2014 13th International Conference on Machine Learning and Applications

A Hybrid Genetic-Programming Swarm-Optimisation Approach for Examining the Nature and Stability of High Frequency Trading Strategies

Andreea-Ingrid Funie
Department of Computing
Imperial College London
London, United Kingdom SW7 2AZ
Email: andreea.funie09@imperial.ac.uk

Mark Salmon
Faculty of Economics
University of Cambridge
Cambridge, United Kingdom CB3 9DD
Email: mhs39@cam.ac.uk

Wayne Luk
Department of Computing
Imperial College London
London, United Kingdom SW7 2AZ
Email: w.luk@imperial.ac.uk

***Abstract*—Advances in high frequency trading in financial markets have exceeded the ability of regulators to monitor market stability, creating the need for tools that go beyond market microstructure theory and examine markets in real time, driven by algorithms, as employed in practice. This paper investigates the design, performance and stability of high frequency trading rules using a hybrid evolutionary algorithm based on genetic programming, with particle swarm optimisation layered on top to improve the genetic operators' performance. Our algorithm learns relevant trading signal information using Foreign Exchange market data. Execution time is significantly reduced by implementing computationally intensive tasks using Field Programmable Gate Array technology. This approach is shown to provide a reliable platform for examining the stability and nature of optimal trading strategies under different market conditions through robust statistical results on the optimal rules' performance and their economic value.**

I. Introduction

Two elements drive trading in high frequency markets: the traditional external economic forces of demand and supply, as captured by the order book of a market, and forces that arise from within the market structure itself, as liquidity moves to balance risk structure in the market. Simple mechanical or technical trading rules based on visual patterns (e.g "head and shoulders"), have been examined for many years in the context of equity and currency markets [1] but with modern technology it is possible to go much further in the design of trading rules.

Machine learning related research presents a number of approaches, all of which tend to be highly computationally expensive, limiting the testing and evaluation process. In addition, a single dominant strategy with suitable performance and stability under a range of different market conditions has yet to be found, thus the need for real-time adaptation to market characteristics. This adaptation implies learning alongside the optimisation algorithm. For such adaption to perform optimally, market conditions must be monitored and fed back to the algorithm in real time to achieve maximum performance. It is thus apparent that the requirement for computational speed becomes paramount both for regulation and trading. This leads to the current situation where high frequency markets are being driven by machine-based trading algorithms that are able to analyse huge streams of data in real-time using advanced hardware and software. Recent developments in hardware acceleration are now commonly adopted by large investment firms in the design and implementation of such trading strategies. This has been made possible through the use of reconfigurable hardware, provided by companies such as Maxeler Technologies. Such developments enable the efficient use of flexible run-time reconfigurable algorithms which are able to rapidly react to changing market conditions.

With these considerations in mind, this study seeks to find profitable trading patterns, or robust predictive structures that can be employed with confidence in the market in real time. This is achieved using artificial intelligence optimisation, hardware acceleration and pattern recognition techniques such as genetic programming. The approach is developed using tick level Foreign Exchange (FX) data, with a focus on examining predictable patterns in varying market conditions and regimes.

II. Trading Platform

Given historical market prices, order book structure, numerical constants, mathematical and statistical operators as inputs, the algorithm will identify optimal trading rules, constructed from a combination of these inputs under certain economic conditions, taking into consideration our minimum gain stopping criteria.

The use of simple evolutionary algorithms provides clear evidence in terms of profitability over standard technical indicators. However, reliability issues arise in market conditions that lack significant predictability [2], and we aim to explore not only good trading rules but also those states of the market in which no trading should take place.

A. Hybrid Evolutionary Algorithm

The genetic programming method provides an effective way to search for both linear and non-linear trading rules. We can then evaluate predictability as widely as possible, without imposing restrictions on the form of model, predictor or trading rule. A trading rule is built as a binary logical tree, which produces true (1) (buy) or false (0) (sell) signals given the set of constructed input variables. The rules are then represented in the form of randomly created binary trees with terminals and operations (mathematical and binary) in their nodes.

A GP is constructed using different individual elements such as: functions, terminals, fitness criterion, genetic operators, variable length programs and population initialisation.

