2014 International Joint Conference on Neural Networks (IJCNN)
July 6-11, 2014, Beijing, China

Combining Technical Trading Rules Using Parallel Particle Swarm Optimization based on Hadoop

Fei Wang, Philip L.H. Yu and David W. Cheung

Abstract— **Technical trading rules have been utilized in the stock markets to make profit for more than a century. However, no single trading rule can ever be expected to predict the stock price trend accurately. In fact, many investors and fund managers make trading decisions by combining a bunch of technical indicators. In this paper, we consider the complex stock trading strategy, called Performance-based Reward Strategy (PRS), proposed by [1]. Instead of combining two classes of technical trading rules, we expand the scope to combine the seven most popular classes of trading rules in financial markets, resulting in a total of 1059 component trading rules. Each component rule is assigned a starting weight and a reward/penalty mechanism based on rules' recent profit is proposed to update their weights over time. To determine the best parameter values of PRS, we employ an improved time variant particle swarm optimization (TVPSO) algorithm with the objective of maximizing the annual net profit generated by PRS. Due to a large number of component rules and swarm size, the optimization time is significant. A parallel PSO based on Hadoop, an open source parallel programming model of MapReduce, is employed to optimize PRS more efficiently. The experimental results show that PRS outperforms all of the component rules in the testing period.**

I. Introduction

TECHNICAL trading rules are widely used in the financial markets for more than a century as technical analysis tools for security trading. Typically, they predict the future price trend by analyzing historical price movement and initiate buy/sell signals accordingly. Many empirical studies (e.g. [2]-[8]) provided supporting evidence to the significant profitability of various trading rules.

As commented by [9], no single trading rule can ever be expected to predict the stock price trend accurately. In fact, many trading decisions are made by combining a bunch of technical indicators [8]. Recently, [1] proposed a complex stock trading strategy, called Performance-based Reward Strategy (PRS), which combines the two popular classes of technical trading rules - Moving Average (MA) and Trading Range Breakout (TRB). Their experiments showed that PRS outperforms all of the component rules for trading the constituent stocks of NASDAQ100.

In this paper, we expand the scope of PRS to consider combining the seven most popular classes of technical trading rules in financial markets: Moving Average (MA), Trading Range Breakout (TRB), Bollinger Bands (BBs), Relative Strength Index (RSI), Stochastic Oscillator (STO), Moving Average Convergence/Divergence (MACD) and On-Balance Volume Average (OBVA). For each class of trading rules, PRS includes various combinations of the rule parameters, resulting in a universe of 1059 component trading rules in all. All the parameters are chosen in order to represent a wide coverage of the parameters for each rule class ([2], [4]). Each component rule is assigned a starting weight which indicates its significance in trading decision. And a reward/penalty mechanism based on component rules' recent performance is proposed to update their weights over time. The trading signal of PRS is determined by the weighted sum of component rules' signals and two additional signal threshold parameters.

Together with component rules' starting weights and other five parameters of PRS (to be discussed later), there are altogether 1064 parameters for PRS. The optimal PRS can be determined by searching for the set of optimal parameter values with the goal of making profit as high as possible. Note that the traditional gradient-based optimization techniques such as Newton's method cannot be employed here as the profit function is non-differentiable. One possible solution is to use stochastic optimization algorithms such as genetic algorithm (GA) and particle swarm optimization (PSO). In our recent work [1], we employed an improved time variant particle swarm optimization (TVPSO) algorithm [10] with the objective of maximizing the annual net profit generated by PRS. Due to a much larger number of component rules, the optimization time used in TVPSO may become very long, say several days. To address this problem, we propose an efficient parallel PSO based on Hadoop (an open source implementation of MapReduce, which is a programming model proposed by Google for parallel computation [11]). With the underlying advanced mechanism for inter-machine communication, processors load balancing and fault tolerance of Hadoop, the proposed parallel PSO speeds up the PRS optimization significantly. See [12]-[15] for more recent development of PSO.

The rest of this paper is organized as follows: Section II gives details of the proposed trading strategy PRS. Section III briefly introduces PSO and describes how PRS is optimized with PSO. Parallel PSO based on Hadoop is then given in Section IV. Empirical results are presented and discussed in Section V. Section VI concludes the paper.

II. Performance-based Reward Strategy

Table I shows seven classes of the simplest and most popular stock trading rules in the literature. Given such a complex and dynamic market, it is hopeless to have a single

Fei Wang and David W. Cheung are with the Department of Computer Science, The University of Hong Kong, Pokfulam, Hong Kong (email: {fwang, dcheung}@cs.hku.hk) and Philip L.H. Yu is with the Department of Statistics and Actuarial Science, The University of Hong Kong, Pokfulam Road, Hong Kong (email: plhyu@hku.hk)

TABLE I

COMPONENT RULES OF PRS

Simple trading rules	Numbers[a]
Moving Average (MA)	130
Trading Range Breakout (TRB)	20
Bollinger Bands (BBs)	220
Relative Strength Index (RSI)	90
Stochastic Oscillator (STO)	324
Moving Average Convergence/Divergence (MACD)	140
On-Balance Volume Average (OBVA)	135
All rules	1059

[a] The total number of trading rules in that class by taking different values of parameters.

trading rule which performs the best all of the time. We thus consider a wide range of combinations of parameter values for each class of trading rules so as to generate a universe of 1059 component trading rules in PRS.

