Optimising Directional Changes trading strategies with different algorithms

Jurgen J. Palsma
School of Computing
University of Kent, Canterbury, U.K.
jjp29@kent.ac.uk

Adesola Adegboye
School of Computing
University of Kent, Medway, U.K.
atna3@kent.ac.uk

Michael Kampouridis
School of Computing
University of Kent, Medway, U.K.
M.Kampouridis@kent.ac.uk

***Abstract*—Directional Changes (DC), a quasi novel approach for sampling market data, allows the extraction of trends in financial time series by converting series from a time based format to an event-driven format. This paradigm has been shown to give some predictability in financial prediction, and has been used to generate profitable trading strategies on the FOREX market. A particular strategy, proposed by [1], is optimised with a genetic algorithm (GA) . We propose to optimise the strategy proposed by [1] using variants of two nature inspired optimisation algorithms, Particle Swarm Optimization (PSO) and Shuffled Frog Leaping Algorithm (SFLA) in order to outperform the existing GA-optimised strategy. After comparing the performance of the three algorithms on 36 different datasets, we see that the PSO and CSFLA proposed in this dissertation present better strategies than the GA in terms of both returns and risk.**

***Index Terms*—Directional Changes, Genetic Algorithm, Particle Swarm Optimisation, Shuffled Frog Leaping Algorithm**

I. Introduction

A new challenge has emerged from the increase in volume and velocity of financial data in the field of computational finance. Although this eruption of data was first a great source of wealth and opportunity, it quickly became a problem due to its magnitude, luring traders into hectic trading strategies, making the market data noisy and chaotic and limiting its predictability. As a result, an approach named Directional Changes (DC) , was discovered to extract trends in financial data by converting time based data series into event-based data series. This approach allows to view the market through events which are remarkable - of a certain magnitude which interests the trader. A particular implementation of a DC-based trading paradigm, proposed by [1], has been shown to generate profitable and risk-averse trading strategies on the foreign exchange market. In fact, their work was shown to outperform traditional technical analysis based trading strategies. The authors achieved their results in part by using a nature-inspired optimisation algorithm, known as a Genetic Algorithm (GA). The promising results provided by the DC literature and the work of [1] has motivated us to replace the existing GA with two different techniques, Particle Swarm Optimisation, and an adaptation of the Shuffled Frog Leaping Algorithm, to optimise the trading strategy proposed by [1]. Our motivation was also driven by research done by [2] showing that GAs have been outperformed by our proposed alternative techniques on different optimisation problems. The objectives of our work is to improve the strategies proposed by [1] by generating higher returns with our proposed optimisers. To demonstrate our findings, we first present background information on DC in section II, and explain in detail the strategy proposed by [1]. We will then present in section III our proposed alternative optimisation algorithms. We will lay out our experimental setup including our data, how we tuned our algorithms, and their final configurations in section IV, present and analyse our results in section V, and finish by concluding on our work in section VI.

II. Background

In this section, we present the background knowledge necessary to understand the setting of our experiments: we first describe Directional Changes, and then how they have been used by [1] to generate trading strategies.

A. Directional Changes

Directional Changes (DC) is a technique which is used to extract trends from data, and particularly from time series. It was originally cited in [3], who used the frequency of Directional Changes to examine volatility in time series. DC trends are captured by sampling significant events that occur in the market or time series data under observation. A Directional Change Event occurs when the price value changes by θ, a threshold of price change defined by the trader which can be seen as the threshold to which the trader considers the change in price to be significant [4]. This event is named a downturn event if the price fell by θ, and named an upturn event the price rises by θ. A DC event triggers a downwards run or an upwards run, which are periods between two different DC events (e.g.: a downwards run occurs between a downturn event and an upturn event). The run is triggered when we reach a DC confirmation point (when a change of θ is observed). After reaching a confirmation point (also called overshoot event), we enter a phase of overshoot (OS), which occurs until we reach our next DC event.

