. These are algorithmically simple strategies which are reasonably succesful on this data set, and which the system could easily have discovered.

A number of issues warrant further investigation. In the work reported

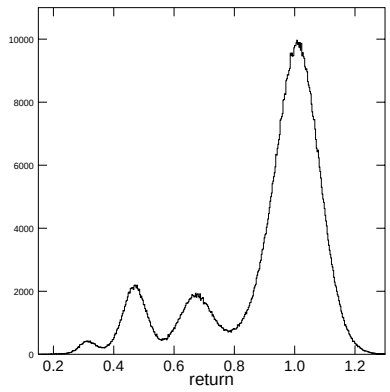


Figure 7: Distribution of yearly returns for data with randomly scrambled increments.

here, no risk measure was included in the fitness function. The extent to which the evolved strategies trade risk against profit needs to be investigated.

More careful testing of the statistical validity of the results is also necessary, as discussed in the section on bootstrap methods (see [20] for a discussion of how to test predictors).

Extending the function set to include such functions as spread, volatility and tick frequency will allow further exploitation of the available price series.

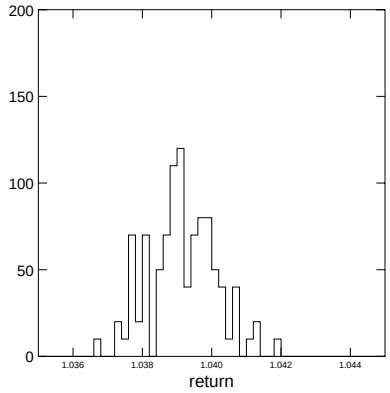
6 Acknowledgements

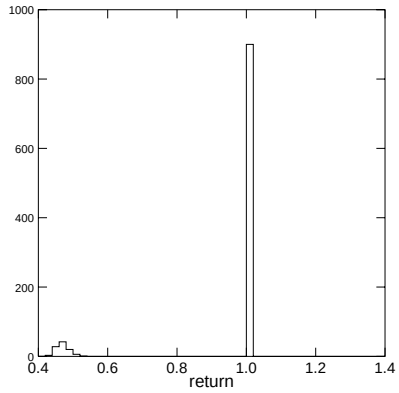
This research was supported by NUTEK (Swedish National Board for Industrial and Technical Development).

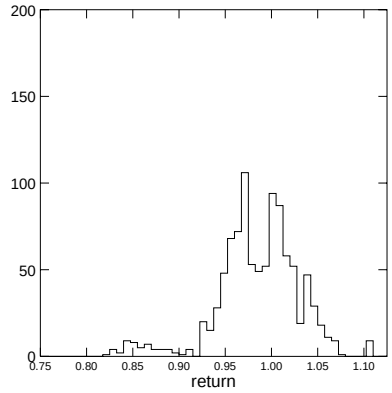
References

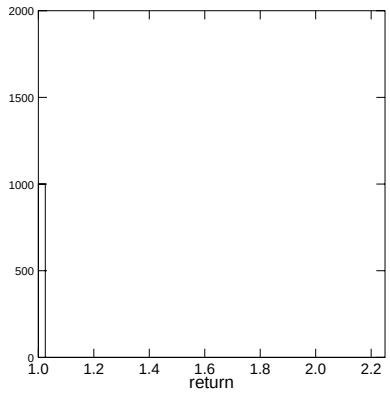
[1] J. R. Koza, *Genetic Programming* MIT Press, Cambridge MA, 1992.

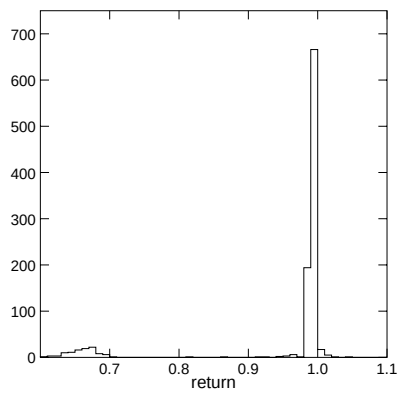
[2] M. Taylor and H. Allen, The use of technical analysis in the foreign exchange market, *Journal of International Money and Finance*, **11** (1992) 304–314.


