TABLE OF CONTENTS

SUMMARY

As electronic market systems grow, traders intend to look for new ways of trading in high tech environments, which involve more machine power rather then human efforts. Several researchers have suggested that is possible to build software-agent traders as good as human traders. However, their market models sometimes do not apply as models of real institutions. Therefore, many computer scientists continue testing their strategies in more and more realistic systems.

The dissertation focuses on optimising ZIP traders in persistent shout double auction, which has partly done by Priest and Tol (1998). However, they have not tested in a realistic market system and the algorithm does not work well in several market models. Hence, the work I have carried out is listed following:

- I implemented a new market system, which has more features close to a real market. (From page 19)
- I suggested a new way to implement order book of stock market. (Page 27)
- I modified a PS algorithm for new order book and more realistic market implementation. (Page 19, page 30)
- I have tested and evaluated ZIP, PS and MPS in a new environment.
- I carried out the research of Cliff (2001) about using genetic algorithm (GA) to find the best market input sets. Furthermore, I am the first one who did evaluate optimization of PS agents using GA method.(Page 38)

ACKNOWLEDGEMENTS

I thank Prof. Dave Cliff for all his involvement in my development system and producing the dissertation report.

I should also thank the library systems of Bristol University, which allow me to access almost every document I need.

A great thank for all of my friends, who did have me to correct my writing.

I. INTRODUCTION

Since 1998, Persistent Shout Double Action (PSDA) becomes an attractive testing model for researchers because it has been used widely in stock and commodity markets. Priest and van Tol (1998) introduced a new algorithm for trader-agents based on ZIP traders. However, that implementation version for PSDA was tested in a rather unrealistic market model. Therefore, the aim of this project is to evaluate ZIP traders in a more realistic implementation of PSDA markets. Furthermore, the project will involve automated optimization of sets of parameters for the ZIP strategy in the market-based systems. For these reasons, I will modify ZIP algorithm for PSDA with a fixed time of a trading day. Furthermore, I also try improving the performance of previous version of ZIP in flat supply and demand market models. Finally, Genetic Algorithm methods will be used to optimize the parameters of PS algorithm.

In this dissertation, the first session of the contents is the economics background, which introduces the reader to the economics necessary for this project. Several related trading agents strategies are presented in the "dynamic pricing strategies" section. In the next section, there is a modified strategy and its experiment results. Furthermore, I also introduce a new suggestion to arrange order book. Finally, a genetic algorithm is employed for optimizing the parameter sets of the algorithm.

II. ECONOMIC BACKGROUND

This aim of this chapter is that provide to the readers basic knowledge about the economic terms in order to understand the contents of the dissertation.

a. SUPPLY AND DEMAND CURVES

According to Mankiw (2001), "a market is a group of buyers and sellers of a particular good or service". The buyer group present the demand for products and the sellers are on the supply side. In a free market, the interactions between demand and supply of a product determine its price. In general, the traders always want to buy or sell products for acceptable prices. Moreover, on buyers' own expectation of a best or fair price for the product is often slightly different from the expectation of other buyers. Therefore, it seems clear that if the price of the product, which is set by the producer, is low then more people willing to buy the product and vice versa.

There are also some others variables such as number of buyers, the location of the store or shop that can affect the demand group. On the supplier side, they desire to maximise their profit. However, in a competitive market, they are not always able to raise the price of their product up to any price they want. When the buyers have several equivalent choices available in the market, they most likely select the product with low cost. In consequence, if the quantity demanded is less than the quantity supplied, sellers have to set their prices lower in order to sell all their products. Such a situation is called a *surplus*. There is also a case, called a *shortage* when the buyers want to buy more products than the providers could produce. For example in Figure 1, the point when the price is £2.50, there is a surplus period. Moreover, when the price is £1.50, it is a shortage period.

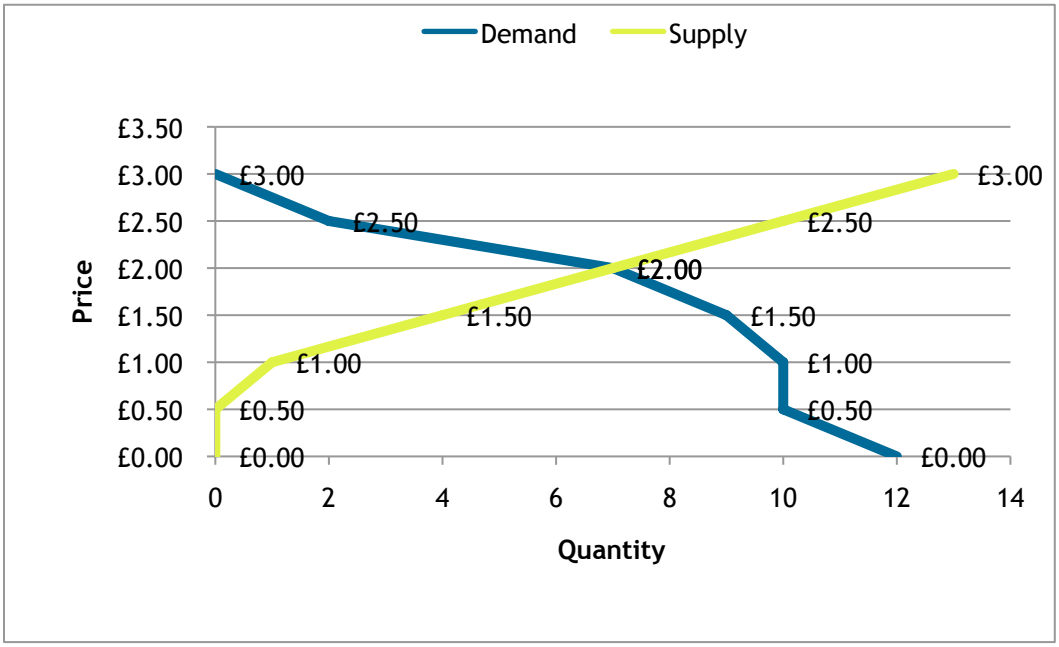

Figure 1 A simple supply demand curves

In such a market, there is a special situation when the supply curve crosses the demand curve. This point is called the market's *equilibrium point*. The price at this point is *equilibrium price* and the quantity is referred to as the *equilibrium quantity*. At the market equilibrium, the buyers are willing to pay at the equilibrium price the same amount of product, which the supplier is able to produce. Moreover, the sellers are also satisfied with that price. For example, in figure 1, the supplier is only produce 7 goods, which also meet their customer demand, and two sides are willing to trade at the equilibrium price £2.00 for all of the products. The trading may happen with the

Another important concept is *market efficiency*, which is the ratio of total actual profit of both buyers and seller and their total maximum profits. For example, when if all traders in the market transact at the equilibrium price, the *allocative efficiency* of the market or market efficiency is 100 percent.

b. AUCTIONS

An auction is *"a market institution with an explicit set of rules determining resource allocation and prices on the basis of bids from the market participants"* (McAfee and McMillan, 1987). In modern life, auctions are an important type of exchange for goods which you can see everywhere in stock markets or the giant websites such as EBay or Amazon. Economists distinguish between many types of auction. For instance, the Double Auction is a kind of auction combining aspects of the English auction (buyers announcing higher bid-prices) and Dutch action (sellers quoting lower offer-prices). In this market mechanism, the buyers and sellers can submit their bids and asks/offers simultaneously during the trading period. However, four common kinds of auctions are: English (first price open cry) auction; first price sealed bid auction; second price sealed bid (Vickrey) auction; and Dutch (descending) auction.

In an English auction, the bidder needs to raise their bid amount anytime when other bidders have committed a higher ask/offer. If there is no one who wants to change the bid price, the winner will be the one who made the bid with the highest price, which is then the price that the winning bidder needs to pay for the good. The first price sealed bid auction mechanism is different from the English one because the participants are only allowed to submit a bid one time. The winner is the bidder, who submitted the highest valued bid. In addition, he has to pay the same amount of money for the object. For both these two kinds of auction, the winner may over-estimate the product and be not satisfied with the price. As a result, the Vickrey auction seems a good choice in this situation because the winner only pays the price of the second-highest bid. Finally, the Dutch auction is opposite to the English one. In this auction, the seller starts with a particular offer-price and then reduces the amount over time. The winner in this auction is the quickest buyer to stop the seller by accepting the current offer-price.

One type of double auction, the Continuous Double Auction (CDA) is used widely in real-world financial markets such as the stock markets and commodity markets. Therefore, in recent years, this type of market-based system has formed a good model for theory and practical research for many computer scientists and economists.

In the CDA mechanism marker, traders can submit bids (buy orders) and offers (sell orders) at anytime during a fixed trading time. A trade is done if any buyer accepts an offer from a seller or any seller agreed with a bid price of a buyer. There is another form of CDA called as persistent shout double auction (PSDA) or "CDA with order queue", which was introduced by Smith and Williams (1983). In

a PSDA, one trading period is divided to many rounds and the trader allows editing or removing their orders in the next round. The New York Stock Exchange uses this type of auction (Priest and Van Tol, 2003).

Many computer scientists have changed slightly the CDA model in order to fit with the problems they want to focus on. The game will be different or less realistic if there is only one good needing to be traded in the market as Herbert (2001) described in his paper. Many works (Das et al. 2001, Cliff and Bruten 1997), based on Smith's (1962) studies, consider the CDA market with fixed roles and fixed strategies. In other words, the trader can only play as a buyer or a seller and uses only one strategy during trading periods. However, with a slightly different view, Posoda *et al.* (2006) allowed traders to change dynamically their strategies in light of their experience, which somehow was more realistic than the other works.

In a continuous double auction or CDA, we have to know some basic terms. Firstly, the traders are only able to make and offer or bid in a **trading day.** The period form the beginning of the trading day until a transaction commit or the time between two committed transaction, named as **trading round (f)**. Next, the **outstanding ask** (OA) is the lowest asking (offer) price for a particular good. As the result, the **valid ask** is an offer with the price less than OA and so the offer price not less than is the **invalid ask**. The idea behind **outstanding bid** (OB) is similar to **outstanding ask**. It is the highest bid in the market. The amount different between **outstanding ask** and **outstanding bid** is called as the **bid-ask spread**.