A. Signal Generation of PRS

In PRS, each component rule r_i is assigned a starting weight w_i, which measures the influence of r_i to the signal generated by PRS. Consider a trading day t, each component rule initiates a signal $s_{i,t}$ which takes value 1, 0 and -1 if the signal is 'buy', 'null' and 'sell', respectively. The signal of PRS on trading day t is given by:

$$s_t = \sum_{i=1}^{1059} w_i s_{i,t} \qquad (1)$$

where the sum of the all weights ($\sum w_i$) should be 1 so that s_t is between -1 and 1.

Note that s_t summarizes all component rules' prediction on stock trend. If s_t is close to 1, this means most of influential component rules suggest buy signals. On the contrary, s_t nearing -1 means more influential rules suggest sell signals. So we propose that PRS initiates a buy (sell) signal if s_t is greater (smaller) than a positive (negative) threshold, bth (sth); otherwise PRS does not initiate any signal and investors do nothing on that day. Higher threshold represents the strategy is more strict in buy or sell and lower threshold represents a more tolerant strategy.

B. Reward and Penalty of Component Rules

Each component rule r_i of PRS is associated with a weight w_i. It represents the influence of r_i to the trading decision making. However, these rules' performance may change during trading, especially over a long time period. It is reasonable to reward a profitable component rule by adding more weight to it and to penalize a non-profitable component rule by deducting some weight from it, and these weights should be updated regularly. As a result, two time spans$-$memory span ms and review span rs, which are introduced in the learning strategy (LS) in [7], are used here. Memory span is a historical period used for evaluating the rule performance. Review span is the time interval over which the weights of component rules should be updated. We set $ms \geq rs$ as suggested by [7].

Suppose on trading day t, we evaluate all rule's performance and update their weights accordingly. Let $profit_i$ denotes the profit of rule r_i from day $t - ms$ to day $t - 1$. For those non-profitable rules, we deduct their weights by a constant:

$$w_i = w_i - \frac{rf}{N}, \quad \text{if } profit_i < 0 \qquad (2)$$

where N is the total number of component rules and rf is a parameter called reward factor, controlling the amount of reward. It is noted that w_i should not be negative, so w_i is set to zero if it is below the weight threshold $\frac{rf}{N}$.

All weights deducted from the non-profitable rules are summed to form a temporary variable W. Then we increase the weight of those profitable rules by:

$$w_i = w_i + \frac{W}{N_+}, \quad \text{if } profit_i > 0 \qquad (3)$$

where N_+ is the number of profitable rules found in the memory span.

Note that the sum of weights remains unchanged after the penalty and reward. However, if most of the rules are non-profitable and only a few rules are profitable, the above reward/penalty approach may add too much weight to those few profitable rules. Imagine that there are 100 rules in which only one rule is profitable in memory span, the weight increment of the only profitable rule is 99 times of the weight decrement of any other rule. The reward may be too much, especially when the rule is just profitable in a short period of time. To avoid a huge reward, we replace Equation (2) with:

$$w_i = w_i - \left(\frac{rf}{N}\right)\left(\frac{N_+}{N}\right), \quad \text{if } profit_i < 0 \qquad (4)$$

where term $\frac{N_+}{N}$ guarantees that the penalty weight of any non-profitable rule and the reward weight of any profitable rule is capped at $\frac{rf}{N}$. It is noted that when all rules are profitable or all of them are non-profitable, our reward/penalty mechanism would not be triggered as in the case there is no need to reward or penalize any rule.

III. PARTICLE SWARM OPTIMIZATION FOR PRS

For PRS, there are 1059 starting weights (w_1 to w_{1059}), two time spans (ms, rs), two thresholds (bth, sth), and a reward factor (rf) to be determined, thus the search space dimension for this optimization problem is 1064.

In the literature, Particle Swarm Optimization (PSO) is one of the most popular optimization algorithms used to optimize trading rules. It is a flexible, stochastic search algorithm based on swarm intelligence, which was first introduced by [16]. Since its inception, PSO has shown great success in solving function optimization problems and has been widely applied in a variety of engineering applications [17], [18].

PSO is motivated by the behavior of bird flocks in finding food. PSO uses a swarm of particles to simulate these birds. Each particle (bird) has a *random* initial position X and velocity V, attempting to move towards the food (the solution of the optimization). The position X of an particle is an

1064-dimensional vector which represents a possible PRS solution to the 1064-dimensional maximization problem, where the objective (or fitness) function to be maximized in this paper is the annual net profit (ANP) generated from the PRS.

PSO searches the maximum fitness value by iteration. In each iteration, PSO records the best position, $pbest$, for each particle that has achieved so far, and the global best position, $gbest$, that all the particles have achieved so far. At the end of the kth iteration, PSO updates each particle's velocity using the following equation:

$$V_{k+1} = \lambda V_k + c_1 r_1 (pbest - X_k) + c_2 r_2 (gbest - X_k) \quad (5)$$

where λ is the inertia weight for the particle wondering in the search space, c_1 and c_2 are the acceleration coefficients, and r_1, r_2 are two random numbers in the range between 0 and 1 [19]. Note that the second term represents the self-cognition of the past experience of a particle so that the particle tends to move towards its past best position. Similarly, the third term indicates that all particles have the social cognition to the whole swarm and are attracted by the global best position.

After updating the velocity, each particle will move to a new position according to:

$$X_{k+1} = X_k + V_{k+1}. \quad (6)$$

The particle movement will repeat iteratively until all particles converge to the optimal position or a termination criterion is met.

Notice that in PSO, the trade-off between global exploration and local exploitation of particles is the key important factor to PSO's performance [19]. Therefore, we considered a refined time variant PSO based on the work of [20] and [10], which linearly updates λ, c_1 and c_2 in Equation (5) over the iterations in order to identify better searching results. More details can be found in these two papers and [21].