Applying this concept of Directional Changes,[5] describes trends named "total moves" which are periods of time between two directional change events. Across data tested on 13 FOREX currency pairs, it was discovered that "[...] on average, a directional change [...] is followed by an overshoot of the same magnitude [...], making the total move double the size of the directional-change threshold [...]." This empirical observation states that an overshoot is, on average, twice as large than its preceding directional change:

$$\Delta_{OS} \approx 2\Delta_{DC} \tag{1}$$

And that we can establish total moves (periods of time between two directional change events) as:

$$TM \approx \Delta_{DC} + \Delta_{OS} \approx 3\Delta_{DC} \tag{2}$$

These empirical scaling laws bear the promise of being able to estimate, on average, the time between trend reversals in financial time series. These scaling laws could thus potentially be used to predict financial trends and prospect on them with automated trading techniques. As a result, they have been tested and applied through a multitude of trading strategies, such as [1], [6], [7], [8], which have shown through their experiments to generate profitable trading strategies. In the next subsection we describe how directional change based trading strategy was evolved in [1], the initial research which we improve on, in this work.

B. Evolving trading strategies with directional changes

Based on Directional Changes and the empirical scaling laws presented in the previous subsection, [1] propose a multi-threshold strategy, where each threshold would advise one weighed trading action and a decision would be suggested from a voting session between each weighed trading action, with the argument that multiple thresholds capture different event magnitudes and allow for improved predictions.

In order to optimize the different parameters of their multi-threshold strategy, the authors used an evolutionary algorithm, the genetic algorithm (GA). The genetic algorithm mimics natural selection by iteratively generating a population of solutions based on a selection of individuals through generations depending on their fitness, or ability to solve a problem (maximizing a fitness function). In our case, the fitness function would take as inputs a set of discrete parameters for the trading strategy, and would give as output the returns generated by this strategy over a given price curve.

The use of a genetic algorithm to evolve a multi-threshold strategy allowed the authors to yield promising results when experimenting on tick and 10-minute data from 5 currency pairs in the FOREX market in a time period of 10 months from August 2013 to May 2014. According to their experiments, the multi-threshold strategy outperformed traditional benchmarking techniques such as buy and hold and a genetic programming FOREX trading strategy proposed by [9].

These promising results motivated us to apply different optimisation techniques to the multi-threshold strategy; the parameters of the problem being discrete make the search space of our optimisation problem virtually infinite and we can thus assume that there might be multiple optimal combinations of parameters: meaning our problem is potentially non convex.

Genetic algorithms often provide correct results in such problems as they do not assume the shape of the search space (being metaheuristic techniques), and offer a certain simplicity whilst allowing for modularity by adding problem-specific functionalities.

However, since they remain metaheuristic techniques, we cannot confirm that their results are the most optimal, and this motivated us to apply different optimisation techniques, namely a variant of the particle swarm optimization algorithm and a variant of the shuffled frog leaping algorithm which have been shown by [2] to outperform genetic algorithms in some optimization problems. Having presented the scope of our optimisation and our motivation to pursue our research in it, we present in the next section our proposed alternative algorithms in detail and our methodology for implementing them.

III. Methodology

In this section, we will present the two alternative algorithms we have used to optimise the multiple DC threshold strategy developed by [1], presented in the previous section. We will first present our approach to the particle swarm optimisation (PSO) algorithm, in subsection III-A and then propose a variant of the shuffled frog leaping algorithm in subsection III-B.

A. Particle swarm optimization

Our first proposed alternative, particle swarm optimisation (PSO), a nature inspired metaheuristic search algorithm that was introduced by [10]. It shares some similarities with a GA as it optimises and transforms a set of candidate solutions. However, instead of mutating and evolving the individuals as in a GA, the PSO algorithm optimises individuals based on a concept of velocity or search direction, towards which the individuals are oriented in the search space. The individuals move through the search space until their convergence, i.e. when their change in velocity reaches a certain change threshold.

An individual (candidate solution) is called a particle and is composed of a vector of n attributes, similar to a GA individual, which represents the set of parameters for our search problem. A particle searches through the search space with a certain velocity, which represents the search direction of each parameter towards an optimal fitness value.