According to He *et al.* (2003), a protocol for a common CDA market is:

Table 1 : Pseudo-code of CDA protocol. Source: He *et al.* (2003)

1.{Initialize the initial bid/ask : OA= ∞ , OB= 0
when a period starts or a deal takes place}
2.{Several situations might arise during a round**}**
Repeat
{When a seller-agent submits an ask a**}**
if a ≥ OA **then** a is an invalid ask
if OB < a < OA **then** OA is update to a
if a ≤ OB **then**
this seller-agent makes a deal at ob
go to 1
{When a buyer-agent submits a bid b**}**
if b ≤ OB **then** b is an invalid bid
if OB < b < OA **then** OB is update to b
if b ≥ OA **then**
this buyer-agent makes a deal at OA
go to 1
until no new bids (asks) are submitted during a trading period

c. SMITH'S EXPERIMENTAL ECONOMICS

Since the knowledge about a market such as supply and demand of particular products is very important for traders decide its prices, there is a question over "how little the individual participants need to know in order to be able to take the right action." (Hayek, 1945:527)? Surprisingly, according to his past experiments, Smith (1992:p.157), one of fathers of *experimental economics* shows that the typical double auction market reaches rapidly its equilibrium with just a small number of inexperienced traders and strict rules. In other words, the market participants do not know all the information about the market, but it still comes to the point that they are willing to sell/buy in a double auction mechanism.

Some people may argue that most of the experiments in his first paper (Smith, 1962) describing a series of his studies in double auction mechanism was based on an uncommon assumption that the subjects are only allowed to trade one unit during the trading day. However, the experiments were intended to cover many cases, which happen in a real market. He even did an experiment when there is a market shock[1]. Moreover, Smith (1982) concluded that the double auction market is the best exchange mechanism in an environment that the traders only know about their private value for their units.

Smith (1962) also shows that the convergence of the transaction prices toward the equilibrium price can be measured from the data of n earlier transaction. He defined this measure by this equation:

$$\alpha = \frac{\sqrt{\left(\sum_{i=1}^{n}(p_i - P_0)^2\right)/n}}{P_0} \qquad (1)$$

where α is standard deviation of n transaction prices around the equilibrium price.

[1] A point is that the supply or demand suddenly changes significantly. In the experiment, Smith gave new prices of the objects to the participants at nearly the end of the trading day.

III. DYNAMIC PRICING STRATEGIES

The chapter focus on introducing several top dynamic pricing algorithms. It also discuss some remain problems with the algorithms.

a. KAPLAN STRATEGY

The Kaplan program was submitted by Todd Kaplan in 1990 for a "Double Auction Tournament" organised at the Santa Fe Institute (REF). Kaplan's program was ranked first in the tournament with surprisingly simple technical implementation. Its strategy is that "wait in the background and let the others do negotiating, but bid and ask get sufficiently close, jump in and steal the deal" (Rust et al, 1993). In the paper of Rust el al (1993), they indicated that Kaplan was very successful in the market with non-Kaplan agents but it is not stable when competing against a market made up of all Kaplan strategies.

After the Kaplan traders won the tournament, there were some experiments between human traders and/or Kaplan agent's traders and the others agents' traders (Posada 2006, Tesauro and Das 2001, Posada and Lopez-Paredes 2007). The results of Kaplan trader's agents are still considerable high compare to some artificial strategies (ZIP, GD). However, in experiences of Tesauro and Das, their version of ZIP and GD perform better than Kaplan agent does in their market model.

b. ZERO-INTELLIGENCE STRATEGY (ZI strategy)

In many years, economists and computer scientists are looking for the answer of the question that how much knowledge or intelligent we need to understand the market. In 2003, table 3 shows that a trading software agent (ZI trader) with "no intelligence, does not seek or maximize profits, and does not observe, remember, or learn"(p.121) performed as good as human traders[2] in CDA market.

Table 2: MEAN EFFICIENCY OF MARKETS (Gode and Sunder, 1993:132)

Traders	Market 1	Market 2	Market 3	Market 4	Market 5
ZI-U	90.0	90.0	76.7	48.8	86.0
ZI-C	99.9	99.2	99.0	98.2	97.1
Human	99.7	99.1	100.0	99.1	90.2

[2] God and Sunder used graduate students of business as human traders.

As illustrated in table 3, there are two kinds of ZI traders in the experiment of Gode and Sunder. They are all use the same uniform distribution random method, but different boundaries. Firstly, ZI-U agents are ZI unconstrained agents, which submit random bids (buyers) or random asks (sellers) in range of 1 to 200. In other words, they can make any deals with between (1-200) even if they will loss money. Secondly, ZI-C agents are ZI with budget constraint. The agents acts like the ZI-U version but, in order to prevent them from losing money, every bids over the private price or any offers below their objects costs from this type of agents will be refused by their double auction mechanism rules.

The results of Gode and Sunder are still questioned by other economists and computer scientists. For example, Brewer *et al.* (2002), Van Boening and Wilcox (1996) claim that ZI agents did not adapt well to more complex models of DA. The papers of Cliff and Bruten(1997, 1997b), however, demonstrates ZI agents are only working well on a specific type of constrained supply and demand curves (A in figure 2).

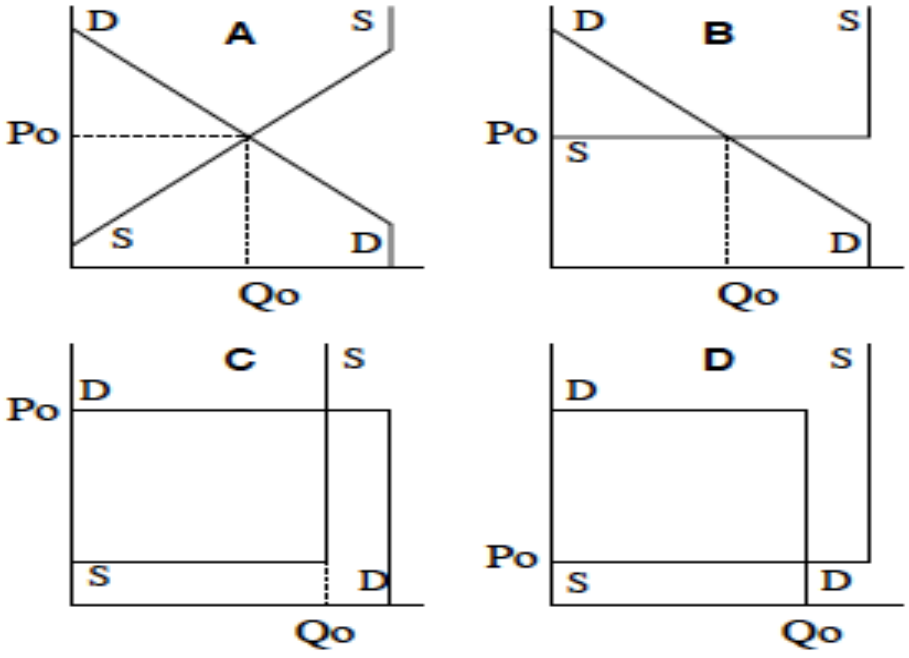

Figure 2. Four types of supply and demand curves. Source: Cliff and Bruten (1997b: 3). Type A is a symmetric supply demand curve. Type B is a flat supply curve. Flat supply and demand curve with excess demand or excess supply could show as graph C or D.

c. ZERO INTELLIGENCE PLUS (ZIP)

Cliff and Bruten invented ZIP algorithm in 1997 in order to prove that software agents still need some information of the market even if it uses a double auction mechanism. They use the institution model similar to the one in Smith (1962).

In general, ZIP traders aim to maximise its potential profit by maintaining the *profit margin* (μ). Each trader has their own limit price for a unit (λ). At time t, a trade i should submit a bid/ask with the value $p_i(t) = \lambda_{i,j}(1+\mu_i(t))$ (*shout-price*) for a unit j. Moreover, their strategy also requires the information of the current traders status (*active*[3] or *inactive*[4]). In particular, the algorithm follow table 3:

Table 3: Pseudo-code for zip trading strategies. Source: Cliff and Burten (1997)

- For SELLERS:
  If (the last shout was accepted at price q then
    1. Any seller s_i for which $p_i \le q$_should raise its profit margin
    2. if (the last shout was a bid) then
       1. Any active seller s_i for which $p_i \ge q$ should lower its margin

  Else
    1. If (the last shout was an offer) then
       1. Any active seller s_i for which $p_i \ge q$ should lower its margin
- For BUYERS:
  If (the last shout was accepted at price q) then
    1. Any buyer bi for which $p_i \ge q$should raise its profit margin
    2. If (the last shout was an offer) then
       1. Any active buyer bi for which $p_i \le q$ should lower its margin

  Else
    1. If (the last shout was a bid) then
       1. Any active buyer bi for which $p_i \le q$ should lower its margin

[3] Subjects are still able to make a deal.
[4] Subjects made enough number of deals. In the original version, it's only required to trade one unit during a trading day.

To update the profit-margin after every shout, Cliff and Bruten (1997) chose the Widrow-Hoff "delta rule" method[5]. Hence, the change in profit-margin at time t is calculated by the following equations:

$$\mu_i(t+1) = (p_i(t) + \Delta_i(t))/\lambda_{i,j} - 1 \quad (2)$$

$\Delta_i(t) = \beta_i(\tau_i(t) - p_i(t))$ (3) is the Widrow-Hoff delta value

$\tau_i(t) = R_i(t)q(t) + A_i(t)$ (4) is the *target price.* $R_i(t)$ and $A_i(t)$ are random perturbations.

Because of a problem with high frequency noise affecting their learning method, they introduce a *momentum coefficient* γ_i to control how many changes an agent should learn. Then we can use equation bellow to calculate the next target price:

$$\Gamma_i(t+1) = \gamma_i\Gamma_i(t) + (1-\gamma_i)\Delta_i(t) \quad (5)$$

From (2), (5) and set $\Gamma_i(0) = 0 \forall i$, we have the update of margin profit equation:

$$\mu_i(t+1) = (p_i(t) + \Gamma_i(t))/\lambda_{i,j} - 1 \quad (6)$$

Cliff and Bruten (1997) show that the ZIP trader agents not only performed better than ZI-C agents, but also its result is close to the experiments of Smith (1962). Moreover, Das et al. (2001) also proved that ZIP agents were out performing on their experiments with human traders.

d. GJERSTAD-DICKHAUT STRATEGY (GD traders)

Based on the experiments of Smith (1982), Gjerstad and Dickhaut(1998) made an argument that analysing the messages of history (H) of actions of trading agents is an important task to help compute the belief functions of sellers and buyers. Therefore, with any ask amount *a,* they decide to calculate the sellers' beliefs function as below:

$$P(a) = \frac{\sum_{d \geq a} TA(d) + \sum_{d \geq a} B(d)}{\sum_{d \geq a} TA(d) + \sum_{d \geq a} B(d) + \sum_{d \leq a} RA(d)} \quad (7)$$

Which also can be rewritten as

[5] Actual output of next step is equal the actual output of current step plus the output different. The different in the output is the multiplication of learning rate coefficient and the different between the outputs this step and our expected output. (Cliff and Bruten, 1997)

$$P(a) = \frac{TAG(a) + BG(a)}{TAG(a) + BG(a) + RAL(a)} \tag{8}$$

In the function, $BG(a)$ is the total number of bids made at a price no less than a. Then, $TAG(a)$ is the frequency of accepted asks at the price no less than a and $RAL(a)$ is the number of refused asked prices no greater than a.