In our implementation we make use of the *generational* approach, where a new generation replaces the old generation and the program cycle continues. Our *terminal set* contains the variables, which take their values from market data and are updated every time new information arrives in the market. Thus, it allows the conditioning information sets to update the trading rule in real time. The algorithm explicitly computes logarithmic values of the conditioning variables, moving average values, and maxima and minima over different periods. The terminal set also includes real numbers and specific market price values (e.g: volume on the bid, depth on the ask side, etc.) as terminal constants. Each of these inputs becomes part of the GP training and evaluation data sets as a GP terminal. Our *function set* used to define the technical rules consists of the binary algebraic operations, max, min, binary order relations, logical operations, and unary functions (e.g: absolute value, change of sign). Such a broad range of functions enables the production of a huge set of non-trivial strategies.

Once we have decided on the terminal and function sets, the *initialisation* of a tree structure becomes straightforward. There are two common methods for initialising tree structures: *full*, and *grow*. The *grow* method (see figure 1) produces irregular shape trees, because nodes are selected randomly from the terminal and function sets throughout the entire tree (except the root node which uses only a logical or binary operator). Once a branch contains a terminal node, that branch has ended, even if the maximum depth has not been reached. Conversely, the *full* method chooses only functions until a node is at the maximum depth (16 in our case). Then it chooses only terminals resulting in a complete binary tree of the maximum allowed depth. Unfortunately neither of the above methods used exclusively

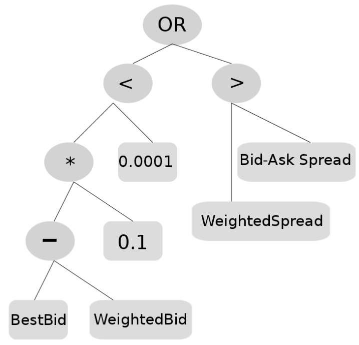

Fig. 1. Trading rule grow initialisation - maximum depth 5

will optimally exploit our design, potentially resulting in a uniform set of structures for the initial population. According to more recent studies [5], a better approach is the *ramped half-and-half method*. Such a method calls for the equal division of the population among the initialised individuals, with trees having all depth sizes up to and including the maximum allowed. For each depth group, half of the trees are initialised with the *full* technique and half with the *grow* technique.

Evolution proceeds by transforming the initial population through the use of genetic operators, which in our case are: *Crossover, Mutation and Reproduction*. In our design, mutation is not performed on the fittest 25% of the population after which it occurs at a rate of 0.2. We use a crossover probability of 0.2 and that is applied over all individuals in the population. The reproduction probability is set to 0.05. We have experimented with a range of values for these parameters and these choices have provided the most consistent results across the entire data set.

The algorithm's *fitness function* is calculated on the training data set and provides continuous feedback regarding performance for the relevant data. Performance (profitability) is measured by computing the *cumulative returns* from the following allocation strategy: according to the signal provided by the trading rule, the trader buys or sells 1 million GBP. Using this approach we control the potential price impact of the trade, through ensuring necessary liquidity for completing the minimum specified transaction size. This assumption limits the potential profitability when the trading signal is strong and our results will therefore tend to underestimate the potential profitability of the trading rules. When new information arrives from the market, the system can re-evaluate the trading signal and adapt to the new market conditions by updating their position accordingly (e.g: buy, sell) [2](see formula 1).

$$R_c = \Pi_t (1 + z_t * r_t) - 1 \tag{1}$$

where r_t = (p_t - p_{t-1}) / p_{t-1} is the one-period return of the exchange rate and p_t corresponds for best bid or best ask price.

To control the frequency of trading, we add a *trading threshold* to the strategy. According to this, the trader is allowed to trade only if the exchange rate exceeds at any point $+k$, $-k$, relative to the last transaction price. Formally, let's define: z_t = - 1 for a short sterling position and z_t = 1 for a long sterling position and p_t the price at time t and p_{t1} price at time t_1 where t_1 is the time of the trader's last transaction. When

$$|p_t - p_{t1}| >= k \tag{2}$$

the trader is allowed to re-evaluate its position. The parameter k is used to filter out weak trading signals and determines an "inertia band" that prompts the trader to trade only once the exchange rate exceeds the value of a certain characteristic by a value of k. In our case the characteristic followed is represented by the past exchange rate values and is tightly related to the *filter trading strategy*[1] which can also accommodate accounting for transactions costs beyond the bid-ask spread. It was shown [3] that in the face of Knightian uncertainty incomplete preferences may lead to an absence of trading, thus the importance of such a filter threshold which should be part of the trader's strategy. For example, if k=0 then trades can take place, every time the mid-quote of the exchange rate changes, leading to an unreasonably large number of transactions given transactions costs. As k increases, the trading frequency drops.