Each parameter in a particle has its own velocity v_{ij}, which is defined, for the j-th parameter of the i-th particle, at an iteration $t+1$, in equation 3. The computation of a particle's velocity is a weighted sum of three variables, its inertia, its memory, and its neighbourhood. These variables are represented in equation 3 with:

- The inertia influence $v_{ij}(t)$ representing the parameters previous position influence on the search direction, weighed by the inertia weight w_s.
- The memory influence $h_{ij} - x_{ij}(t)$ representing the parameters previous best (historical) position h_{ij} relative to the parameters previous position $x_{ij}(t)$, weighed by the historical weight w_h
- The neighbourhood influence $g_{ij} - x_{ij}(t)$ representing the parameter's neighbour's best position g_{ij} relative to

the parameter's previous position $x_{ij}(t)$, weighed by the neighbourhood weight w_g. The particle's neighbourhood is a subdivision of the swarm which allows to focus either on exploitation or exploration depending on the neighbourhood' size.

$$v_{ij}(t+1) = w_s.v_{ij}(t) + w_h.(h_{ij} - x_{ij}(t)) + w_g.(g_{ij} - x_{ij}(t)) \quad (3)$$

We also propose some enhancements on top of the canonical aspect of the PSO to maximise our algorithms performance. First, we introduce an early stopping criterion to minimise computation costs by minimising fitness evaluations. We also submit our particles to a technique named clamping which prevents particles from exponentially increasing in velocity by defining a maximum velocity a particle can have. Our final enhancement allows us to deal with outlying individuals with excessively low fitness (which violate fitness constraints), by "resetting" their velocity by setting their inertia and memory weight to 0 for one iteration. This allows the particle to ignore its low-fitness inducing memory weight and makes it more prone to explore the parameter space in direction of its neighbours.

At each iteration of the algorithm, until we reach convergence, we move each particle by applying its velocity to its coordinates (parameter values) in the search space, represented by the equation 4.

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij} \quad (4)$$

For the sake of clarity, we present the PSO algorithm in a high-level pseudocode in algorithm 1.

Algorithm 1 Particle Swarm Optimization

```
With: fitness function: $f()$
1: Initialise population: $P \leftarrow$ with $N$ random particles
2: for each particle $x_i$ in $P$ do calculate $f(x_i)$
3: while convergence not reached do
4:     for each particle $x_i$ in $P$ do
5:         for each attribute $x_{ij}$ in $x_i$ do
6:             $x_{ij} = x_{ij} + v_{ij}$
7:             $v_{ij} = w_s.v_{ij} + w_h.(h_{ij} - x_{ij}) + w_g.(g_{ij} - x_{ij})$
8:     for each particle $x_i$ in $P$ do calculate$f(x_i)$
```

B. Continuous shuffled frog leaping algorithm

We propose a second alternative optimizer, the continuous shuffled frog algorithm (CSFLA) based on the shuffled frog leaping algorithm introduced by [11], improved by [12], which we adapt to continuous search spaces.

The individuals in the CSFLA, named frogs, are represented as a vector of n attributes which are in our case the parameters of the multi-threshold trading strategy.

The CSFLA starts by initialising a population of N randomly generated frogs. Then, the algorithm iteratively evolves the population by evolving niches of candidates to minimise fitness evaluations (which is crucial in our case, as our fitness evaluations are costly by nature, being trading simulations over a large dataset). Thus, the algorithm repeats the following process for a total of G_m iterations (generations):

First, we perform an initial fitness evaluation and order the frogs in descending fitness value order. We then divide our population into M memeplexes, which are sub groups of individuals which allow the exploration of local optima (niches). The division into memeplexes is done with the following process: we assign the first individual in our list of sorted frogs to the first memeplex, the second individual to the second memeplex, the M-th individual to the M-th memeplex, the $M+1$-th individual to the first memeplex, and so on, until each frog is assigned to a memeplex.

To evolve our individuals into niches, we divide our memeplex into "submemplexes". The division is done according to a probabilistic distribution which assures that individuals with higher fitness have a higher chance of being selected. We present the probability of selecting the i-th individual x_i into a submemeplex in equation 5:

$$p(x_i) = \frac{2(N+1-i)}{N(N+1)} \quad (5)$$

This division into submemeplexes allows to focus on exploitation of niches and ignoring the lower scoring individuals to minimise fitness function calculations and encourage convergence towards an optimum in the memeplex. To reduce computational costs, we evolve, in each submemplex, only the individual with the lowest fitness x_w. The lowest individual evolves according to the position in the search space of best performing frog x_b in the submemeplex, with the following 3-step procedure:

- First, we attempt a learning procedure where x_w learns from x_b with equation 6, with r a random number between 0 and 1.

  $$x_w(t+1) = x_w(t) + r(x_b(t) - x_w(t)) \quad (6)$$

  Considering our fitness function f and if $f(x_w(t+1)) > f(x_w(t))$ then we assume the worst frog has positively learned and we assign $x_w(t)$ to its new value $x_w(t+1)$.
- If $f(x_w(t+1)) < f(x_w(t))$, the worst frog has not improved in terms of fitness, and thus we try to learn from the best individual x_s in the entire swarm with equation 7:

  $$x_w(t+1) = x_w(t) + r(x_s(t) - x_w(t)) \quad (7)$$

  As in the previous step, considering the fitness function f and if $f(x_w(t+1)) > f(x_w(t))$ then we assume the worst frog has positively learned and we assign $x_w(t)$ to its new value $x_w(t+1)$.
- Otherwise, we assign $x_w(t+1)$ to a randomly generated position, with a and b some arbitrarily set maximum and minimum boundaries, in equation 8.

  $$x_w(t+1) = a + r(b-a) \quad (8)$$

We repeat this evolution process in our sub-memeplex for a certain number G_s of sub-generations, and then return our individuals to our memeplex, and repeat our process for a certain number of G_m generations. Once the generations have past, the algorithm yields a list of candidates sorted by fitness. In our case, we pick the highest performing individual as our candidate solution for the optimisation problem. A high-level pseudocode representation of the CSFLA is presented in algorithm 2.

Algorithm 2 Continuous Shuffled Frog Leaping Algorithm

```
With: fitness function: $f()$, $g_m = 0$, $g_s = 0$
 1: Initialise population: $P \leftarrow$ with $N$ random particles
 2: for each frog $x_i$ in $P$ do calculate $f(x_i)$
 3: Sort $P$ by decreasing fitness value
 4: while $g_m < G_m$ do
 5:     Generate $M$ memeplexes
 6:     for each memeplex do
 7:         Generate $n$ sub-memeplexes
 8:         for each attribute $x_{ij}$ in $x_i$ do
 9:             while $g_s < G_s$ do
10:                 Move the worst frog of the memeplex
11:                 According to equations $6, 7, 8$
12:                 $g_s = g_s + 1$
13:             Insert all frogs back in $P$
14:     for each particle $x_i$ in $P$ do calculate$f(x_i)$
15:     $g_m = g_m + 1$
```

Having presented our two alternative algorithms, we can now lay out, in the following section, our experimental setup.

IV. Experimental setup

In this section, we present how we set up our experiment: we describe the data we used, how we tuned our algorithms for our optimisation problem, and lay out the different algorithm configurations we used for testing.

A. Data

The data used in the experiment consists of 12 months of 10-minute interval data from the FOREX market, from June 2013 to May 2014, from 4 different currency pairs: Euro / US Dollar , Euro / British pound, British pound / Swiss franc, and British Pound / US dollar. The first 3 months of data were used to tune the algorithms, and the remaining months were used to evaluate the results of our algorithms on the optimisation problem.

B. Algorithm tuning and parameters

We applied the same tuning methodology to both the PSO and the CSFLA: we first conducted primary analysis on each hyper parameter of the algorithm, by analysing the independent effect of each parameter on performance on our training data. Once we have extracted sufficient knowledge from the independent tuning of each parameter, we combined different configurations which we thought would fit well together. For example, we could find that some parameters could be complimentary with one another, or that some parameters could influence performance on their own (for example, the size of the swarm in the case of the PSO), independently of the other parameters. We tested over 50 different PSO combinations and over 40 CSFLA combinations, and chose the configurations which gave the highest returns on the test data.

The final PSO configuration is presented in table I, the final CSFLA configuration in table II, and the GA parameters, set as equal to the ones set by [1] in their experiments, are presented in the table III.