200
150
100
50
0
1.036
1.038
1.040
1.042
1.044
return


1000
800
600
400
200
0
0.4
0.6
0.8
1.0
1.2
1.4
return


200
150
100
50
0
0.75
0.80
0.85
0.90
0.95
1.00
1.05
1.10
return


2000
1500
1000
500
0
1.0
1.2
1.4
1.6
1.8
2.0
2.2
return


700
600
500
400
300
200
100
0
0.7
0.8
0.9
1.0
1.1
return

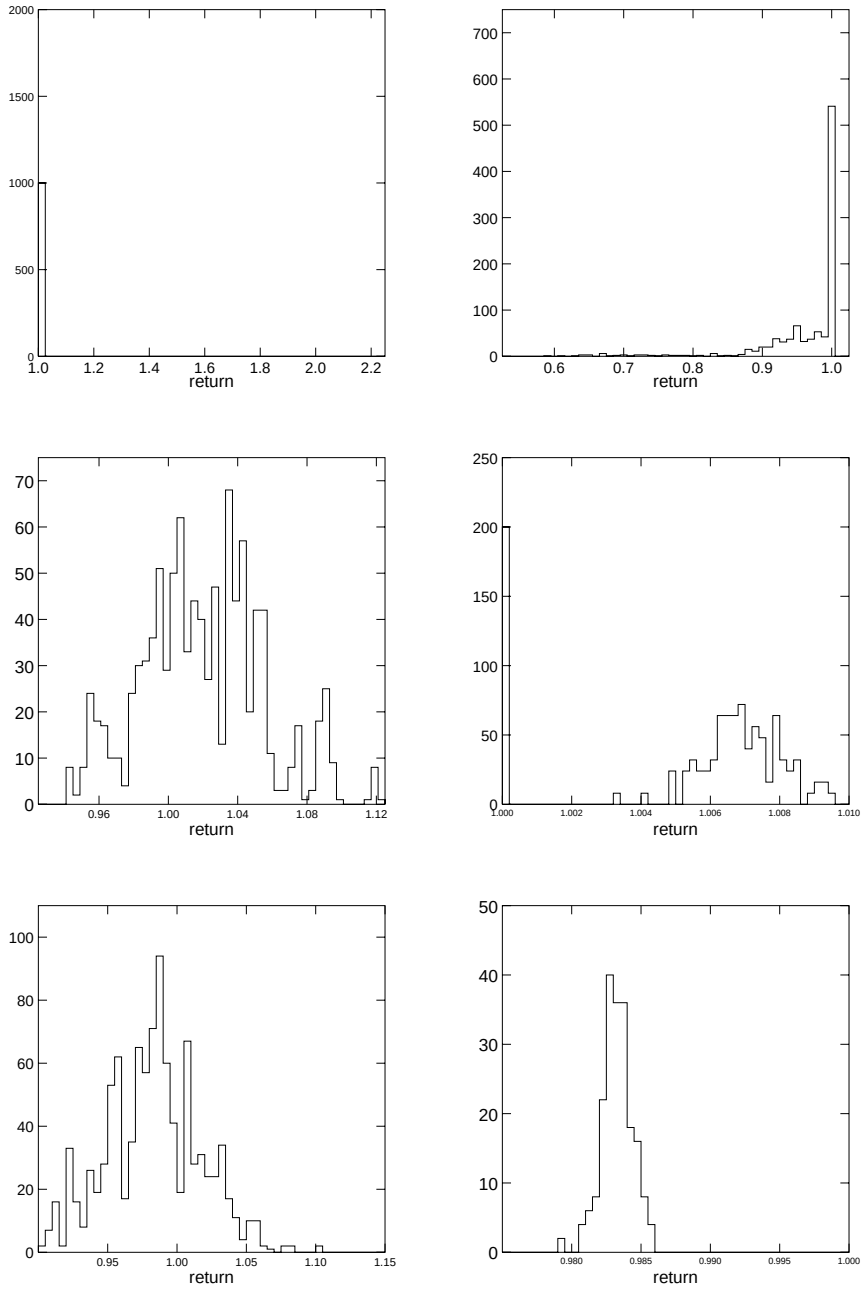


Figure 8: Monthly distributions of performance on resampled data

[3] E. F. Fama, Efficient capital markets: a review of theory and empirical work, *Journal of Finance,* **25** (1970) 383–417.

[4] B. LeBaron, Technical trading rules and regime shifts in foreign exchange, Santa Fe Institute, Economics Research Program, working paper 91-10-044.

[5] S. Taylor, Rewards available to currency futures speculators: Compensation for risk or evidence of inefficient pricing?, *Economic Record*, **68** (1992) 105–116.

[6] R. M. Levich and L. R. Thomas, The significance of technical trading-rule profits in the foreign exchange market: a bootstrap approach, *Journal of International Money and Finance*, **12** (1993) 451–474

[7] B. LeBaron, Technical trading rule profitability and foreign exchange intervention, Technical report, University of Wisconsin–Madison, 1994.

[8] W. Silber, Technical trading: when it works and when it doesn't, *Journal of Derivatives*, Spring 1994, 39–44.

[9] W. A. Brock, J. Lakonishok, and B. LeBaron, Simple Technical Trading Rules and the Stochastic Properties of Stock Returns, *Journal of Finance*, **47** (1992) 1731–1764.

[10] E. Acar and P. Lequeux, Trading rules profits and the underlying time series properties, in Proceedings of HFDF-1, First International Conference on High Frequency Data in Finance, Olsen and Associates, Zurich, 1995.

[11] R. Karjalainen, Using genetic algorithms to find technical trading rules in financial markets. Ph.D. Thesis, Wharton School of the University of Pennsylvania, 1994.

[12] O. V. Pictet, M. M. Dacorogna, R. D. Davé, B. Chopard, R. Schirru, and M. Tomassini, Genetic algorithms with collective sharing for robust optimization in financial applications, Technical report, Olsen and Associates, 1996.

[13] K. Sims, Artificial evolution for computer graphics, *ACM Computer Graphics*, 25 (1991) 319-328.

[14] J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.

[15] M. Keijzer, Representing computer programs in genetic programming, M. Sc. thesis, Leiden University, 1995.

[16] P. Nordin, A compiling genetic programming system that directly manipulates the machine code, in *Advances in genetic programming,* K. E. Kinnear, ed., MIT Press, 1994. 311–334.

[17] D. J. Montana, Strongly typed genetic programming, *Evolutionary Computation*, 3 (1995) 199–230.

[18] H. S. Seung, H. Sompolinsky, and N. Tishby, Statistical mechanics of learning from examples, *Physical Review* A 45, (1992) 6056–6091.

[19] B. Efron, Bootstrap methods: another look at the jackknife, *Annals of Statistics*, 1 (1979) 1–26.

[20] A. Weigend and B. LeBaron, A bootstrap evaluation of the effect of data splitting on financial time series, *IEEE Transactions on Neural Networks*, to appear.