The algorithm uses a *ranking selection* based on the fitness order into which the individuals can be sorted. After each iteration of the algorithm, trading rules that have poor performance according to the fitness function are removed from the population, and only the more profitable ones survive

[1]A filter trading strategy is a trading strategy where technical analysts set rules for when to buy and sell investments, based on percentage changes in price from previous lows and highs. [4]

to carry their structure onwards to create new trading rules. At each point in time, as the size of the training set evolves, this algorithm is replicated over a number of iterations until ultimately it converges to the trading rule that achieves the best in-sample performance given the conditioning information.

We decided to implement a Hybrid Genetic Programming (GP) and Particle Swarm Optimisation (PSO) algorithm as shown in figure 2. The additional complexity, which decreases

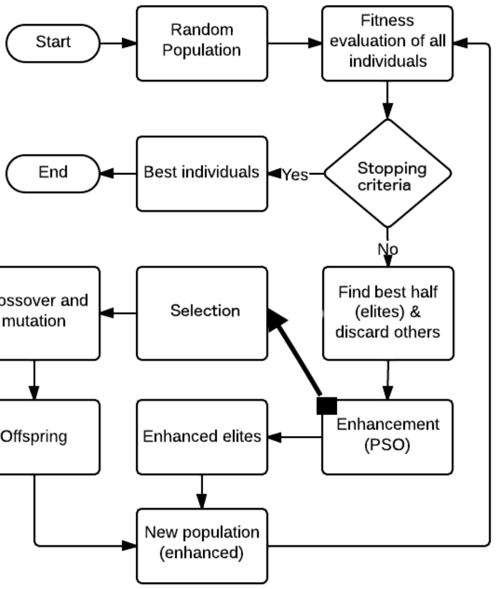

Fig. 2. Hybrid GA/PSO algorithm flow chart

the execution speed of the program significantly increases its performance. As we have previously mentioned, we apply the genetic operators after the fitness selection step. We perform the fitness selection and then apply the PSO to the best half of the trading rules. Afterwards, we apply the genetic operators on only the new rules and send the offspring and the enhanced rules into the new population. The PSO search by its nature, will seek to find the global minimiser or maximiser of a function that we give it as input, resulting in this case in groups of trading rules which converge towards our optimum target.

The fitness evaluation is achieved by using a *proportional fitness criterion* (roulette wheel) approach. This fitness level is used to associate a probability of selection for each individual rule. If f_i is the fitness of individual i in the population, its probability of being selected is

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j} \tag{3}$$

where N represents the number of individuals in the population. After calculating each individual fitness level, we perform the PSO on the best half of population and receive back the small groups of trading rules which hope to maximise the return. Formula 1 is applied with two constraints: no two combinations of groups are the same, and each group will not contain the same trading rule twice. Once these groups are returned we apply the genetic operators within each group ensuring that a child receives the parent's proven positive characteristics. This clearly increases the chance of obtaining a better child strategy compared to the case when crossover is performed between completely random parents.

B. Field Programmable Gate Array Acceleration

From genetic programming theory we know that fitness evaluation is the most computationally expensive part of the algorithm. In order to improve execution time, a Field Programmable Gate Array (FPGA) is used to provide hardware acceleration: our FPGA is a Xilinx V6-SXT475 device in a Maxeler MAX3A Vectis dataflow engine, running at 100MHz.

Our design involves sending the population of individuals from the CPU (represented as trees) as *two streams*: the first being represented by each trading rule terminal and the second containing each trading rule operator. We keep the order of the trading rules such that the array block address for the trading value in the first stream corresponds to the array block address from the operators value in the second stream. Together with those two streams we currently send a number of transactions on which we want to compute our individuals' fitness. The transactions (each transaction being represented by an array of market data price values on which we evaluate our trading rules) are all sent together to the FPGA as the *third stream* of data. We perform the fitness evaluation of the current population on the FPGA: we then evaluate each of the individual trading rules on all of the transactions and get back a result representing the actual fitness value of the evaluated individual. Once the computation is done, we return to the CPU an array with the fitness values for each trading rule. Then we proceed with the remaining operations on the CPU.