TABLE I
PSO CONFIGURATION

Parameter	Value
Swarm size	50
Maximum velocity	150
Inertia weight	0.85
Memory weight	0.45
Neighbourhood weight	0.05
K	5
Minimum velocity threshold	0.00005
Maximum iterations	7

TABLE II
CSFLA CONFIGURATION

Parameter	Value
Number of frogs	200
Frogs per memeplex	20
Number of memeplexes	40
Maximum generations	3
Maximum sub-memeplex generations	40

TABLE III
GA CONFIGURATION

Parameter	Value
Population size	1000
Number of generations	35
Tournament size	4
X-over probability	0.90
Mutation probability	0.0025

V. Results and analysis

In this section, we present the results of our experiments running the algorithms presented in section III with the setup presented in section IV.

By looking at the overall accumulated returns across all the four currency pairs and 9 months of testing data in table IV, we can see that the GA looses money overall, whilst both the PSO and CSFLA have positive returns. We can also see that the PSO yields nearly twice the gains of the CSFLA. From this perspective, we can argue that the PSO is the best suited optimizer for this problem from the 3 compared algorithms.

If we look at the accumulated returns on each currency pair separately in table IV, we can see that although the GA is profitable on one currency pair for which it outperforms the

PSO and CSFLA, the CSFLA is profitable on two out of the four currency pairs, and the PSO yield positive results for three currency pairs and reaches near-null accumulated returns on the last currency pair. These results can be used to argue that the PSO is on average a better performer than the GA or CSFLA. In our specific optimisation problem, on the given datasets, it yields a much safer strategy as it minimizes losses compared to the two other optimisers.

TABLE IV
CUMULATED RETURNS OF EACH ALGORITHM ACROSS THE FOUR CURRENCY PAIRS ON 9 MONTHS OF DATA

Algorithm	PSO	CSFLA	GA
EUR/GBP	-0.00014	-0.00197	-0.00459
GBP/CHF	0.000896	0.000978	0.00317
GBP/USD	0.008697	0.007279	-0.000229
EUR/USD	0.000928	-0.000984	-0.021428
Total	0.010381	0.005303	-0.023077

To judge the consistency of the results of the three different algorithms, we analysed the average variance in returns for each month across all our testing data. We can see, in table V the PSO and CSFLA both have significantly low average variance in returns when compared to the GA. This shows that the PSO and CSFLA yield within our optimisation problem's constraints more homogeneous results, which is particularly advantageous for the context of our problem as it allows for a higher robustness in results.

TABLE V
AVERAGE VARIANCE IN RETURNS FOR EACH ALGORITHM

Algorithm	Average variance in returns per month
PSO	0.000003
CSFLA	0.000002
GA	0.000018

As a result, we argue that given the constraints of our optimisation problem, which is to optimise DC-based trading strategies, our proposed alternatives are more approriate algorithms. Indeed, they allow to increase the profitability of the DC-based strategies by yielding overall higher cumulated returns across 36 different 10-minute FOREX datasets (9 months from 4 currency pairs). Our alternatives also are preferred in terms of consistency, as they perform better in terms of variability.

More precisely, we argue that the best optimiser for this problem is the PSO algorithm, since, in comparison to the CSFLA and GA, it yields better average results , has a low variability of results, and is highly profitable on three out of the 4 currency pairs. This makes it the best choice not only in terms of gross returns for our specific constraints, but also potentially a more sustainable alternative across different problems, due to its low variability in perfomance.

VI. CONCLUSION

In conclusion, we have presented a new way to model financial data, Directional Changes (DC) , and trading strategies derived from it. We have described how [1] used a Genetic Algorithm to optimise DC-based trading strategies, and laid out our hypotheses which were that using different techniques to optimise these strategies could potentially yield better results. From this hypothesis, we have derived an experiment with the following objectives: to apply two different algorithms, Particle Swarm Optimization (PSO) and Continuous Shuffled Frog Leaping Algorithm (CSFLA), to the optimisation problem, and to outperform the GA on this task. The results of our experiments are threefold: first, they reinforce the potential of Directional Changes by generating profitable DC-based strategies. Then, they show that using different optimisation techniques, namely the PSO algorithm, as a DC-based strategy optimiser results in better performing strategies in terms of gross results, and finally, they show that our proposed alternatives allow to generate more sustainable strategies by reducing variability in their returns. Therefore, we encourage further research on the subject, by applying the PSO-optimised multi-threshold strategy on different datasets. We should also take into account that our results could improve even more if we allocated more resources and more data to our algorithms, or if we tried to optimise more complex trading strategies, which took into account more thresholds for example.

