An Efficient Implementation of the Backtesting of Trading Strategies

Jiarui Ni and Chengqi Zhang

Faculty of Information Technology,
University of Technology, Sydney,
GPO Box 123, Broadway, NSW 2007, Australia
{jiarui, chengqi}@it.uts.edu.au

Abstract. Some trading strategies are becoming more and more complicated and utilize a large amount of data, which makes the backtesting of these strategies very time consuming. This paper presents an efficient implementation of the backtesting of such a trading strategy using a parallel genetic algorithm (PGA) which is fine tuned based on thorough analysis of the trading strategy. The reuse of intermediate results is very important for such backtesting problems. Our implementation can perform the backtesting within a reasonable time range so that the tested trading strategy can be properly deployed in time.

1 Introduction

Backtesting also known as Systems Testing is the concept of taking a strategy and going back in time to see what would have happened if the strategy had been faithfully followed. The assumption is that if the strategy has worked previously, it has a good but not certain chance of working again in the future and conversely if the concept has not worked well in the past, it probably will not work well in the future.

The backtesting of trading strategies is important for brokers and investors to judge if the strategies are profitable under certain circumstances. It helps the users learn how a trading strategy is likely to perform in the marketplace. It also provides the users with the opportunity to improve a trading strategy. A detailed discussion of the benefits of backtesting is given by [6].

Due to the many benefits of backtesting, it is widely used by brokers and investors. And there are a lot of backtesting systems available in the market, for example, MetaStock from Equis International (www.equis.com) and TradeStation from TradeStation Securities (www.tradestation.com), etc. These systems help the users develop and back test their own trading systems.

Early trading strategies such as Moving Average Crosses were relatively simple and easy to implement and test. As more and more people join in the game of searching for better trading systems, more complicated trading strategies are investigated. Intraday data instead of interday data are utilized, which increases the data to be processed by a factor of hundreds or even thousands. More complicated indicators that are hard to calculate are exploited. Furthermore, people

Y. Pan et al. (Eds.): ISPA 2005, LNCS 3758, pp. 126–131, 2005.

sometimes have to try a strategy against multiple stocks and even multiple markets. All these factors make the backtesting of these trading strategies much more time-consuming, and the ready-for-use commercial products become incapable of dealing with them. People need more efficient implementations in order to perform the backtesting of such trading strategies within an acceptable time range.

In the following sections, we will analyze a simplified trading strategy in detail and present an efficient implementation of it using parallel genetic algorithm (PGA) based on the analysis.

2 A Simplified Trading Strategy

Today's trading strategies tend to exploit several indicators and filters in combination to make the final decision. To make the discussion easier, we will introduce a simplified trading strategy in this section.

Our simplified trading strategy exploits a modified Bollinger band only. Bollinger band is among the most popular technical analysis techniques. It includes 3 lines: the upper band, the lower band, and the center line. The center line is simply the moving average, and the upper and lower bands are, respectively, the center line plus/minus twice the standard deviation [5]. In our strategy, the standard deviation is no longer timed by a fixed coefficient of 2. Instead, the coefficient becomes a variable to be optimized, and its value can be different for upper and lower bands. Given the price series $P_n(n = 1, 2, 3, \cdots)$, the center line C_n, upper band U_n and lower band L_n of a p-period Bollinger band can be calculated as follows:

$$C_n = \frac{1}{p} \sum_{i=n-p+1}^{n} P_n \ , \tag{1}$$

$$U_n = C_n + V_u \times \sqrt{\frac{\sum_{i=n-p+1}^{n} (P_n - C_n)^2}{p-1}} \ , \tag{2}$$

$$L_n = C_n - V_l \times \sqrt{\frac{\sum_{i=n-p+1}^{n} (P_n - C_n)^2}{p-1}} \ , \tag{3}$$

where V_n and V_l are variables to be optimized, and we assume they vary between 1.0 and 2.0 with a step of 0.1. Besides, we assume that p can take any integer value between 11 and 50.

The financial explanation for Bollinger band is as follows: the closer the prices move to the upper band, the more overbought the market, and the closer the prices move to the lower band, the more oversold the market. Based on this understanding, we derive our simple trading rules as follows:

During the normal trading hours, at the end of each bar (a short time period, e.g., 10 or 30 minutes), we evaluate the current price. If the price crosses over the lower band, i.e., $P_{n-1} <= L_{n-1}$ AND $P_n > L_n$, buy 100 shares at the beginning of the next bar. If the price crosses under the upper band, i.e., $P_{n-1} >= U_{n-1}$

AND $P_n < U_n$, sell all shares in hand at the beginning of the next bar. To make the discussion simple, we assume that our trading volume is small enough that our buy/sell orders can be executed at the current price without delay. We also ignore the brokerage fee here. Of course, the real trading strategies have to deal with all these issues.