As a result of this approach our accelerated design runs 15.62 times faster than the corresponding software running on an Intel i7-2600 processor at 3.4GHz. It enables us to test our program on a larger data set, thus improving our statistical and profitability evaluations.

III. Tests and Results

A. Robustness Tests

For this approach to be seen as a reliable trading mechanism we need to examine its robustness. This was achieved through aggregate testing, as well as the insertion of noisy signals into both training as well as evaluation data sets.

We split our data from 2003 and 2004 each into three different data-sets (three months split into individual months: January, February, March) to identify any predictable market price patterns as well as to evaluate the consistency of the algorithm's performance under noisy signals. As different market conditions will lead to different trading rules, consecutive periods of three months are sufficient to identify any exploitable price patterns which in practice will only exist for a very short fraction of time within each day.

1) Aggregate Testing: We first evaluate the number of times the best fitting in-sample trading rule from one month (e.g: January) appears in the top 15% of the best performing trading rules in each of the other two months (e.g: February and March) (see tables below, e.g: the January trading rule under the column named Feb/March). We evaluate the 2003 and 2004 data sets on a population of 150 individual strategies, each with a range of iterations (X). These results show that in 2003 the

TABLE I. 2003 AGGREGATE TESTING

X	Feb/March	Jan/March	Jan/Feb
1000	137/118	142/108	136/120
800	127/102	130/103	122/119
600	122/98	117/92	117/103
400	117/92	121/99	115/92
200	103/86	97/93	95/84

TABLE II. 2004 AGGREGATE TESTING

X	Feb/March	Jan/March	Jan/Feb
1000	107/98	122/92	102/110
800	102/92	99/92	104/88
600	99/89	94/97	107/91
400	86/92	78/86	87/84
200	71/79	74/69	67/62

best trading rule from one month appears in the top 15% best trading rules from the other months in over 50% of the cases. So, even though the best trading rule for one time period might not naturally maintain its ranking over the remaining individual time periods, it appears in the top ranked strategies most of the time. This results provides some confidence that the algorithm is able to perform consistently through time and can identify re-occurring predictable trading patterns.

2) Algorithm Behaviour in Presence of Noisy Data: This test consists of taking the best fitting in-sample trading rule and examining its performance on the out-of-sample data, with and without different amounts of spurious noisy prices added (to a subset, a percentage of the data values) to examine sensitivity of our results to the strength of predictability, or signal, in the price data. We take the out-of-sample market data values and measure their mean and standard deviation and then add to the original data different amounts of noise following a *Normal distribution*. We vary the standard deviation and hence the amount of noise keeping the prices' mean fixed and generate new price values, until our selected trading rule stops predicting and starts giving bad results. We performed the test on the best trading rule from an out-of-sample "January-March" period, rule which has a performance of 1.17213 with 1000 iterations. The buy/sell price means for the tests are 1.61065/1.61075 with their original standard deviation (std) being: $6.99e-05/7.41e-05$.

In Table III, the last four columns represent the performance results of the same trading rule applied on the original out-of-sample market data but with a specified (percentage) amount of random noise added. We notice that for a standard deviation value of 0.0001 for the prices modification we get a performance very close to the original, while for the 0.1 standard deviation value we obtain the lowest performance with 0.639034. These results show what we would expect to see: if we add some random noise to the out-of-sample data, in terms of both a percentage of the data affected and noise level, the best GP/PSO trading rule's performance starts to decrease and this continues while increasing the standard deviation. Thus, Table III indicates that our algorithm performs well because its performance only drops off gradually as the signal

TABLE III. 2003 NOISE BEHAVIOUR - PERFORMANCE MEASURE

Random Noise Amount	Noise Std	Noise Std	Noise Std	Noise Std
	0.0001	0.001	0.01	0.1
50%	0.088953	-0.203530	-0.650471	-1.048890
25%	0.454831	0.128904	0.003891	-0.378158
10%	1.119038	1.005998	0.899640	0.639034

to noise ratio decreases initially, but then its performance falls away significantly as the signal is drowned by extreme levels of the added noise. This result implies there is structure in the data that our algorithm can detect and shows how sensitive the results are to variations in the signal to noise ratio.