The buyers' beliefs function is constructed:

$$Q(b) = \frac{TBL(b) + AL(b)}{TBL(b) + AL(b) + RBG(b)} \tag{9}$$

Then $TBL(b)$ is the number of sellers' bids less than or equal to b, $AL(b)$ is the number of asks no greater than b and $RBG(b)$ is the frequency of rejected bids' prices no less than b.

In the original work, Gjerstad and Dickhaut were working on a double auction market with no order-queue and only one good in the market. As the result, a particular message of history includes the price of bid or offer and the seller/buyer identify. For experiments in a more realistic model (PCDA), it was extend by Tesauro and Das (2001). Their work involved adding a waiting period ("grace period") before added in to the history message H. Moreover, the modifier version of the algorithm allows traders have their own set of limit price, so they could trade multiple goods if they want.

Although in recent experiments (Gjerstad and Dickhaut 1998, Tesauro and Das 2001, Posada et al. 2005 and Posada 2006), the result of GD agents is really competitive compare to the other top strategies' agents (ZIP, ZI, GD, K), Tesauro and Bredin (2002) still expecting improvement in its performance and efficiency by using the optimisation methods. In their works, the dynamic programming method is use to optimise GD in a CDA market with single unit tradable. Beside of the history vector H, they have added a variable T, which is the time remain before the trading day finish (L: all time of a trading day).

Hence, the bid price at the time left T is

$$P^*(T) = \arg\max(f(p,T)[S_M(p) + \gamma V(M-1, N-1)] + (1 - f(p,T))\gamma V(M, N-1)) \tag{10}$$

where $f(p,T)$ is the belief function at L-T, $s_M(p)$ is the surplus in the M^{th} unit at the price p and $V(m,n)$ is the expected value of m^{th} unit give n chances to bid.
In the table 2, there is an algorithm used to compute table V

Table 4: Pseudo-code of expected value computation in Tesauro and Bredin (2002) `s model CDA

1: **for** n = 1 to N **do**
2: **for** m = 1 TO M **do**
3: $V(m, n) = \begin{matrix} P^*(T) = \arg\max(f(p,T)[S_M(p) + \gamma V(M-1, N-1)] \\ +(1-f(p,T))\gamma V(M, N-1)) \end{matrix}$ (11)
4: **end for**
5: **end for**

The final results of their experiments shows that the optimised version works better than the GD and ZIP agents in an only single tradable CDA market.

e. MODIFIED ZIP FOR PSDA (PS agents)

Priest and Tol (1998) claimed that the market model in Cliff and Burten is not widely use in real would and GD agents are complex and slow. As the result, they have modified ZIP to work in the PSDA model. Furthermore, PS traders allow observing markets when they do not have intent to trade. Moreover, they also introduce a heuristic based on OA and OB for which seems simpler than the one Cliff and Burten (1997) used. However, they use the same learning method Widrow-Hoff with momentum as ZIP traders.

Table 5. Pseudo code of PS agents (Priest and Tol (1998)

For BUYERS;
If $OA > OB$ then
target = $OB + \delta$
If $OA \leq B_{max}$ then
target = $OA - \delta$
For SELLERS;
If $OA > OB$ then
target = $S_{min} - \delta$
If $OA \leq OB$ then
target = $OB + \delta$
	If the target is $OB + \delta$
	then $\delta = r_1 OB + r_2$
	If the target is $OA - \delta$
	then $\delta = r_1 OA + r_2$
	r_1, r_2 is random number in range [0.0,0.2]

Their basic idea is the agent traders can reach equilibrium prices faster if ZIP traders refer to OA or OB rather than their last shouts like the original ZIP traders. Furthermore, in a trading day, a PS agent will maintain their profit margin even when it has no intended to trade any more goods. But, it will not submit any bids/asks in these trading periods. The result (Figure 3 and Figure 4) shows that performances of the heuristics ones are significantly higher than the old versions in a number of supply demand curves (model A in Figure 2). However, PS agents are very slow to equilibrium prices compare to ZIP agents on

flat supply curves or flat supply and demand with either excess demand or excess supply curves (Model B, C, and D in Figure 2).

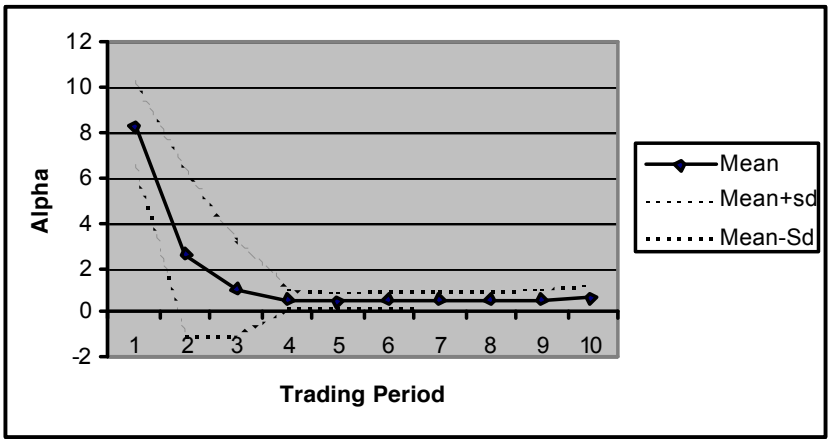

Figure 3. Alpha of PS agents with learning rate of 0.7 (Priest and Tol (2003:12)); lower values are better.

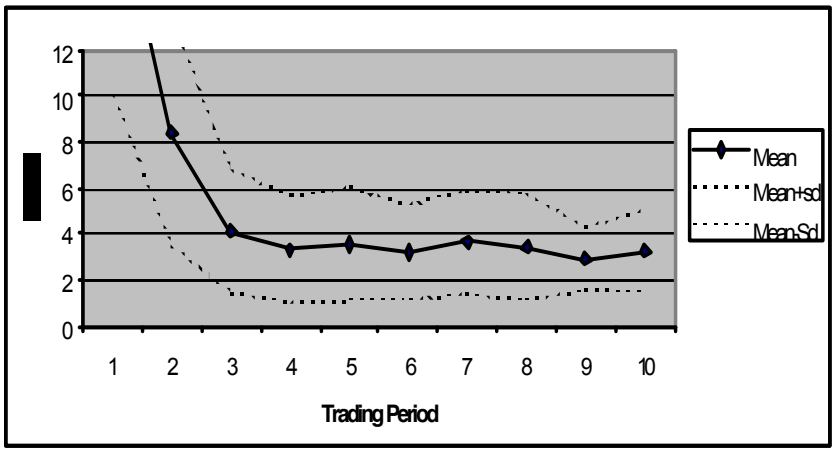

Figure 4. Alpha of ZIP agents with learning rate of 0.7 (Priest and Tol (2003:12))

Priest and Tol (1998,2003) did not figure out the reason but it seems come from the status of PS agents when they are not active. The traders, who already got a deal, can be confused of the offer prices of the last trading agents, who are impossible to get a deal because of their limit prices. Moreover, in the beginning of any trading days, the unsuccessful traders should try to shout at acceptable low price and they may not have many correct information about OA and OB. For example, in flat supply and demand with excess demand curves when the market time does not finish, the sellers will receipt a larger number as OA. Therefore, the algorithm does not work well in flat models.

IV. MODIFIED ZIP AND PS AGENTS FOR PERSISTENT SHOUT DOUBLE AUCTION (MPS)

It seems clear that neither ZIP nor PS agents work totally well in PSDA. They (PS agents) may out perform in normal supply demand curves but too slow in flat models. ZIP, however, is worse in normal models but really successful in flat models. Therefore, a combination of ZIP and PS may be a good solution for trading agents in PSDA market model. The modified version of PS agents will focus on distinguishing between the ones, who have a deal with the ones who have not any deal after the first round. In other words, the unsuccessful trader should learn from the previous day that they have to start with considerable prices. The new strategy suggests these prices should close to last shouts of them.

FOR SELLER:

```
IF the first trading day
      IF the first round
                  ASK                 =
                  limitPrice*(1       +
                  initMargin)
      ELSE
                  ASK = PS
ELSE
      IF the first round
            IF hadDeal
                  ASK = PS
            ELSE
                   ASK      =      (1-
                   cr)*lastShout     —
                   ca
                  IF ASK <limitPrice
                    ASK = limitPrice
      ELSE
                  ASK = PS
```

FOR BUYER:

```
IF the first trading day
      IF the first round
                BID   =   limitPrice*
              (1 - initMargin)
      ELSE
                BID = PS
ELSE
      IF the first round
            IF hadDeal
               BID = PS
            ELSE
               BID                    =
            (1+cr)*lastShout + ca
               IF BID > limitPrice
                  BID = limitPrice
      ELSE
            BID = PS
cr,ca  is  random  number  in  range
[0,0.2]
```

Furthermore, the algorithm does not need any observations from agents who have no intent to trade, unlike the original ZIP. Otherwise, MPS acts the same way as a normal PS trader. It means that each agent works on a set of parameters, which includes a profit margin, learning rate (Widrow-Hoff) and the momentum term for smoothing-frequency noise in the learning method.

a. EXPERIMENTAL SETUP

In order to compare the performance of MPS, PS and ZIP, I used the same experiments as Cliff (1997), which can indicate the stability of a particular group of agents in that market model. However, the experiments of Cliff (1997) or Priest and Tol (1998) do not show that their algorithm could work efficiency in a different supply and demand curves. Therefore, it is necessary to have multiple sets of experiments to explore each type of market models.