IV. Parallel PSO of PRS Based on Hadoop

As PRS requires evaluating the performance of many component rules at regular intervals ($review\ span$), the computation of fitness function could thus be very time consuming. For a large swarm size, each iteration of PSO might take half an hour and hence the optimization time for just one trial could take several days. To resolve this problem, an efficient parallel PSO based on Hadoop is proposed here.

Note that in each iteration of PSO, all particles update their velocities using Equation (5) and move to new positions by Equation (6). After that each particle evaluates the new position's fitness and updates its personal best $pbest$ if possible. Since each particle updates itself independently of each other, this step can then be parallelized. The only common information shared by the swarm is the global best $gbest$. For any particle which moves to a better position that has higher fitness value than the current $gbest$, its new position may be the new $gbest$. The new $gbest$ is calculated from those new position with highest fitness value than the current $gbest$, and the updated $gbest$ needs to be sent to all particles for next iteration's particle updating.

Hadoop is an open source project for reliable, scalable and portable distributed computing. It implements the MapReduce for parallel computing. Since MapReduce is not designed for iterative programs, a new MapReduce job thus needs to be initialized for each PSO iteration's calculation. In each iteration, $gbest$ is first calculated from those possible candidates, then a separate map function invocation is used to update each particle. All possible new global bests are stored on the distributed file systems for next generation's calculation.

A. Representation of Particle and Gbest

A particle is represented as a key/value string pair. The key is a numerical particle ID, and the value consists of particle's position (pos), velocity (vel) and fitness value, $pbest$'s position and $pbest$'s fitness value. A particle is of the form:

$id : fitness;\ pos;\ vel;\ pbest.fitness;\ pbest.pos$

The $gbest$ has the following string form:

$gbest.fitness;\ gbest.pos$

B. Grouping Particles in Hadoop

Typically, one Hadoop job is distributed into a number of map tasks and reduce tasks which can be handled concurrently. In a Hadoop cluster, one machine is assigned a role as JobTracker, and all other machines are assigned as TaskTracker roles. A user job is submitted to the JobTracker and each TaskTracker is assigned one or several tasks distributed from the job. It is common that a single machine has multiprocessors, thus each TaskTracker can handle several tasks with multi-processors concurrently in just one machine. In Hadoop, a map task or reduce task is handled by one Mapper or Reducer, respectively. Each Mapper or Reducer can invoke map or reduce function many times.

At the beginning of parallel PSO, a large number of particles are randomly generated. Suppose there are m machines in the computing cluster, and each machine has p available processors. The initial particles are randomly but evenly divided into $(m-1) \times p$ groups. All groups can be handled concurrently because there are $m-1$ TaskTrackers (machines in cluster) and each TaskTracker can run p Mappers in parallel. Particles belonging to the same group are stored together in a single file, in which each particle is a line of string. Assume there are g particles in a group. An example of a particle group is like this:

$id_1 : fitness_1;\ pos_1;\ vel_1;\ pbest_1.fitness;\ pbest_1.pos$
$id_2 : fitness_2;\ pos_2;\ vel_2;\ pbest_2.fitness;\ pbest_2.pos$
......
$id_g : fitness_g;\ pos_g;\ vel_g;\ pbest_g.fitness;\ pbest_g.pos$

Note that the initial position of a particle is also its initial $pbest$ position. These personal bests are candidates of the global best. They are stored together in a single file in which each $pbest$ is a line. Suppose the swarm size is s, this file is of the following form:

$gbest_1.fitness;\ gbest_1.pos$
$gbest_2.fitness;\ gbest_2.pos$
......
$gbest_s.fitness;\ gbest_s.pos$

C. Mapper for PSO

A Mapper takes a particle group as input. Before it invokes the map function to update particles, the Mapper reads all $gbest$ candidates from the distributed file systems and finds the largest one as $gbest$ of current iteration. In addition to $gbest$ calculation, a Mapper needs to load application data, adjust the coefficients of PSO and some other application related preprocessing tasks. Then each Mapper reads particles from a particle group and transfers them into individual key/value pairs. For each key/value pair, the Mapper call the map function to update particle's velocity and position using Equations (5) and (6), respectively. For each particle in the group, the fitness of its new position is evaluated and its $pbest$ is updated if the new position has a higher fitness than $pbest$. After all particles in this group are updated, they replace the old particle group and are stored as the next iteration's input.

In a particle group, if some of the particles move to better positions with higher fitness than the current $gbest$, the best one from these new positions is regarded as a $gbest$ candidate (c_gbest) for the whole swarm. Therefore, each particle group will generate zero or just one $gbest$ candidate and they are stored as the next iteration's $gbest$ candidates. The only common information among particles is the $gbest$ and it is calculated by all Mappers at the beginning of each PSO iteration. Therefore no Reducer is needed to collect the updated swarm information produced by Mappers.

Note that in a typical parallel PSO based on MapReduce (MRPSO), the reduce phase is in charge of the particles collection and the $gbest$ updating. In this study, the Mapper is quite different from Mappers in MRPSO. In fact, the Mapper's first task $t1$ in nature is doing the $gbest$ updating and task $t3$ is the particles collection. Be aware that finding the $gbest$ from candidates many times concurrently takes the same time as doing it just once, therefore $t1$ does not waste time although it is processed by all Mappers repeatedly. In addition, the updated particles and $gbest$ candidates are directly accessed as input and output of Mappers, thus no intermediate data is generated. It significantly reduces the I/O cost of intermediate data and communication between Mappers and Reducers, especially when the data size is huge. Since there is no reduce phase in our proposed parallel PSO, we can call it Map-PSO (MPSO).