3) Rejection-Acceptance Testing: We now check whether those poor (rejected) strategies may be reselected within the best set if they are allowed to remain and re-entered within the next iteration's set of strategies. In a sense this exercise mirrors the aggregate testing described above, when we examined if the best strategy would re-occur in later periods, but now we are looking at the *consistency of rejection* within the iterations of the best strategy selection process. We examine the rejection-acceptance for each iteration measured at the end of the X iterations. Keeping the same market data testing split choice, we evaluate our algorithm on a different number of iterations for a different number of tests. Briefly, let's assume we have a population of N individual trading rules to be examined. When each iteration ends we split this population into the best half (accepted for performing GP operations next) and the worst half (rejected for the next iteration).

The 2003 and 2004 results presented in Tables IV and V were obtained using a population of 150 individual strategies for each of the individual months examined once for each of the X number of iterations. Formula 4 provides the required calculation:

$$TotalAccepted_X = \sum_{i=0}^{N-1} i * itAccepted_i \tag{4}$$

where $TotalAccepted_X$ is the total number of individuals accepted in the next iteration after originally being rejected in the previous iteration, after X total iterations; $itAccepted_i$ stands for the number of individuals being accepted after one iteration only, and i represents the iteration number.

TABLE IV. 2003 REJECTION-ACCEPTANCE TESTING

X	Jan%	Feb%	March%	Jan-March%
1000	38,40%	30,51%	40,07%	36,33%
800	50,93%	48,40%	53,32%	50,88%
600	45,11%	44,19%	39,44%	42,91%
400	41,43%	31,47%	40,96%	37,95%
200	28,70%	21,08%	25,18%	21,08%

TABLE V. 2004 REJECTION-ACCEPTANCE TESTING

X	Jan%	Feb%	March%	Jan-March%
1000	57,76%	47,87%	54,73%	53,44 %
800	53,00%	51,09%	52,05%	52,05%
600	40,57%	43,64%	42,08%	42,10%
400	43,26%	34,71%	32,64%	42,31%
200	38,53%	24,77%	26,78%	30,03%

Looking at Table IV, with the January-March 2003 data we re-accept least fit strategies between 21.08% and 50.88% of the time. Naturally with a greater number of iterations the probability of re-selection will rise. There is a trade-off here between ensuring consistent rejection of failed strategies and flexibility to ensure the ability to adaptively respond to new regimes in the data when different strategies may become optimal. Since our worst result arises with 800 iterations and this is a little over 50% we feel this trade off between flexibility and consistency in optimal rule selection is acceptable. Table V with 2004 data shows us that the re-acceptance rate of failed strategies is higher than in the 2003 data. This might suggest that for 2004, the potential market data price patterns may be harder to find, even non-existent (we cannot be sure at any time that a predictable pattern exists, until we have confirmed the presence of the trading rules giving profitable returns). These results reinforce the impression that the algorithm can identify shifting patterns of predictability in the information set, given that in the majority of cases we have found a relatively low level of re-acceptance of failed strategies. Thus, the algorithm's power to consistently reject poor trading rules while retaining flexibility to adapt to new signals in the data.

B. Individual Returns

Before introducing the results of statistical tests below we perform a brief comparison of the best strategy's returns for the original GP and the improved GP/PSO versions.

Table VI shows the daily returns of the best fit trading strategy after a different number of iterations (X) were used in the GP. From this we can see a clear decrease in return level from 2003 and 2008. This might be due to different market conditions belonging to different regimes but more likely this indicates the greater efficiency of the FX market in 2008 with the huge growth in electronic high frequency trading between 2003 and 2008. Table VII shows the corresponding returns for the GP/PSO of our algorithm. We can draw similar conclusions as from Table VI regarding the 2003 and 2008 performance, but more clearly as we notice a considerable increase in profitability due to the use of PSO, allowing our program to perform genetic operations only on the best-fit individuals, thus generally converging to better trading rules. Comparing our results to the *Salmon and Kozhan* [2] paper, that didn't employ PSO and FPGA technology, it's clear how important these tools are in exploiting the information hidden in price data. FPGA's computation power is shown to be important as the degree of profitability increases with the iterations' number used to search for the optimal rules and PSO appears to show clear profitability remaining in 2008 if the correct tools are used, unlike the results in *Salmon and Kozhan* [2].