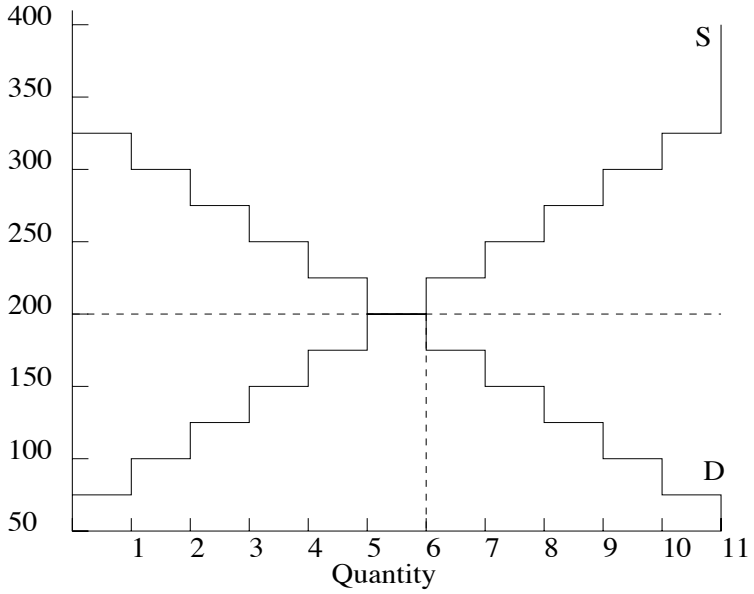

Figure 5: Example of a normal supply demand curve (Cliff (1997:34))

In the random generating experiments' sets, the equilibrium prices will not change much because, in a normal market, the equilibrium price can move a small amount during that a long period. Moreover, the limit prices[6] of the traders may change, as the result of the fact is that they have more or less interest on the trading subject. Therefore, I design a simple mechanism to generate at random the required kind of supply demand curves. At first, an equilibrium price (ep) in an appropriate range[7] is generated from a uniform distribution and then

[6] the price in which sellers is not willing to trade more than or the buyer is not willing to trade less than
[7] Around [1.9,3.2] in all experiments

the limit prices lists of sellers and buyers are produced randomly in the range [0,ep] or [ep,5].

The software system use for the experiment includes 11 sellers' threads, 11 buyers' threads[8] and a processing message object. The numbers of traders' threads can reduce to 6 sellers or 6 buyers with 11 buyers or 11 sellers respectively in order to simulate flat supply and demand with either excess demand or excess supply. A trader thread will run MPS algorithm and send a message with its shout value to the process object. The classified shouts work like as the CDA protocol of He *et al.* (2003), however, the k-auction rules is applied with k=0.5[9]. The system is based on multi threading system and one trader thread can access to the message centre at one time (first in first serve). After the message have taken, the thread will receive a message contained the status of last shout (accept, deny, or deal), OA and OB.

Furthermore, my implementation of the order book works on a market idea that the sellers see a number of potential customers for their products or stocks and they do actually have some information about the boundary of money or commodities of the customers are willing to trade. However, they also have own limit prices, which may be based on the amount of their resources have been spent for the goods. At the beginning of a trading day, their initial shouts were calculated by the setting margin of the modified PS algorithm. All the buyers will submit their bids first and then the sellers will come to the market but the sellers do not know about the buyers' bids. Next, if any seller/buyer meets the price of buyer/seller, a trade will happen. In fact, the traders are delayed 10 milliseconds after their shout to allow the other traders joining the market.

Table 5 illustrates that changes in the booking order do not reduce the random selection of the traders. The trading agents still reach just close to the equilibrium even when they change their partners.

[8] In order to compare with the result of Priest and Tol (1998)
[9] It means the transaction price will be average of buyer's shout price and seller's shout price

Table 6: The simple output after a trading day in one experiment in a normal supply demand curve

Day 1	Day 2
buyer 9 seller 4 got a deal at price 1.94	buyer 5 seller 0 got a deal at price 1.94
buyer 6 seller 1 got a deal at price 2.0	buyer 6 seller 4 got a deal at price 1.9
buyer 10 seller 0 got a deal at price 1.63	buyer 8 seller 1 got a deal at price 1.89
buyer 8 seller 3 got a deal at price 1.95	buyer 10 seller 3 got a deal at price 1.94
buyer 5 seller 5 got a deal at price 2.0	buyer 9 seller 5 got a deal at price 2.0
buyer 7 seller 2 got a deal at price 1.97	buyer 7 seller 2 got a deal at price 1.92
Day 3	**Day 4**
buyer 7 seller 5 got a deal at price 2.03	buyer 9 seller 3 got a deal at price 2.02
buyer 9 seller 0 got a deal at price 2.03	buyer 6 seller 4 got a deal at price 2.01
buyer 8 seller 2 got a deal at price 2.04	buyer 8 seller 2 got a deal at price 2.04
buyer 6 seller 1 got a deal at price 2.0	buyer 10 seller 0 got a deal at price 2.05
buyer 10 seller 4 got a deal at price 2.03	buyer 5 seller 5 got a deal at price 2.0
buyer 5 seller 3 got a deal at price 2.0	buyer 7 seller 1 got a deal at price 2.03

Priest and Tol (1998), in their experiments, allow their agents to trade in a unlimited time which does not happen in many real markets. All normal traders have a fixed time because many markets have a specific open and close time. In my version of simulation PSDA market, a normal trading period appears as 8 seconds only. Sellers and buyers have to work out their deal before the market close. When a seller and a buyer agree on a price, they will leave the market and the next round will begin with the rest of traders.

b. EXPERIMENTAL RESULTS

For the first experiments, I use the second parameter set of Priest and Tol (1998), which is a 0.7 learning rate and a 0.05 momentum. I have set the initial margin to 0.3 because PS agent has a random initial margin and the initial parameters may then also be applied for ZIP agents in order to compare PS agents and ZIP agent in Priest and Tol (1998). The results in Figures 3 and 4 showed that PS agents are more stable than ZIP agent with the same input.

However, PS agents can reach just around 1% and remain stable after 4 trading periods. In the same curve with the same amount of experiments running (50), alpha values decrease sharply from 18% on the first days to around 2.93% on the second days with MPS strategy. Moreover, MPS agents manage to reach 1% and more importantly remain between nearly 0.1% and around 0.3% for the rest periods.

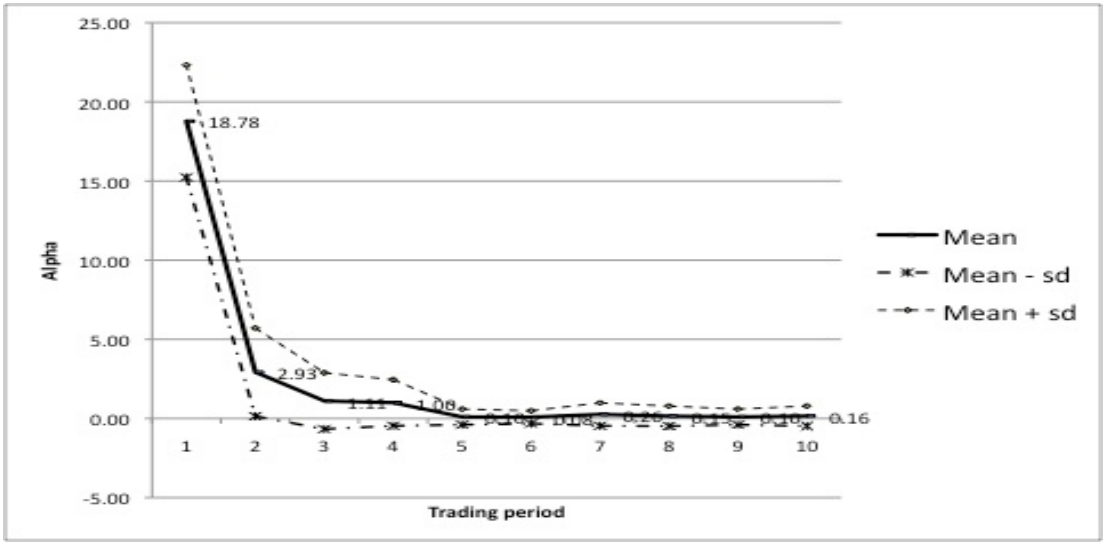

Figure 6: Average alpha of 50 MPS agents experiment with a learning rate 0.7 in a symmetric supply demand curve

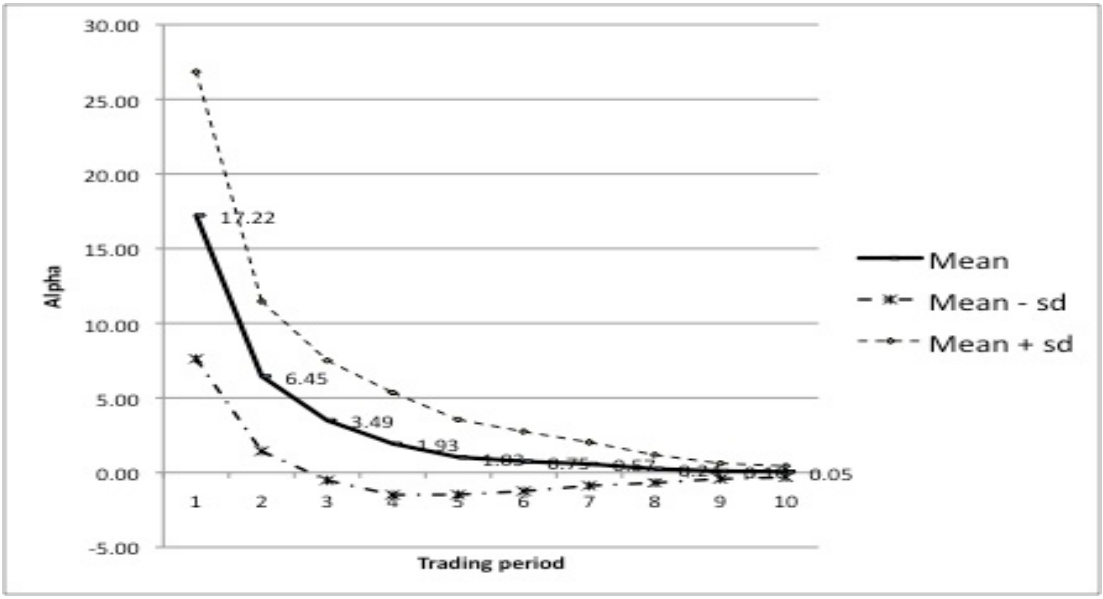

Figure 7: MPS with a learning rate 0.7 in mixed 50 randomise experiments

Furthermore, the modified algorithm seems work well even in a range of different normal supply demand curves (50 different random curves). In Figure 7, the alpha proportion drops quickly and steady from above 17% to just nearly 2% after 4 periods and finishing with 0.05% in day 10. The trend seems still continue after day 10. Moreover, the standard deviation decreases steadily in the experiment. In other words, there are not many changes of the alpha values over random curves.