Figure 1 shows the overview of a single iteration of the proposed parallel PSO based on Hadoop which consists of the following execution steps:

(1) Preprocessing: at the beginning of a PSO iteration, $n = (m-1)p$ Mappers are initiated and each Mapper reads one particle group as input.
(2) Task $t1$: Mappers read $gbest$ candidates generated from the previous iteration and calculate $gbest$. At this stage, all Mappers do the same work.
(3) Task $t2$: Mappers begin to update particles (update velocity and position, evaluate new position's fitness) in parallel.
(4) Task $t3$: Updated particles are written into distributed file systems as the next iteration's input.

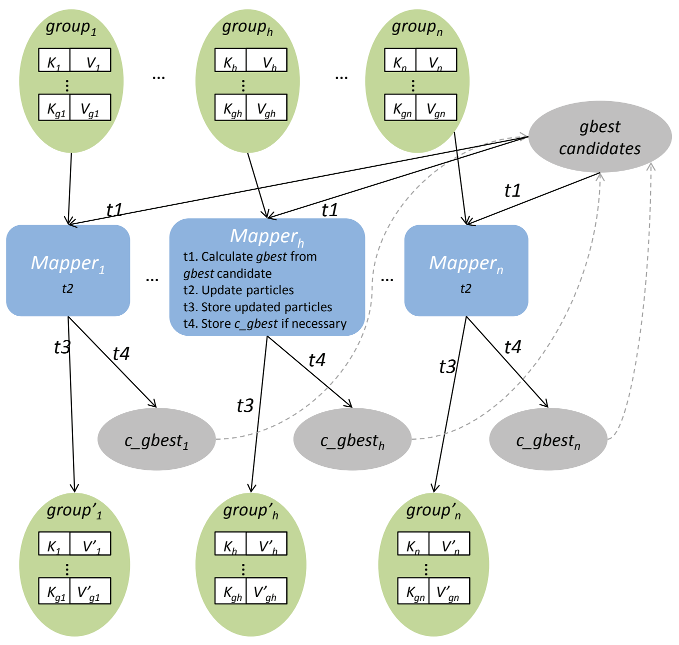

Fig. 1. Overview of a single iteration of parallel PSO based on Hadoop.

(5) Task $t4$: If any particle group generates c_gbest, it is written into distributed file systems as the next iteration's input.

V. Application for NASDAQ100

A. Data

The constituent stocks of NASDAQ100, which are the 100 largest domestic and international non-financial stocks of the NASDAQ stock market, are considered in our study. The daily stock prices (high, low and close prices) and volume data from 1994 to 2010 are collected from Reuters 3000Xtra. Because not all the stocks were issued before 1994, 52 stocks having data throughout the whole period are considered in our experiments. The data from 1995 to 2002 is used for training the PRS and the data from 2003 to 2010 is used for testing the profitability of the trained PRS. It is noted that for some component rules such as Moving Average (MA) with nl = 250, it needs data over the past 250 days to calculate current day's long-period moving average. Therefore, the data in 1994 and 2002 are reserved for data preparation in training and testing, respectively. It is assumed that all buy and sell trades are occurred at the close prices.

B. Optimization Set-up

As the high dimension and complexity of PRS, the swarm size is set to 420 and the number of iterations is set to 300. An 8 nodes cluster is built for the parallelization of PSO. A stable Hadoop version 0.20.2 is adopted as the MapReduce system in this study. With 42 available working processors in cluster, the parallel PSO can achieve a speedup of 23 (see the speedup test later). Table II gives the search space boundary of PRS optimization. In order to avoid the data snooping bias to any component rule in the training period, the range of

TABLE II

SEARCH SPACE BOUNDARY OF PRS PARAMETERS

PRS Parameters	Boundary
α_i $(i = 1..1059)^*$	$[-1, 1]$
ms	$[150, 300]$ trading days
rs	$[20, 150]$ trading days
rf	$[0, 1]$
bth	$[0, 0.3]$
sth	$[-0.3, 0]$

$^*w_i = \frac{e^{\alpha_i}}{\Sigma e^{\alpha_i}}$.

TABLE III

PERFORMANCE OF PRS AND THE SEVEN BEST COMPONENT RULES IN THE TESTING PERIOD

Trading rules (parameters)	ANP	Sharpe	Payoff	NT	Win%
PRS	21.8%	0.94	4.40	335	52.8%
MA (nl=150, ns=125)	18.8%	1.07	3.53	531	54.4%
TRB (n=125)	16.0%	1.05	5.84	276	52.2%
BBs (n=30, k=2.3)	18.6%	1.14	4.62	627	49.8%
RSI (n=13, ob=80, os=30)	9.0%	0.60	0.81	801	70.9%
STO (n=10, m=3, ob=90, os=20)	11.6%	0.76	0.74	1401	71.7%
MACD (nl=100, ns=40, m=15)	11.4%	0.82	2.75	1674	38.5%
OBVA (nl=75, ns=50)	10.7%	0.72	1.49	2157	53.7%

α is set as $[-1, 1]$ so that the range of w is approximately $[0.0001, 0.007]$. In this paper, short selling of stocks is not allowed so that the equity is always positive, and for each buy or sell trade, a transaction cost of 0.1% is considered.