TABLE VI. 2003-2008 GP INDIVIDUAL RETURNS

X	Jan(20-24) '03	Feb(17-21) '03	March(10-14) '03	March 31 '08
1000	1.142	1.094	1.003	0.991
800	1.032	1.012	0.850	0.786
600	0.903	0.855	0.739	0.704
400	0.845	0.684	0.714	0.593
200	0.799	0.684	0.709	0.549

TABLE VII. 2003-2008 GP/PSO INDIVIDUAL RETURNS

X	Jan(20-24) '03	Feb(17-21) '03	March(10-14) '03	March 31 '08
1000	1.273	1.202	1.185	1.044
800	1.075	1.086	1.032	0.885
600	0.879	0.884	0.923	0.915
400	0.756	0.773	0.724	0.702
200	0.663	0.687	0.688	0.613

C. Statistical Tests

1) Anatolyev-Gerko Tests: The crude return calculations previously reported must be evaluated in a proper statistical framework to examine their true significance. One test that achieves this is the AG test which statistically compares return levels based conditioning information with those that might occur randomly within the correct distribution/inference framework. We apply this test to the "majority rule" strategy ie. 99 independent runs of the algorithm have been performed (each containing 1000 iterations) to select the best in-sample trading rules. Our "majority" rule produces a buy/sell if the majority of the 99 best fit in-sample trading rules produce a buy/sell signal. The resulting rule is then used to trade out-of-sample [2]. We adjust returns to a daily basis, and express their amount in percent, taking into account the k threshold value as well as transaction costs which are reflected in the bid-ask spread. The real-returns needed to perform the AG test can also be found by calculating actual values from the data, while the predicted-returns are calculated according to what our "majority" rule tells our algorithm to do. The test statistic values shown in Table VIII are based on different conditioning information sets, making variability degree comparisons difficult in each data set but all the reported test statistics are statistically significant. This highlights two things: the results for 2003 are all positive while two for 2008 are negative, and the results vary with different levels of inertia parameter. This also tells us that in 2003 there was significant profitability, which decreased as the number of trades decreased as k increased. Secondly, the negative results for 2008 tell us that for values of k=0 and k=4, despite what we have seen in Table VII the use of conditioning information would have given us negative returns compared to purely random decision making as to how to trade. Hence the test statistic is negative, moreover within the correct distribution for comparison, this contrast is also statistically significant.

TABLE VIII. 2003-2008 GP/PSO AG TEST WITH FILTER

k	Jan (20-24) 2003	Feb (17-21) 2003	March (10-14) 2003	March 31, 2008
0	1.73	2.11	0.97	-0.84
2	1.07	1.10	0.28	0.24
4	0.76	0.69	0.54	-1.92

So, despite the apparent predictability seen in Table VII, the GP/PSO algorithm does't deliver greater profitability in 2008 whereas it did in 2003. This is important as it confirms the impression from earlier results that the efficiency of FX markets increased between 2003 and 2008. The apparent advantage of FPGA and PSO have not in fact generated statistically significant conditional profitability in the 2008 data where it was clearly present in 2003 data.

2) T-statistics tests: In order to explore this conclusion further we next performed *t-tests* to examine if the mean of the returns is significantly different from zero. We start by testing the GP's profitability under different number of iterations until converge. Table IX results appear not be significant at 5% level but reconfirms the AG test results which says there was a greater probability of profitability in 2003 data, and consistently less so in 2008. We performed the same test on the GP/PSO program and obtained similar results to the classic GP. We now perform the t-test both on 2003 and 2008 data,

TABLE IX. 2003-2008 GP T-STATISTICS

X	Jan(20-24) '03	Feb(17-21) '03	March(10-14) '03	March 31 '08
1000	1.630	0.810	0.323	0.592
800	1.629	0.809	0.290	0.593
600	1.591	0.784	0.314	0.583
400	1.587	0.780	0.288	0.572
200	1.580	0.766	0.274	0.572

but we keep the number of iterations fixed at 1000 and we vary the "inertia band" parameter k within the range 0-20. In our case, we measure the filter parameter k in *basis points* (bp = unit equal to one hundredth of a percentage point). Table X presents the out-of-sample percentage of cumulative returns generated varying the k parameter for the GP while Table XI presents the same test performed for the GP/PSO approach. The results obtained in Table XI for the 2003 data indicate that the systematic pattern that seems to exist inside of the market data values as noticed in Table IX, maintains its virtual pattern for different values of k. Small values of the inertia band parameter reflect a high number of transactions, which implies a large cumulative transaction cost that more likely exceeds the profits from trading.