REFERENCES

[1] M. Kampouridis and F. E. Otero, "Evolving trading strategies using directional changes," *Expert Systems with Applications*, vol. 73, pp. 145–160, May 2017. DOI: 10.1016/j.eswa.2016.12.032.

[2] E. Elbeltagi, T. Hegazy, and D. Grierson, "Comparison among five evolutionary-based optimization algorithms," *Advanced Engineering Informatics*, vol. 19, no. 1, pp. 43–53, 2005, ISSN: 1474-0346. DOI: https://doi.org/10.1016/j.aei.2005.01.004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1474034605000091.

[3] D. M. Guillaume, M. M. Dacorogna, R. R. Dave, U. A. Muller, R. B. Olsen, and O. V. Pictet, "From the bird's eye to the microscope: A survey of new stylized facts of the intra-daily foreign exchange markets," *Finance and Stochastics*, vol. 1, no. 2, pp. 95–129, Apr. 1997, ISSN: 0949-2984. DOI: 10.1007/s007800050018. [Online]. Available: https://doi.org/10.1007/s007800050018.

[4] E. P. K. Tsang, "Directional changes: A new way to look at price dynamics," in *Computational Intelligence, Communications, and Business Analytics*, J. K. Mandal, P. Dutta, and S. Mukhopadhyay, Eds., Singapore: Springer Singapore, 2017, pp. 45–55, ISBN: 978-981-10-6427-2.

[5] J. B. Glattfelder, A. Dupuis, and R. B. Olsen, "Patterns in high-frequency fx data: Discovery of 12 empirical scaling laws," *Quantitative Finance*, vol. 11, no. 4, pp. 599–614, 2011. DOI: 10.1080/14697688.2010.481632. eprint: https://doi.org/10.1080/14697688.2010.481632. [Online]. Available: https://doi.org/10.1080/14697688.2010.481632.

[6] A. Bakhach, E. Tsang, W. L. Ng, and V. L. R. Chinthalapati, "Backlash agent: A trading strategy based on directional change," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, Dec. 2016, pp. 1–9. DOI: 10.1109/SSCI.2016.7850004.

[7] A. Golub, J. Glattfelder, and R. B. Olsen, "The alpha engine: Designing an automated trading algorithm," *SSRN Electronic Journal*, May 2017. DOI: 10.2139/ssrn.2951348.

[8] J. Gypteau, F. E. B. Otero, and M. Kampouridis, "Generating directional change based trading strategies with genetic programming," in *Applications of Evolutionary Computation*, A. M. Mora and G. Squillero, Eds., Cham: Springer International Publishing, 2015, pp. 267–278, ISBN: 978-3-319-16549-3.

[9] M. Kampouridis and E. Tsang, "Eddie for investment opportunities forecasting: Extending the search space of the gp," in *IEEE Congress on Evolutionary Computation*, Jul. 2010, pp. 1–8. DOI: 10.1109/CEC.2010.5586094.

[10] G. Venter and J. Sobieszczanski-Sobieski, "Particle swarm optimization," *AIAA Journal*, vol. 41, no. 8, pp. 1583–1589, Aug. 2003, ISSN: 0001-1452. DOI: 10.2514/2.2111. [Online]. Available: https://doi.org/10.2514/2.2111.

[11] M. M. Eusuff and K. E. Lansey, "Optimization of water distribution network design using the shuffled frog leaping algorithm," *Journal of Water Resources Planning and Management*, vol. 129, no. 3, pp. 210–225, 2003. DOI: 10.1061/(ASCE)0733-9496(2003)129:3(210). eprint: https://ascelibrary.org/doi/pdf/10.1061/%28ASCE%290733-9496%282003%29129%3A3%28210%29. [Online]. Available: https://ascelibrary.org/doi/abs/10.1061/%28ASCE%290733-9496%282003%29129%3A3%28210%29.

[12] Z. Zhen, D. Wang, and Y. Liu, "Improved shuffled frog leaping algorithm for continuous optimization problem," in *2009 IEEE Congress on Evolutionary Computation*, May 2009, pp. 2992–2995. DOI: 10.1109/CEC.2009.4983320.