C. Trading Performance

After training, the optimized PRS produced the ANP of 32.9% in the training period. It has no doubt that it outperformed all the component trading rules in the training period (the best component in training is MA(250, 200) which produced an ANP of 25.1%). To have a fair assessment of PRS, we compare the optimized PRS with the best seven component trading rules having the highest annual net profit (ANP) in the testing period. In addition to ANP, two more performance measurements – Sharpe ratio (Sharpe) and payoff ratio (Payoff) – are considered in our comparison. Sharpe ratio was proposed by [22] and measures the return per unit of risk. Payoff ratio is simply the average profit of winning trades divided by average loss of losing trades. It is widely used by traders to compare the expected return to the amount of capital at risk. The higher the Sharpe ratio and payoff ratio, the better the trading rule. The results are shown in Table III. The number of trades (NT) and the percentage of profitable trades (Win%) for each trading rule in the testing period are also given.

In Table III, the seven trading rules are the best Moving Average, Trading Range Breakout, Bollinger Bands, Relative Strength Index, Stochastic Oscillator, Moving Average Convergence/Divergence and On-Balance Volume Average rules having the largest ANP in the testing period, respectively.

TABLE IV

STOCK PROFIT SUMMARY OF PRS AND THE SEVEN BEST COMPONENT RULES IN THE TESTING PERIOD

Trading rules	Summary	Profitable	Non-profitable	All stocks
(Parameters)		stocks[a]	stocks[a]	(Profit ratio)[b]
PRS	No.Stocks[c]	44	8	52(84.6%)
	Net Profit	20.22	-0.27	19.95
MA	No.Stocks	34	18	52(65.4%)
(150, 125)	Net Profit	15.89	-0.45	15.44
TRB	No.Stocks	41	11	52(78.8%)
(125)	Net Profit	12.15	-0.25	11.90
BBs	No.Stocks	39	13	52(75.0%)
(30, 2.3)	Net Profit	15.44	-0.33	15.11
RSI	No.Stocks	43	9	52(82.7%)
(13, 80, 30)	Net Profit	5.40	-0.26	5.14
STO	No.Stocks	46	6	52(88.5%)
(10, 3, 90, 20)	Net Profit	7.47	-0.16	7.31
MACD	No.Stocks	37	15	52(71.2%)
(100, 40, 15)	Net Profit	7.51	-0.40	7.11
OBVA	No.Stocks	38	14	52(73.1%)
(75, 50)	Net Profit	6.75	-0.24	6.51

[a] Profitable stock means the stock whose final equity is more than its initial equity and vice versa.
[b] Profit ratio = (number of profitable stocks) / (total number of stocks).
[c] No.Stocks is the number of stocks. Net Profit is in million dollars.

The parameters of each trading rule are given in the parentheses. Among all simple trading rules, the MA (nl=150, ns=125) is the highest profitable rules, and we can also observe that the best MA, TRB and BBs are much better than the best RSI, STO, MACD and OBVA in terms of both profit and risk. It is also notable that RSI and STO perform poorly even though their Win% are high. One reason may be that they earn little in many winning trades but lose too much in some losing trades. When compared with simple trading rules, the annual net profit of PRS is much higher than the best seven trading rules in the testing period. However, the Sharpe ratio and payoff ratio of PRS are not the best. This is mainly because PRS is optimized without considering risk. Nevertheless, PRS can generally take the advantage of combining component rules and make more profit by scarifying a little bit more risk.

Table IV gives more details about the 52 stocks' profits (each stock is assigned an initial equity \$100,000) of PRS and the seven best component rules in the testing period. The annual net profit (20.22 million) of profitable stocks generated by PRS is higher than any of the seven rules. Although the loss (-0.27 million) of nonprofitable stocks generated by PRS is not the smallest (-0.16 million), it is acceptable for trading especially when PRS earns a lot from most of the stocks in the market (84.6% profit ratio). It shows that PRS can generate significant profit (44 stocks with 20.22 million net profit) from those profitable stocks and recover from those few nonprofitable stocks (8 stocks with -0.27 million loss).

D. Parallel PSO Set-up

To investigate the speedup of proposed parallel PSO based on Hadoop, performance experiments are run on the China

National Grid (CNGrid) point at the University of Hong Kong. An 8 nodes cluster is setup for the parallelization of PSO. A node is assigned as the master node and the other 7 nodes are assigned as slave nodes. Each node has 16 Intel(R) Xeon(R) CPU (E5540@2.53GHz) and 16 Gigabytes main memory. As the running of PRS evaluation needs a large main memory size, we allow each slave node to run 6 Mappers simultaneously at most and each Mapper was assigned 2 GB memory. Therefore, there are at most 42 processors running in parallel in the cluster. A stable Hadoop version 0.20.2 is used and all programs are written in Java programming language. For PSO, a swarm of 420 particles is used for optimization, and these 420 particles are divided into 42 particle groups evenly so that the 42 processors can update all groups concurrently.

E. Running Time Estimation

From the experiments for single node PSO, an individual PRS evaluation may take about 3 to 8 seconds (different with the length of review span of PRS), and the average is approximately 4 seconds. As Hadoop needs to initiate a new job for each PSO iteration's computation and all data in memory will be lost after the old job is finished, there are some preprocessing steps such as reading data for each Mapper in a new iteration. The preprocessing may take about 6 seconds for each Mapper. However, serial PSO just needs to do the preprocessing work once and store these necessary data in the main memory for the following iteration. Serial PSO can only update each particle one by one, it may take about 1680 seconds to update 420 particles in one iteration.

For MRPSO, 42 Mappers are initiated at the beginning of an iteration. Then all Mappers do preprocessing work in parallel and the time is about 6 seconds. After preprocessing, Mappers take single particle as input and update it. Once the updating of a particle is done, the Mapper takes another particle which is not updated to update it until all particles are updated. Because all Mappers work concurrently, the running time in map phase is given by:

$$\begin{aligned} time_{map} &= \frac{420(particles) \times 4(seconds)}{42(Mappers)} + 6(seconds) \\ &= 46(seconds) \end{aligned} \tag{7}$$

The intermediate data created by Mappers are then passed to Reducers for each particle's $gbest$ updating. The computation load of Reducers (finding $gbest$ from a small number of possible $gbest$ candidates) is very small and the running time is less than 1 second.