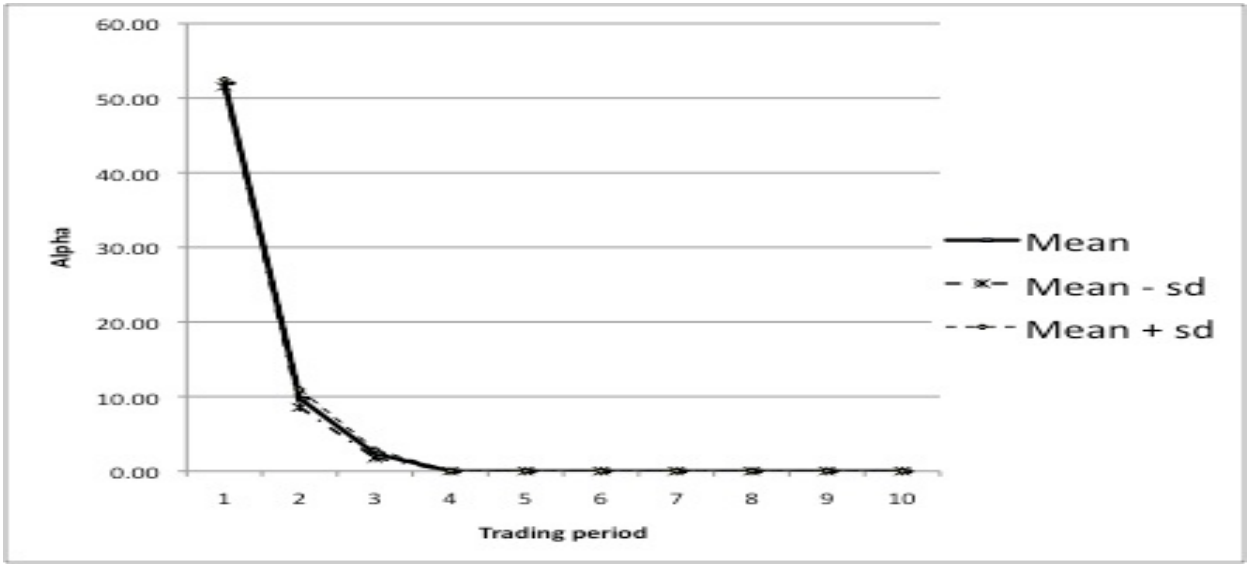

Figure 8: Average alpha of 50 MPS agents with a learning rate 0.7 in a flat supply (3.2) and demand (2) with excess demand

MPS is not only successful in a number of symmetric supply and demand models (model A) but also generates outstanding results with model C. Figure 8 illustrates that although on the first day, MPS agents could not get nearly to the equilibrium price, they have decrease around 5 times per a day from the first day (52%) to the second day (9.75) and then third day (2.33). Between Day 4 and Day 10, the convergence of transaction prices is in a perfect situation.

The Figure 9, however, illustrates that MPS does not outperform every models but the results in Model D also prove the ability of MPS in term finding the equilibrium price in a few trading days. Furthermore, ZIP did not cover the clear market price in 30 days in Cliff's (1997) model of system. Therefore, MPS agents with a learning rate of 0.7 seem to outperform compare to ZIP and PS in PSDA market.

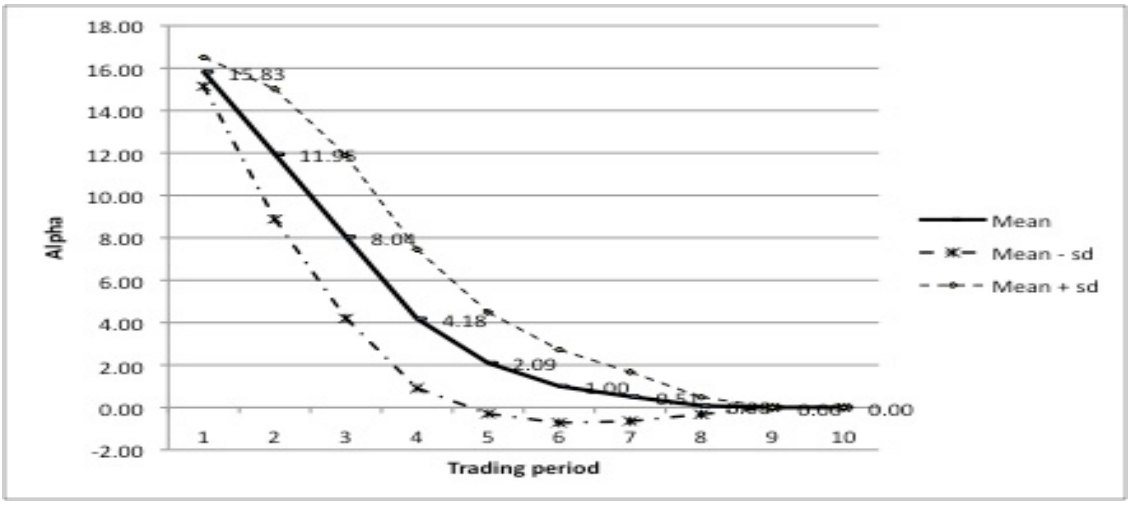

Figure 9: Average alpha of 50 MPS agents with a learning rate 0.7 in a flat supply (3.2) and demand (2) with excess supply

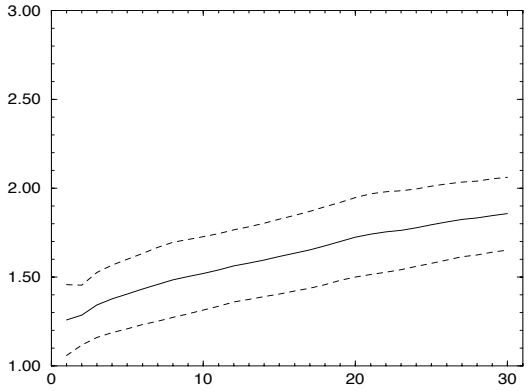

Figure 10: Mean ZIP of transaction price in excess demand mode. (Supply 2.0 Demand 3.2) (Cliff, 1997:47)

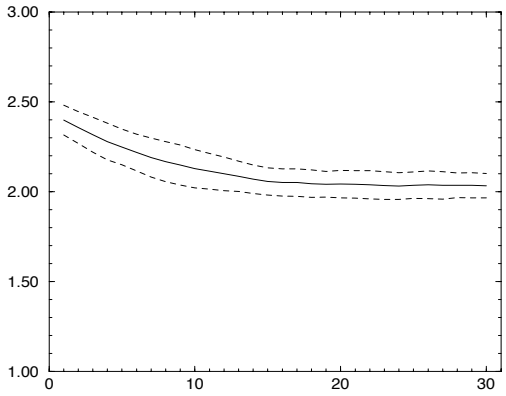

Figure 11: Mean ZIP of transaction price in excess supply mode. (Supply 0.5 Demand 2.0) (Cliff, 1997:47)

While MPS with a learning rate of 0.7 have considerable results, the experiment with a 0.3 learning rate in a basic normal supply and demand curve shows one drawback of MPS. The modified version with the learning rate of 0.3 seems unstable although it did constantly decrease for the first 10 days. Compare to Figure 4, after Day 4, the convergence rates of ZIP traders are around 4% and there is no trend in reaching closer to the equilibrium price.

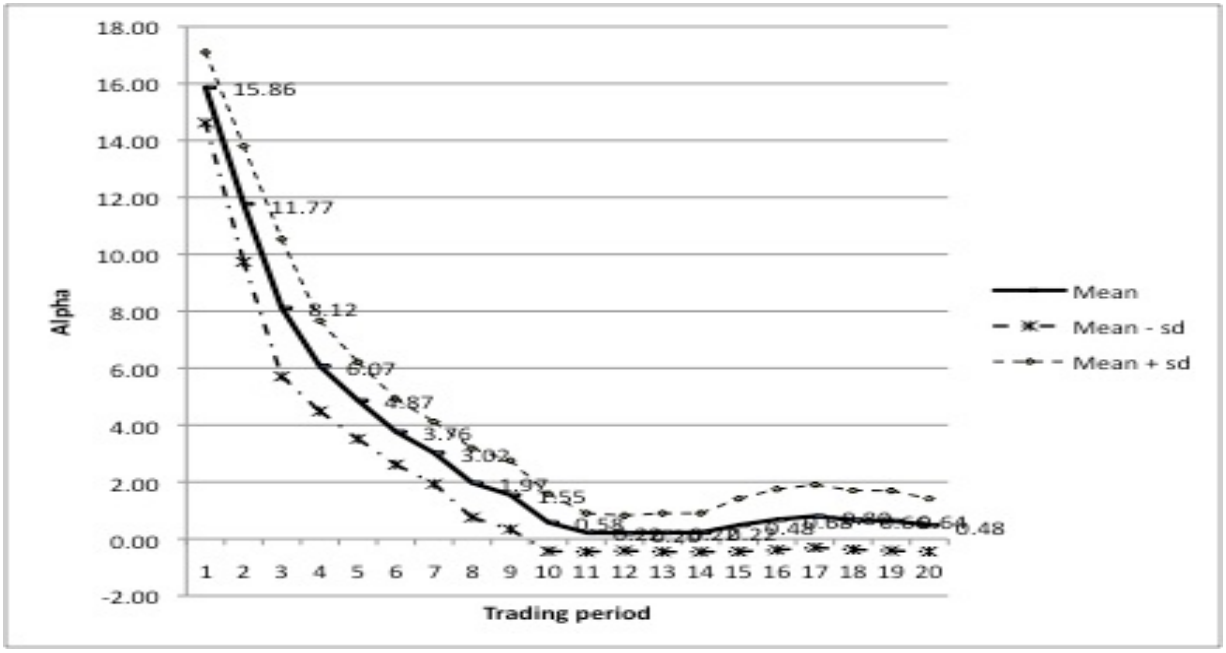

Figure 12: Alpha average after 20 training days and 50 MPS experiments with learning rate 0.3 in a symmetric supply demand curve.