TABLE X. 2003-2008 GP T-STATISTICS WITH FILTER

k	Jan(20-24) '03	Feb(17-21) '03	March(10-14) '03	March 31 '08
0	1.630	0.810	0.323	0.592
4	1.523	0.722	0.268	-0.412
8	0.991	0.703	0.013	-0.081
12	0.269	1.083	0.215	-0.106
16	0.018	0.590	0.027	0.790
20	-0.166	-0.608	-0.148	-0.429

TABLE XI. 2003-2008 GP/PSO T-STATISTICS WITH FILTER

k	Jan(20-24)'03	Feb(17-21)'03	March(10-14)'03	March 31 '08
0	1.683	0.832	0.358	0.618
4	1.680	1.003	0.617	-0.139
8	1.035	0.788	0.146	-0.489
12	0.117	0.912	0.018	0.101
16	0.242	0.713	0.109	0.664
20	-0.042	0.184	-0.029	-0.254

The *t-test* results shows us that our evolutionary approach can be adjusted to handle different levels of transactions costs providing what looks like a systematic predictable pattern for the 2003 data set but little or no pattern for the 2008 data set. These results are consistent with those obtained with the AG test for conditional predictability and demonstrate substantial increases in the efficiency of FX markets enabled by high frequency markets. This confirms the utility of the tool we have developed to examine the nature and stability of high frequency trading strategies. We have demonstrated that we have built a tool that can capture predictable structure if it exists (2003) and hence can be used to monitor irregular patterns by regulators-perhaps caused by market manipulation or extreme events, flash crashes, which would cause market interventions to maintain stability within the market.

IV. CONCLUSIONS

We have built an evolutionary hybrid genetic program which uses aspects of swarm intelligence (particle swarm optimisation) in order to seek reliable and profitable trading patterns that can be used to enhance trading strategies. Naturally, evolutionary methods are time consuming and thus, to reduce execution time and produce more advanced trading rules, hardware acceleration techniques are required, allowing us to obtain significant computational speedup.

We have analysed what happens when negative results are produced, due to market unpredictability. Furthermore we evaluated the algorithm for robustness, profitability, reliability and predictability, aspects which are critical for the employment of the resulting trading rules in the real world.

From a *regulator's* point of view, our tool will be beneficial because it helps them identify and analyse market behaviour under different market regimes, as well as identify trading rules which could cause significant market changes (e.g: a sudden and substantial change in the GBP/EUR cross due to non-obvious market conditions). Such results can be obtained at great speed, aiding them in avoiding significant risk taking. The analysis of the generic stability of the best performing trading rules is left to further research. *Financial institutions* could also see great benefit from this tool, particularly in the context of increasing trading profits.

Given the positive results produced by this working prototype, it is clear to see that a hybrid genetic program enhanced with swarm optimisation and hardware acceleration techniques is capable of producing valuable real time feedback in today's high frequency trading environment, to the point that we have removed any potential we are tracking spurious signals. It also provides a valid framework for evaluating market stability when the market is populated by such strategies.

ACKNOWLEDGMENT

This work is supported in part by the United Kingdom Engineering and Physical Sciences Research Council, by the European Union Seventh Framework Programme under grant agreement number 257906, 287804 and 318521, by the Maxeler University Programme, and by Xilinx.

REFERENCES

[1] Christopher J. Nelly and Paul A. Weller, "Technical Analysis in the foreign exchange market", Federal Reserve Bank of St. Louis-Research Division (working paper series), Working Paper 2011-001B, January 2011.

[2] Roman Kozhan and Mark Salmon, "The information content of a limit order book: The case of an fx market", in *Journal of Financial Markets*, 2012, pp. 1-28.

[3] D. Easley, M. O'Hara, "Liquidity and valuation in an uncertain world", in *Journal of Financial Economics*, 2010, pp. 1-11.

[4] Investopedia, *Filter Rule* [Online], Available: http://www.investopedia.com/terms/f/filterrule.asp

[5] Jeroen Eggermont, *Genetic Programming* [Online], Available: http://www.win.tue.nl/ipa/archive/falldays2007/HandoutEggermont.pdf