As mentioned above, the MPSO divides particles into 42 groups and there are 10 particles in each particle group. The Mapper updates particles in one group serially, thus the average updating time for one particle group is 40 seconds. Remember there is no reduce phase in MPSO, thus the preprocessing in Mapper includes the calculation of $gbest$. However, it takes less than 1 second and hence the preprocessing time is still estimated as 6 seconds. As a result, the total running time for a Mapper to update 10 particles is 46 seconds. Since there are 42 processors to run the 42 Mappers simultaneously and the running time is nearly the same, one iteration may only take 46 seconds in MPSO, which is the same as MRPSO.

The above estimation is based on the ideal circumstance that the communication between nodes, the I/O cost and the initialization of Mappers and Reducers are zero time consuming. Therefore, the real running time must be longer than the ideal case. It seems that MRPSO and MPSO spend the same time for one PSO iteration's updating, but it is notable that MPSO does not emit intermediate data and there is no communication between Mappers and Reducers as no Reducers exists in MPSO. These advantages of MPSO may help to save a lot of time in optimization.

F. Speed-Up Performance

Figure 2 shows the running time of MPSO, MRPSO and serial PSO for one PSO iteration. The swarm size is 420 and the dimension is 1064. The time is the average of times spent for 10 iterations. The running time of serial PSO is 1671.5 seconds, which is very close to the ideal running time – 1680 seconds. However, the actual running times of MPSO and MRPSO are 72.7 and 95.8 seconds, respectively. Although both of them are larger than the ideal running time – 46 seconds, MPSO is better than MRPSO as expected. MPSO saves 23.1 seconds when compared with MRPSO, which is a 24 percent improvement. The speedup ratios of MPSO and MRPSO in a 42 processors cluster are thus 22.99 and 17.45 respectively, indicating that MPSO has a better speedup ratio than MRPSO.

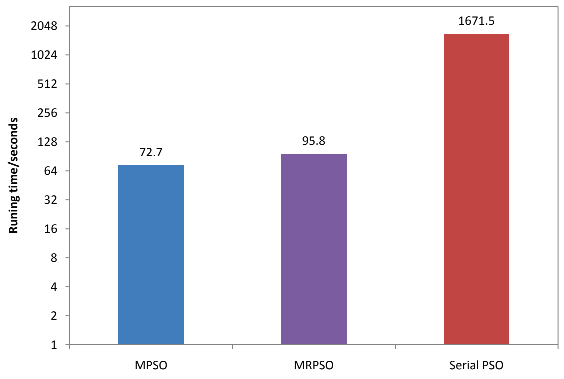

Fig. 2. Running time of one iteration (420 particles, 1064 dimensions)

The main improvements of MPSO can be summarized as below:

1) No intermediate data are emitted by any Mapper. If the intermediate data are too much to fit in memory, they are first stored on the hard disk of local machine and then fetched by Reducers. The I/O cost will be significant if there are too much intermediate data. Therefore MPSO reduces the I/O cost especially when the PSO dimension is very high and hence too much intermediate data are emitted.

2) Less communication between nodes in the cluster. In MRPSO, the intermediate data emitted by different Mappers are passed to Reducers on different cluster nodes, which may be very time consuming if the intermediate data are too much. For MPSO, the $gbest$ candidates generated by Mappers are stored in the distributed file systems, which means they are distributed on the cluster nodes. The loading of $gbest$ candidates in the preprocessing of Mappers will trigger the data transfer between machines. However, there are at most 42 $gbest$ candidates after each iteration and it is very fast to read them from the distributed file systems. Consequently, MPSO reduce the inter-machine communication in cluster.
3) Saving time in intermediate data's sorting and the initialization of Reducers. In Hadoop MapReduce, the intermediate data are sorted before being passed to Reducers. In addition, the startup of Mappers and Reducers need time as well. MPSO avoids time consuming in intermediate data sorting and Reducers startup.

VI. Concluding Remarks

In this paper, a complex stock trading strategy, namely performance-based reward strategy (PRS), is proposed for stock trading. Unlike the previous complex trading strategies, a comprehensive universe of simple trading rules are taken as the components of PRS. Seven popular simple trading rules: Moving Average, Bollinger Bands, Relative Strength Index, Stochastic Oscillator, Moving Average Convergence/Divergence and On-Balance Volume Average are selected as the component rules of PRS. For each type of simple trading rules, we take different parameter values to get a number of various rules representing that rule class's performance in a wide range. Therefore there are 1059 component rules for PRS. The signal of individual component rule is multiplied by a weight assigned to this rule and the trading signal of PRS is determined by the weighted sum of component rules' signals. Considering the dynamic stock market, a reward/penalty mechanism is proposed to update component rules' weights over time. All component rules are evaluated at a regular time interval, then a good rule is rewarded by increasing its weight and a bad rule is penalized by deducting some weight from it.

To decide the best set of starting weights of component rules and some other parameters of PRS, a time variant Particle Swarm Optimization (TVPSO) is used to optimize PRS. The optimization of PRS is a very time consuming job since it involves a large number of component rules and training data. In this regard, the TVPSO is parallelized with Hadoop based on an open source implementation of MapReduce to speedup the optimization. Unlike the typical MapReduce PSO, the proposed parallel PSO with Hadoop lets the Mappers to access particles directly from distributed file systems and calculate $gbest$ without initiating any Reducers. It significantly cuts down the I/O cost as no intermediate data is created and reduces the data transfer among cluster nodes. The experimental results show that the proposed parallel PSO has better speedup ratio than MRPSO and it indeed saves a lot of time for the optimization of PRS.