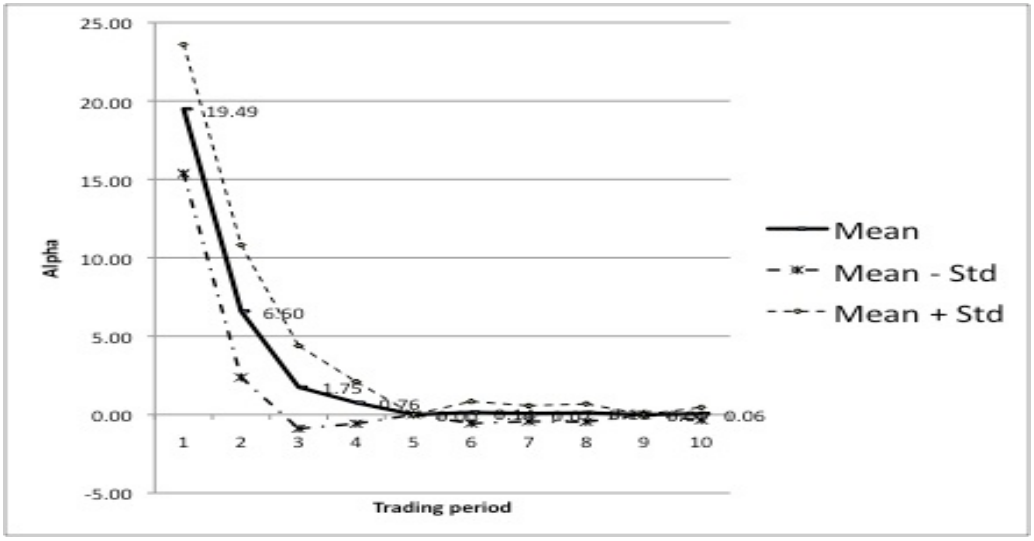

Figure 13: Alpha after 50 MPS experiment (10 days trading of each experiment) and learning rate 1 in a symmetric supply demand curve.

On the contrary, Figure 13 shows a similar result to Figure 6. In other words, these learning values may in sets of parameters, which will perform well in an identical type of market systems.

V. PROBLEMS WITH ORDER BOOK

One of remain problems in testing trading agents strategies is which side seller/buyer should appear in the market first. In the other words, in a commodity market, sellers and buyers come, go and submit their bids anytime they want, thus these trading strategies have to test in practically unpredictable market simulation to prove their performance. Furthermore, in sprite of the differences from the targeting models, simulations of random interactions between buyers and sellers are necessary.

In fact, starting with specific type of ordering agents come to the market can lead to totally differential results. For example, the modified PS strategy is very unstable if started with unordered agents. There are several solutions for the issues of first quote order.

Firstly, Cliff and Bruten (1997) start with a random seller/buyer agent and then the other buyer/seller agents will reply with their bids/asks. If there is more than one answer from the agents, a random one of these agents will be chosen as a match. The method seems right when we look at any particular points of time there is one trader wait of for the other shouts. However, in a real market, the reactions for a bid/ask may come from both of sides. For example, when a buyer commit a bid, he expects ask of an active seller but there is a common case that a challenger bid appear in the market.

Priest and Tol (1998) (2003) used similar method when they divided each trading day to into several rounds where agents' traders can submit and update their bids/asks if they want. A trade happen when the highest bid is equal to or higher than the lowest offer. Their system, however, did not reset after a trade like the New York Stock Exchange (NYSE) as they mentioned. In fact, the trading process will continue until all agents have a trade or reach their limit price. On the other words, there is no limit time to trade. In addition, the time actually spent to learn from only one supply/demand curve may be unacceptable if a number of time rounds could convert to a real trading day. Furthermore, the frequencies of interactions between agents in a round may be the same when rounds are divided equally. On the other hand, a real trading day is sometime very busy and also it is the most important time to take a good deal. Other times, the traders may active slower and there is not much information agents can extract in these periods.

On the other hand, MPS agents were tested firstly in a limited time trading day. Furthermore, the simulation applied the NYSE rule that is a reinstall mode after a trade. In a trading round, trading agents are allowed to post their bids/asks anytime. A trading round finishes when a trade happens or all agents reach their

limit price. The 'time out' uses to stop or finish a running trading round.

In 2001, Das *et al.* introduced their simulation in order to test performances of ZIP and Gjerstad-Dickhaut (GD) agents comparing to human traders. The simulation includes an asynchronous CDA server to control the communication between traders and software agents. The order book was based on random sleep and wake up time of agents' threads. Tesauro and Bredin (2002) show a different approach by using a random probability to decide whether an agent can shout or not at a particular point.

My suggestion for a random order book is that trading agents will be taken as random pairs consisting of a buyer agent and a seller agent drawn from a list of trader agents. The chosen agents will not make any moves before all the agents are ready to trade (open market time). When a pair of traders has a match, the market resets and the traders will not be able to commit any shouts. They, however, can observe the other agents' movements in order to make the right price in next trading days.

Furthermore, all of the previous shouts are removed from the market system. The bidders do not allow buying or selling after the finish time of a trading day[10]. Furthermore, a round may finish without any trades (a 'bad' round) because all agents reach their limit prices. In addition, a trading day can finish earlier than its fixed time, if the rest of the agent have not came to an agreement even with their limit price. In other words, a 'bad' round leads to an early finish. In consequence, every agent encourages submitting shouts even with their limit price.

[10] 4 seconds for a trading day

Table 7: Pseudo-code of randomize book order process

```
Input(List<buyerThread>, List <sellerThread>)
Process :
      maxRemainTraders = buyerList.size()< sellerList.size()?
      sellerList.size(): buyerList.size();
  for (int i = 0; i < maxRemainTraders ; i++) {
      if (i<buyerList.size()) {
            Buyer buyer = getRandomBuyer(buyerList,selectedBuyer)
            buyer.start(); // still have to wait for market open
      notification
            selectedBuyer.add(buyer);
      }
      if (i<sellerList.size()) {
            Seller seller = getRandomSeller(sellerList, selectedSellers)
            selectedSellers.add(seller);
            seller.start();// still have to wait for market open
            notification
      }
   }
  Notify all thread about market opening.
```

Table 8: Pseudo-code of getRandomBuyer

```
Input (buyerList, inTheList)
Process :
            if (inTheList.size() > 0) {
                  buyerList.removeAll(inTheList);// remove the ones have
already chosen
            }
            randomBuyerIndex = U[0,buyerList.size());// get random number
from 0 to size of buyer list -1;
Return : buyers.get(randomBuyerIndex);
```

The following experiments will show unexpected result when combine PS agents with NYSE market rules.

a. EXPERIMENTAL SETUP

The new experiments set upped following the suggestion order book in the previous section. It is focussing on examine how PS agents, MPS agents and ZIP agents work in a random booking order. Furthermore, number of agents is exactly the same with the experiment for MPS agents in ordered book.

After several experiments, the best parameter set of MPS and PS agents includes learning rate (0.3), momentum (0.05) and random perturbations (0.05). The initial margin is generated from a uniform distribution [0, 0.3] in order to establish the beginning shout at the market opening time. In their paper (Priest and Tol, 1998), there is a remaining question when exactly a non-active agent has to stop to observe the market? Do they stop just after get the virtual deal or continue adjusting their price until the market round stop? In my version of implementation, after number of tests, I supported the case that the agents will follow the market until it closes. Indeed, non-active agents may update their quote price too frequency because they do not have to send the price to the main system and wait for feedback. Therefore, every non-active agent should wait for 10 milliseconds (normal agents also wait the same time after their bids/asks)[11]. Furthermore, it does not have any modification non-active agents' status if they shout a correct price, which can win the deal.

b. EXPERIMENTAL RESULTS

Again, ZIP agents have low performance in symmetric supply demand curves even if the non-active mode has been enabled. Figure 14, however, illustrates that PS agents made a compatible good result to the experiment of Priest and Tol (1998) in same symmetric supply demand curve. The alpha's Smith value quickly drops to about 2% in the second periods. Furthermore, mean of alpha keep normally around 0.5% during the next days. Considering the performances of PS agents in a much more random order book and more realistic market implementation, the changes in standard deviations are acceptable.

[11] All strategies would not work well if non-active agents have no limit in their performs.

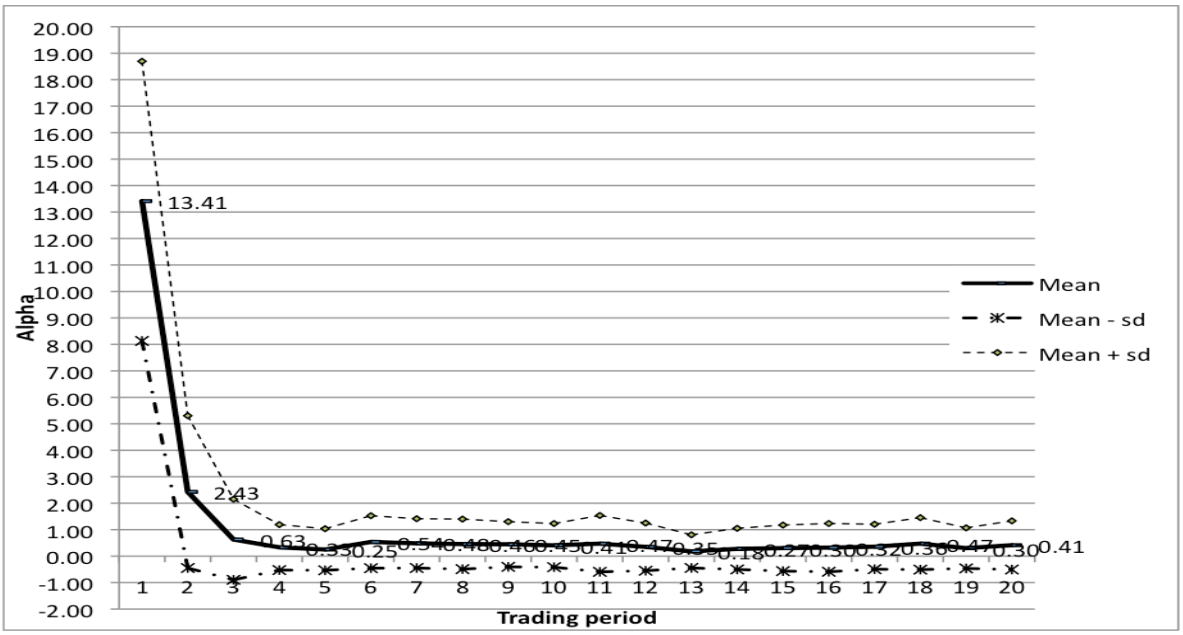

Figure 14 Symmetric supply demand curves. (Alpha of average over 50 PS agents' experiments (20 trading period per each experiment) with learning rate of 0.3, a random starting order and all agents have to wait for an open market time)

The PS agents, however, shows exceptional results in flat supply demand with either excess demand or excess supply. In figure 15 and figure 16, the alpha value decrease sharply from more than 20% to 0% after just one trading days and remain stable at 0%. Hence, the modifications on behaviours of agents, when they are not intended to trade, make some progresses on PS agents rather than ZIP agents in new market environment because Priest and Tol (1998) (2003) commented "… ZIP agents outperform PS agents; when there are flat supply/demand curves, and an excess of either supply or demand."