The purpose of the backtesting of this trading strategy is to answer the following questions: 1) Can this strategy make profit when applied to certain stocks for a time period such as one year (the training period)? 2) If it can make profit for a certain stock, what values for the parameters p, V_u, V_l can give the most profit? 3) Can these values also give a reasonable profit during the following time period such as the next six months (the testing period)?

Question 3 is a simple yes-or-no question which can be easily answered by running the trading strategy once with the values given by question 2. However, because the trading strategy is very sensitive to the change of any parameter, there is no simple relation between the profit and the parameters. Therefore, question 1 and 2 can not be easily answered before trying all the possible triples of (p, V_u, V_l).

3 A Direct Implementation of the Backtesting Problem

The simplest approach to the above optimization problem can simply loop through all the possible triples of (p, V_u, V_l) and run the trading strategy with each triple. The following pseudo-code illustrates this approach.

```
Algorithm 1
for(p = 11; p <= 50; p ++) {
    Calculate Cn and standard deviation Dn;
    for(Vu = 1.0; Vu <= 2.0; Vu += 0.1) {
        Un = Cn + Vu * Dn;
        for(Vl = 1.0; Vl <= 2.0; Vl += 0.1) {
            Ln = Cn - Vl * Dn;
            Run trading strategy with Cn, Un, and Ln;
        }
    }
}
```

Note that C_n and D_n only have to be calculated once for each value of p, and that U_n only has to be calculated once for each pair of (p, V_u). The idea is to reuse the intermediate results as much as possible. Even in this simple case, experiments show that the execution time is reduced by a factor of 10 with the reuse of intermediate values of C_n, D_n and U_n.

Alg. 1 is easy to implement and understand. However, the running time is too long for really complicated trading strategies. While backtesting one trading strategy provided by our industrial partner, our initial try with such an approach took 8 hours to work thru only 1/40 of the whole search space on one single stock. Obviously, The users can not wait that long for the backtesting result. We have to reduce the computing time.

4 Using PGA

A straightforward way to speed up Alg. 1 is to parallelize its execution. Once a triple of (p, V_u, V_l) is given, a processing element (PE) can execute the whole trading strategy by itself. If we have N PEs, we can easily partition the search space into N equal subspaces and then start one process to deal with each subspace. Because there are no communication or synchronization requirements, the parallelization causes almost no overhead in this problem. However, this approach is limited by the number of available PEs. Normal brokers or investors can not expect to have more than 1 or 2 CPUs ready for use at any time. We need better software solutions.

Genetic algorithms (GAs) have demonstrated to be particularly successful in the optimization, classification and control of very-large-scale and varied data. PGAs further provide the basis for tackling problems in a wider range of fields [4]. GAs and PGAs have been widely used in many disciplines from astronomy [2] to molecular design [1]. [3] discusses many ways in which GAs can be parallelized, including the master-slave model. Based on this model, a basic PGA for our backtesting problem is developed as follows:

```
Algorithm 2 -- Master process
Generate random population of n triples;
While(true) {
    Partition the population into N equal groups;
    Send the triples in each group to one slave process;
    Receive the fitness value for each triple from the slaves;
    Exit if no better fitness value is found;
    Select triples with better fitness from the population;
    Generate new population by crossover and mutation;
}

Algorithm 2 -- Slave process
While(true) {
    Receive a triple of (p, Vu, Vl) from the master process;
    Calculate Cn and standard deviation Dn;
    Un = Cn + Vu * Dn;
    Ln = Cn - Vl * Dn;
    Run trading strategy with Cn, Un, and Ln;
    Send the profit (fitness) to the master process;
}
```

The crossover and mutation operations used here are very simple. Suppose we have two triples (p_1, V_{u1}, V_{l1}) and (p_2, V_{u2}, V_{l2}), the crossover operation randomly select p, V_u and V_l independent of each other from the first or second triple and form a new triple. And the mutation operation just randomly change the value of one variable in a given triple to create a new triple.