The empirical results demonstrate that our proposed MPRS has very good profitability in the testing period. The annual net profit generated by PRS is higher than all the component rules. Regardless of the performance fluctuation of the component rules, PRS is able to learn good rules from them and outperforms all of them.

In this paper, the optimization of PRS is done in terms of the annual net profit. However, the risk is another important consideration in stock trading and it is often required to strike a balance between return and risk in investment. Some previous works have optimized trading rules in terms of a risk-adjusted return measure. Therefore, it is worth to optimize PRS based on some risk-adjusted measures and to test its performance in the out-of-sample data set. This interesting work will be studied in the future.

Appendix: Parameter values of PRS component rules

A. Moving Average (MA)

In MA, there are two averages of stock prices over two moving windows of nl days and ns days as follows:

$$Avg_{t,nl} = \frac{1}{nl}\sum_{i=t-nl+1}^{t} p_i, \quad Avg_{t,ns} = \frac{1}{ns}\sum_{i=t-ns+1}^{t} p_i$$

where $nl > ns$, t is the current trading day and p_i is the close stock price on day i.

The signal generation of MA is simple. Consider a trading day t, MA initiates buy (sell) signal if the short-period moving average $Avg_{t,ns}$ is above (below) the long-period moving average $Avg_{t,nl}$. The two parameters of MA are set as: nl = 15, 20, 25, 30, 40, 50, 75, 100, 125, 150, 175, 200, 250; ns = 1, 2, 5, 10, 15, 20, 25, 30, 40, 50, 75, 100, 125, 150, 175, 200.

B. Trading Range Breakout (TRB)

Trading Range Breakout is also called Channel Breakout. It calculates the highest and lowest close price of past n days as follows:

$$H_{t,n} = \max(p_{t-1}, p_{t-2}, ..., p_{t-n}), \quad L_{t,n} = \min(p_{t-1}, p_{t-2}, ..., p_{t-n}).$$

The highest and lowest prices form a running channel (trading range) for each day's stock price and the trading signals are invoked by the stock price's breakout from the channel. A buy signal is generated when $p_t > H_{t,n}$ and a sell signal is generated when $p_t < L_{t,n}$, otherwise a null signal is generated. The only parameter of TRB is set as: n = 10, 15, 20, 25, 30, 35, 40, 45, 50, 60, 70, 75, 80, 90, 100, 125, 150, 175, 200, 250.

C. Bollinger Bands (BBs)

Bollinger Bands is a volatility indicator that considers the fluctuations of stock prices. For trading day t, BBs calculates an n-day moving average of past close prices $Avg_{t,n}$, which is the middle band. An upper band and lower band are k times standard deviation above and below from the middle band, respectively. The upper and lower bands form a price channel. Essentially, a buy signal is generated when the close price fall below the lower band, and a sell signal is generated when the close price is above the upper band, otherwise a null signal is generated. The parameters of BBs are set as: n = 10, 15, 20, 25, 30, 35, 40, 45, 50, 60, 70, 75, 80, 90, 100, 125, 150, 175, 200, 250; k = 1.5, 1.6, 1.7, 1.8, 1.9, 2.0, 2.1, 2.2, 2.3, 2.4, 2.5.

D. Relative Strength Index (RSI)

The Relative Strength Index is a very popular oversold/overbought indicator which measures the velocity and magnitude of directional price movements. An n-day RSI of a trading day t is calculated as follows:

$$RSI = 100 - \frac{100}{1 + Avg.U/Avg.D}$$

where $Avg.U$ is the average of all up price moves and $Avg.D$ is the average of all down price moves in an n-day period. Note that RSI oscillates between 0 and 100. There are many different ways to generate trading signals from RSI. Typically, a trading rule based on RSI emits a buy signal when RSI rises back above the oversold threshold os, and a sell signal when RSI falls back below the overbought threshold ob. The parameters of RSI are set as: n = 11, 12, 13, 14, 15, 16, 17, 18, 19, 20; ob = 80, 75, 70; os = 20, 25, 30.

E. Stochastic Oscillator (STO)

The Stochastic Oscillator is another momentum indicator which is also very popular with traders. The calculation of STO involves the high, low and close prices in an n-day look-back period. The STO on trading day t is given as follows:

$$STO = 100 \left(\frac{p_t - p_{lowest}}{p_{highest} - p_{lowest}} \right)$$

where p_{lowest} and $p_{hihgest}$ are the lowest low and highest high prices in the look-back period, respectively. Similar to RSI, STO is between 0 and 100.

The STO is usually smoothed with a m-day moving average to form the fast %K. The fast %K is then smoothed with another m-day moving average to form the fast %D. Various kinds of signal generation methods are proposed and used in practice. Typical method includes an oversold and overbought thresholds. Buy signal is generated when fast %D line is below the oversold threshold os and accompanied with that the fast %K line rises above the fast %D line. Sell signal is generated when fast %D line is above the overbought threshold ob and accompanied with that the fast %K line falls below the fast %D line. The four parameters of STO are set as: n = 5, 10, 15, 20, 25, 50, 75, 100, 125, 150, 200, 250; m = 3, 7, 11; ob = 80, 85, 90; os = 10, 15, 20.