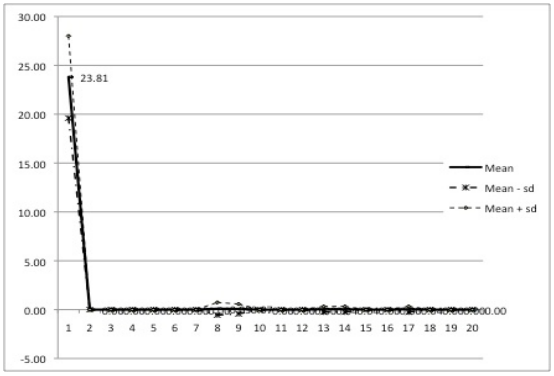

Figure 15: Alpha average over 50 PS agents' experiments (20 trading periods per each experiment) with learning rate of 0.3; a random starting order and all agents has to wait for an open market time. Flat supply and demand with excess supply. (Supply 2.0 Demand 3.2)

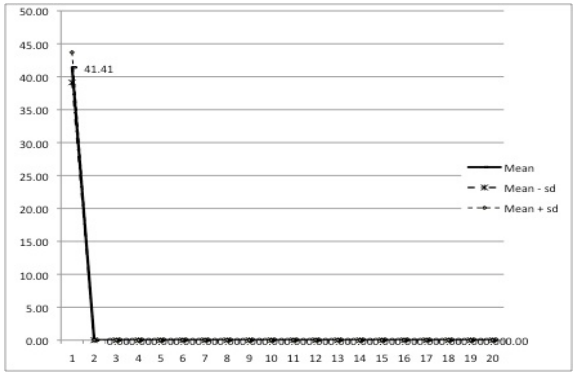

Figure 16: Alpha average over 50 PS agents' experiments (20 trading periods per each experiment) with learning rate of 0.3; a random starting order and all agents has to wait for an open market time. Flat supply and demand with excess demand. (Supply 0.5 Demand 2.0)

Figure 17 illustrates the performance of MPS agents in the same supply demand curve. It is clear that the modified version also work very well in the market. Firstly, the two strategies versions improve gradually just after the second trading periods and constantly produce low Smith's alpha. It means that the transaction prices all cover very near the equilibrium price.

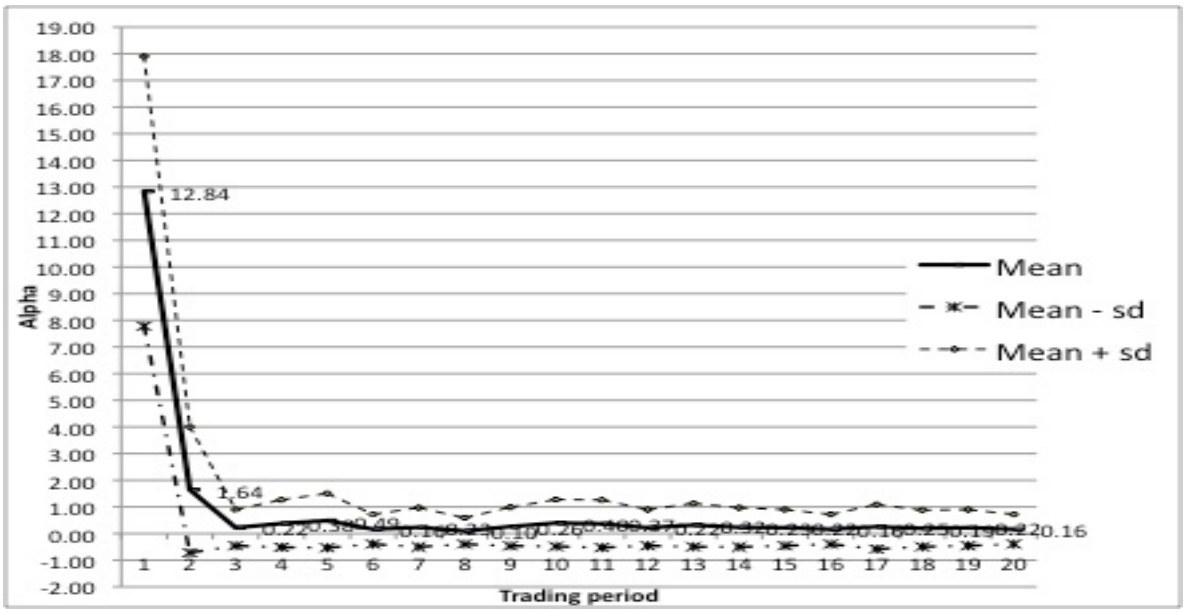

Figure 17: Alpha average over 50 MPS agents' experiments (20 trading periods per each experiment) with learning rate of 0.3; a random starting order and all agents has to wait for an open market time.

The graphs (figure 14 and figure 17) also show that the MPS agents are more stable and have better results than PS agents in that symmetric supply demand curve. The mean of alpha value of PS agents is about over 0.4% after second trading periods, while MPS agents' alpha means are about 0.2%. Moreover, the boundaries of alpha values of MPS agents are smaller than the PS agents ones. In consequence, it is clear show MPS traders did cover very close to the equilibrium price and stabilise slightly faster (0 to around 1%) than the other type of traders (0 to around 1.5%).

Again, figure 18 and figure 19 demonstrate that MPS strategy and PS strategy are robust in a flat supply demand curve. In general, the average of alpha values in the graphs falls quickly to around 0.16% before the first trading period and reach around 0% in the next days. The MPS traders have shown a stable results when all traders found the equilibrium price after one training days. However, the performance of PS traders are slightly different because in one of 50 their experiments, the alpha value was suddenly changed worse in one of the trading periods[12]. Therefore, MPS agents appear to be fitter one in the flat supply demand curve.

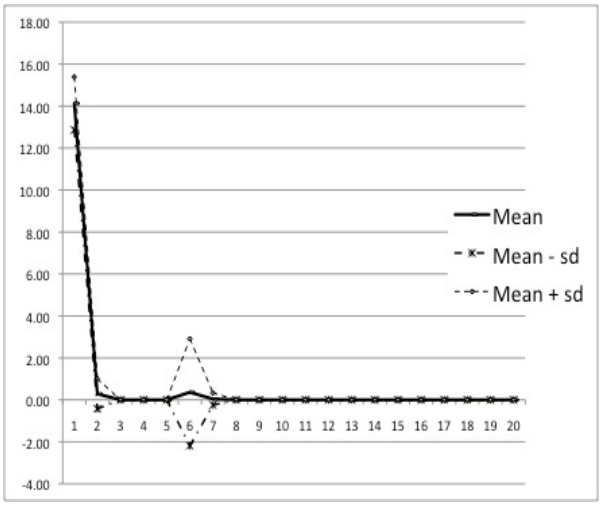

Figure 18: Alpha average over 50 PS agents' experiments (20 trading periods per each experiment) with learning rate of 0.3; a random starting order and all agents has to wait for an open market time. Flat supply and demand. (Supply 2.0)

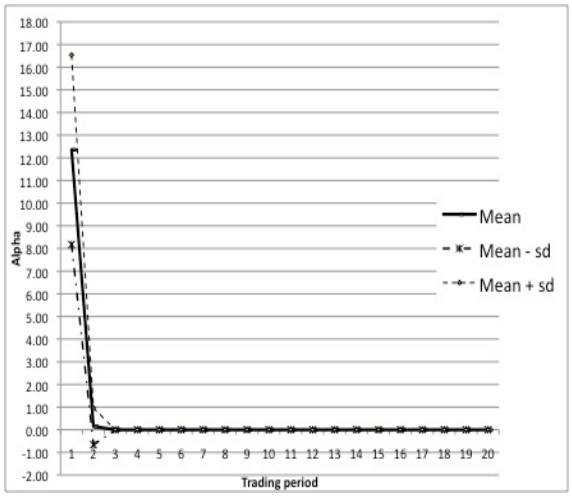

Figure 19: Alpha average over 50 MPS agents' experiments (20 trading periods per each experiment) with learning rate of 0.3; a random starting order and all agents has to wait for an open market time. Flat supply and demand. (Supply 2.0)

There is a drawback of MPS agents because they did not outperform in a simple flat supply demand curve with either excess supply or excess demand. All two kinds of agents started from a wide range of transaction price day 1 but the price

[12] It happens in different days when I retested PS strategy. Hence, it may be a problem of random parameter.

decrease gradually and then reach just about the theory equilibrium prices. But, figure 15 and figure 16 illustrate that PS agents only need 1 trading period to reach equilibrium prices, while MPS traders spend around 4 days to coverage the equilibrium prices (figure 18 and figure 19). As the results, the PS agents seem to perform better than MPS in overall although they loose in their favourite model.