Note that because the triples are randomly generated in Alg. 2, we can no longer reuse the intermediate values of C_n, D_n and U_n in the way we did in

Alg. 1. This decreases the efficiency of the algorithm. To alleviate this unwanted effect, we have to refine the algorithms to take advantage of the intermediate results as much as possible. The only change to the master process is that the population should be sorted by p and V_u before it is partitioned into N equal groups. The refined algorithm for the slave process is showed below:

```
Algorithm 3 -- Slave process, refined
p_old = Vu_old = Vl_old = newpFlag = 0;
While(true) {
    Receive a triple of (p, Vu, Vl) from the master process;
    if(p != p_old) {
        Calculate Cn and standard deviation Dn;
        p_old = p; newpFlag = 1;
    }
    if(newpFlag > 0 || Vu != Vu_old) {
        Un = Cn + Vu * Dn;
        Vu_old = Vu;
    }
    if(newpFlag > 0 || Vl != Vl_old) {
        Ln = Cn - Vl * Dn;
        Vl_old = Vl;
    }
    Run trading strategy with Cn, Un, and Ln;
    Send the profit (fitness) to the master process;
}
```

5 Performance Evaluation

For the simplified trading strategy described in this paper, we have carried out a set of experiments for all the algorithms described above to illustrate the effect of each algorithm. We performed the backtesting of each algorithm over 1 year's period against one stock ANZ from the Australian stock market. 30-minutes bar data (open price, close price, best bid, best ask) were used in the experiment. The data were stored in a text file and read by the algorithms at startup. To count the execution time more accurately, each algorithm was repeated 20 times. The average execution time is shown in the upper half of Table 1.

The parallelization introduces some overhead. And for this simple trading strategy, the overhead is quite noticeable. When running in parallel on 4 CPUs, it took Alg. 1 much longer than a quarter of the time it needed when running sequentially. And 8 CPUs make hardly any difference than 4 CPUs.

The crossover and mutation operations in the PGAs also result in some overhead that is noticeable in the backtesting of this simple strategy. Alg. 2 is much slower than Alg. 1 when both running in parallel on 8 CPUs. However, Alg. 3 shows less running time that Alg. 1. It means that the PGA can reduce the execution time. The difference between Alg. 2 and Alg. 3 emphasizes the importance of reusing the intermediate results wherever possible.

Table 1. Execution time for different algorithms

Trading strategy	Algorithm	Execution time
Simplified	Alg. 1 (sequential)	87 s
	Alg. 1 (parallel on 4 CPUs)	24 s
	Alg. 1 (parallel on 8 CPUs)	22 s
	Alg. 2 (parallel on 8 CPUs)	32 s
	Alg. 3 (parallel on 8 CPUs)	18 s
Real	Alg. 1 (sequential)	ca. 300 h
	Alg. 1 (parallel on 8 CPUs)	40 h
	Alg. 3 (parallel on 8 CPUs)	$1 \sim 2$ h

For the real trading strategy we tested for our industry partner, we have got the following results as shown in the lower half of Table 1. In this case, the speedup factor of multiple PEs is very apparent. When executed in parallel on 8 CPUs, the execution time of Alg. 1 is reduced to near 1/8. The PGA achieves an speedup factor of at least 20 comparing with the parallel version of Alg. 1. Finally we are able to back test the trading strategy against one stock within $1 \sim 2$ hours. This makes the backtesting of the complicated trading strategy against multiple stocks and multiple markets feasible.

6 Conclusion

In this paper, we have demonstrated step by step the implementation of the backtesting of a complicated trading strategy. We have shown that PGAs can speedup the backtesting process greatly. Furthermore, the reuse of intermediate results is very important for accelerating the backtesting of complicated trading strategies. We believe that the same principles used in this paper can and should be applied in the implementation of the backtesting of other complicated trading strategies as well to make the backtesting feasible.

References

1. Clark, D. E. (Ed): Evolutionary Algorithms in Molecular Design, Wiley-VCH, Weinheim, 2000.
2. Metcalfe T. S., Charbonneau, P.: Stellar structure modeling using a parallel genetic algorithm for objective global optimization, Journal of Computational Physics, Volume 185, Issue 1, p. 176-193., 2003
3. Nowostawski M., Poli R.: Parallel genetic algorithm taxonomy, Proc. of 3rd Int. Conference on Knowledge-based Intelligent Information Engineering Systems (KES'99), Adelaide, Australia, 1999
4. Stender J.: Parallel Genetic Algorithms: Theory & Applications, IOS Press, Amsterdam, 1993
5. stockcharts.com/education/IndicatorAnalysis/indic_Bbands.html
6. www.traids.com