F. Moving Average Convergence/Divergence (MACD)

The MACD indicator is a combination of two exponential moving average (EMA) of close price. An n-day EMA on day t is calculated as follows:

$$E.Avg_{t,n} = \alpha p_t + (1-\alpha) E.Avg_{t-1,n} \text{ with } \alpha = \frac{2}{1+n}.$$

Similar to MA, there are a long-period exponential moving average $E.Avg_{t,nl}$ and a short-period exponential moving average $E.Avg_{t,ns}$. The MACD is the difference between these two exponential moving averages, which is given by:

$$MACD = E.Avg_{t,ns} - E.Avg_{t,nl}$$

Another m-day EMA of the MACD which is called the MACD signal, is calculated for the signal generation. A buy signal is emitted when the MACD line crosses above the MACD signal line from the below, and a sell signal is emitted when the MACD line crosses below the MACD signal line from the above. The parameters of MACD are set as: nl = 20, 25, 30, 40, 50, 100; ns = 5, 10, 15, 20, 30, 40; m = 7, 9, 11, 13, 15.

G. On-Balance Volume Average (OBVA)

As a volume based trading rule, OBVA is the same as MA except that it calculates MA with stock volume instead of stock price. Although the two parameters of OBVA are the same as MA, their values are a little bit different: nl = 5, 10, 15, 20, 25, 30, 40, 50, 75, 100, 125, 150, 175, 200, 250; ns = 1, 2, 5, 10, 15, 20, 25, 30, 40, 50, 75, 100, 125, 150, 175, 200.

References

[1] F. Wang, P.L.H. Yu, and D.W. Cheung, Combining technical trading rules using particle swarm optimization, *Expert Systems with Applications*, vol. 41, pp. 3016-3026, 2014.

[2] W. Brock, J. Lakonishok and B. LeBaron, Simple technical trading rules and the stochastic properties of stock returns. *Journal of Finance*, pp. 1731-1764, 1992.

[3] R. Gencay, The predictability of security returns with simple technical trading rules. *Journal of Empirical Finance*, vol. 5, pp. 347-359, 1998.

[4] R. Sullivan, A. Timmermann and H. White, Data-snooping, technical trading rule performance, and the bootstrap. *Journal of Finance*, pp. 1647-1691, 1999.

[5] L. Kestner, *Quantitative Trading Strategies: Harnessing the power of quantitative techniques to create a winning trading program*, McGraw-Hill Professional, 2003.

[6] M. Austin, G. Bates, M. Dempster, V. Leemans and S. Williams, Adaptive systems for foreign exchange trading. *Quantitative Finance*, vol. 4, pp. 37-45, 2004.

[7] P. Hsu and C. Kuan, Reexamining the profitability of technical analysis with data snooping checks. *Journal of Financial Econometrics*, vol. 3, pp. 606-628, 2005.

[8] R. Pardo, *The Evaluation and Optimization of Trading Strategies*, Wiley, 2008.

[9] M. Pring, *Technical analysis explained: the successful investors guide to spotting investment trends and turning points*, McGraw-Hill, 1991.

[10] A. Ratnaweera, S. Halgamuge and H. Watson, Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. *IEEE Transactions on Evolutionary Computation*, 8(3), pp. 240-255, 2004.

[11] J. Dean and S. Ghemawat, MapReduce: Simplified data processing on large clusters. *Communications of the ACM* vol. 51, 107-113, 2008.

[12] M. Panella and G. Martinelli, Neurofuzzy networks with nonlinear quantum learning, *IEEE Transactions on Fuzzy Systems*, vol. 17, no. 3, pp. 698-710, Jun. 2009.

[13] R.E. Precup, R.C. David, E.M. Petriu, S. Preitl and M.B. Radac, Novel adaptive gravitational search algorithm for fuzzy controlled servo systems, *IEEE Transactions on Industrial Informatics*, vol. 8, no. 4, pp. 791-800, Nov. 2012.

[14] M.Z. Ali, K. Alkhatib and Y. Tashtoush, Cultural algorithms: Emerging social structures for the solution of complex optimization problems, *International Journal of Artificial Intelligence*, vol. 11, no. A13, pp. 20-42, Oct. 2013.

[15] S.K. Saha, S.P. Ghoshal, R. Kar and D. Mandal, Cat swarm optimization algorithm for optimal linear phase FIR filter design, *ISA Transactions*, vol. 52, no. 6, pp. 781-794, Nov. 2013.

[16] J. Kennedy and R. Eberhart, Particle swarm optimization. In *Proceedings of the Proceedings of IEEE International Conference on Neural Networks*, Piscataway, NJ, vol. 4, pp. 1942-1948, 1995.

[17] J. Robinson and Y. Rahmat-Samii, Particle swarm optimization in electromagnetics. *IEEE Transactions on Antennas and Propagation*, vol. 52, pp. 397-407, 2004.

[18] A. Briza and P. Naval Jr, Stock trading system based on the multi-objective particle swarm optimization of technical indicators on end-of-day market data. *Applied Soft Computing*, vol. 11, 1191-1201, 2011.

[19] Y. Shi and R. Eberhart, A Modified Particle Swarm Optimizer. In *Proceedings of 1998 IEEE International Conference on Evolutionary Computation*, vol. 1, pp. 69-73, 1998.

[20] Y. Shi and R. Eberhart, Empirical study of particle swarm optimization. In *Proceedings of 1999 IEEE International Conference on Evolutionary Computation*, vol. 3, pp. 101-106, 1999.

[21] A. Banks, J. Vincent and C. Anyakoha, A review of particle swarm optimization. Part I: background and development. *Natural Computing*, vol. 6, pp. 467-484, 2007.

[22] W. Sharpe, Mutual fund performance. *Journal of Business*, pp. 119-138, 1966.