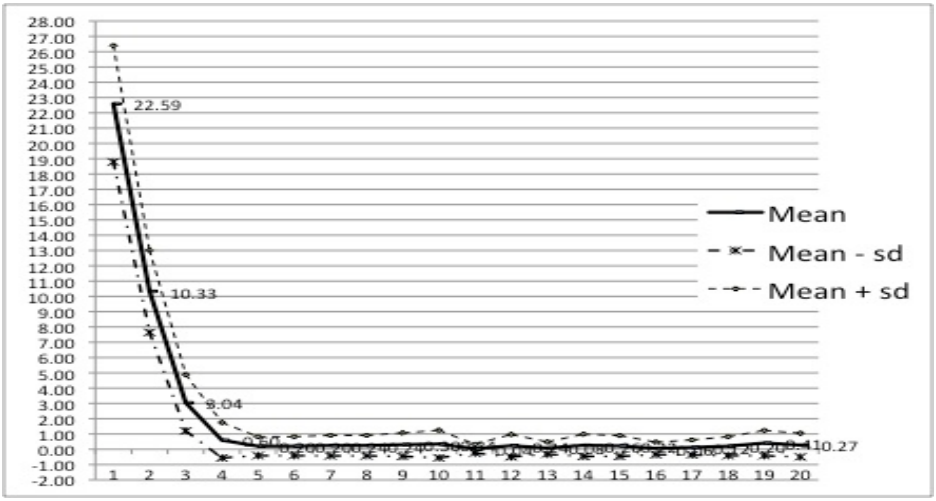

Figure 20: Alpha of average over 50 MPS agents' experiments (20 trading periods per each experiment) with learning rate of 0.3; a random starting order and all agents has to wait for an open market time. Flat supply and demand with excess supply. (Supply 2.0 Demand 3.2)

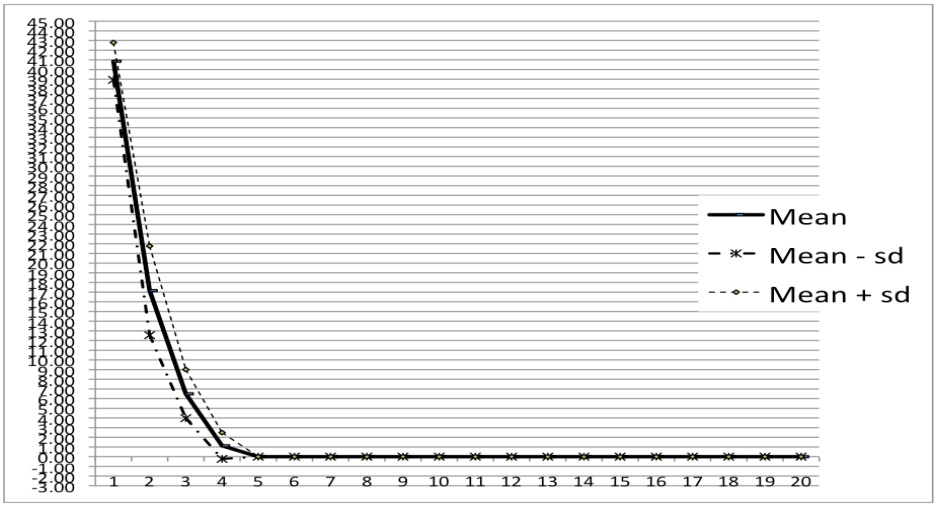

Figure 21:Alpha of average over 50 MPS agents' experiments (20 trading periods per each experiment) with learning rate of 0.3; a random starting order and all agents has to wait for an open market time. Flat supply and demand with excess demand. (Supply 0.5 Demand 2.0)

The following chapter will focus on how to select the best parameter using genetic algorithm.

VI. EVOLUTIONARY ALGORITHMS (EA)

As mentioned in the previous sections, strategies like ZIP, MPS and PS work by analysing information of past bids or ask. These algorithms have performed as good as and even better than a number of human traders in some of the experiments such as Das *et al.* (2001). In the real world, however, human traders are still dominant because their strategies are not only learning from experience, they are also learning from the other traders as well. Evolutionary Algorithms (EA) is a possible solution for the issue because it allows setting up an evolutionary process from which the best versions of a strategy could be selected.

In general, an EA firstly needs to initial a *population* of *individuals*, which are the trading agents in case of the auction game. The process involves random *selection*, *mutation* and *recombination* methods. The aim of selection process is to choose high-performing (high fitness-value) individuals such as the one that got the highest profit in auction game, at the end of each generation. In other words, the subjects adapted well to a new environment are the ones that have more chance to survive. Thus, the next generation includes the parents with high fitness value and their descendants who are created by the recombination mechanism. To generate other forces of the new population, the mutation process uses to explore the solution space to a new area. Hence, after enough fixed number of generations, the individuals are out performed in this type of the test environment.

VII. GENETIC ALGORITHM (GA)

In 1975, Holland introduced the theory of Genetic Algorithms (GAs), which is classified as an EA. The original concept in a GA is the probability of mating and survival of an individual depended on how well it is adapted to the environmental conditions.

According to Eshelman (Bäck et al., 2000), a typical GA will work as follows:

1. Initial the population of individuals;
2. Evaluate fitness for each individual in the population;
3. Repeat until one of the termination conditions have met.
    a. Select probabilistic individuals based on their fitness.
    b. Product a new mating buffer using crossover and mutation in current mating buffer.
    c. Evaluate fitness in each individual in new mating buffer.
    d. Select new population based on the new mating buffer and the past population.

The differences between GA and the other EAs are the representation component, the selection method based on probability proportional to fitness, and the crossover method. In fact, an object in aGA can represent in many different ways such as bit strings (in the original implement) or real-valued variables. The bit strings version is considered as quite a slow version because it needs to decode the bit strings in to real value every time before calculation the fitness. However, its mutation and crossover[13] tasks are fairly simple. For example, flipping randomly bits does a mutation. Eshelman (Bäck et al., 2000) also notes that using crossover helps covering a larger population, which could be much slower in the GA version with only mutation.

[13] Two bit strings exchanges their sub strings.

VIII. GENETIC ALGORITHM WITH ZIP

In order to explore the successful results of the original version of ZIP trader agents, Cliff (2001) decided to use GA to evolve the traders he had previously used to replicate results from the early set of experiments by Smith (1962). In Cliff's implementation, an individual ZIP trader is represented as a vector V of eight parameters μ, μ_Δ, β, β_Δ, γ, γ_Δ, ca and cr. The parameters β and β_Δ are used to calculate a learning rate by generating a uniform distribution (U) over range of β, $\beta + \beta_\Delta$ at random. The same explanation is applied for γ and γ_Δ to calculate a momentum value. However, $\mu(0)$ is calculated by the following equations:

$$\mu_h = \mu_b + \mu_\Delta$$

For seller: $\mu(0) = U(-\mu_h, -\mu_b)$

For buyer: $\mu(0) = U(\mu_b, \mu_h)$

In case of the original in the past experiences (Cliff and Bruten, 1997), the parameter was set as *Vcb* = [0.10, 0.40, 0.00, 0.10, 0.05, 0.30, 0.05, 0.05]. This vector will used to prove whether there are any improvements after the training or not. The other two vectors uses in his test are

Hard vector: $V_i \in$ [0.75, U_Δ, 0.75, U_Δ, 0.75, U_Δ, U_C, U_C]; where U_Δ = *U* (0.00, 0.25) and U_C = *U* (0.75, 1.00).

And the zero vector V_0 = [0, 0, 0, 0, 0, 0, 0, 0]; ∀*i*.

These bounds used for randomizing the initial population. Each of these vectors is the first generation (30 individuals) in Cliff's experiments. In fact, this way of setting initial generation could take more generation than using a basic random method to generate the population. For example, the best final individual, when he start the generation with 30 random hard vector, is
V = [0.361, 0.206, 0.000, 0.444, 0.191, 0.075, 0.000, 0.118] after 200 generation. This vector is far from the basic hard vectors and seems closer to those *Vcb.* However, the experiment with Vcb after the same amount of generations did not indicate that vector. Therefore, starting with all exactly individuals in one generation appears to be a wise choice.

The fitness function for an individual i is calculated by the equation bellow:

$$F(V_i) = \frac{1}{n}\sum_{e=1}^{n} S(V_i, e) = \frac{1}{n}\sum_{e=1}^{n}\frac{1}{6}\sum_{d=1}^{6} w_d \alpha(d)$$

Where w_d[14] is weigh of day d and $\alpha(d)$ is calculate by equation (1)

To evaluate to performance of each genome, Cliff (2001) used the market supply and demand curves shown in Figure 5. These parameter sets were tested 50 times each, and each test was a simulation of six trading days. As mentioned in the experimental setup session, this kind of settings shows how stable of the individuals in a fixed supply demand curve.

To receive improvements after each generation, elitism method, a standard method of selection is adopted. The concept of the method is that the top individuals in each generation preserve for the next generation. In his version of implementation, Cliff (2001) save the leading parameter vector for the next generation.

The parents are chosen from the simple tournament method between three individuals. It basically compares and takes two leading fitness as parents. For the reproduction process, the child will take the first parameter from his mom and the process continue with a random probability x which denote whether the child will continue take the next parameter of his mom or his father.

[14] Day 1 (1.75) Day 2 (1.50) Day 3 (1.25) Day 3 (1.0) Day 4 (1.0) Day 6 (1.0)

IX. GENETIC ALGORITHM WITH PS AGENTS

a. EXPERIMENT SETUP

To increase on the earlier performance/results by using an evolutionary process, there are several changes in the experimental setup and in GA operators consider with Cliff (2001). With these modifications, randomly generated 25-parameter vectors at the first generation are expected to quickly improve their performance in a few simulations of natural selection.

Firstly, the test experiments consist of a symmetric supply demand curve, which is a common supply demand taken from Cliff (1997). I intend to examine any market parameters ten times in the market model. The testing plan was expected to find some good set of parameters that can deal with the dynamic of the supply and demand curves and also stable in their performances. However, it could have the wrong result because the results are highly random in each of the parameter. In addition, the simulation will setup following the suggestion I have mentioned. Therefore, the booking order is always in stochastic states, where even the previous outperformed parameter (learning rate 0.3) sometime gives unstable results.

Secondly, considering the result from experiments of Cliff (2001), I see no problems to keep the fitness function and the GA operator standard. However, the initial generation is created by a uniform distributed method from [0,1]. In fact, these random parameter sets will have the form like

$$V_i = \left[\beta_i, U(0,1-\beta_i), \gamma_i, U(0,1-\gamma_i), \mu_i, U(0,1-\mu_i)\right]$$ [15]

The parameter vector $V_{core} = [0.7,0.0,0.05,0.0,0.3,0.0,0.02,0.02]$ is one of the vectors in the first generation in order to check the advantageous of evolutionary methods.

[15] U(x,y) is a uniform distribution in range of x to y

b. END RESULTS

Figure 22 demonstrates the results of evolutionary in 38 generations. The fitness averages are quite high around 11 and decrease slowly from the first generation to the fifth generation. In the next generation, the average result fall sharply to just about 6%. Then, the parameter sets of the next generations seem to continue the downtrend.

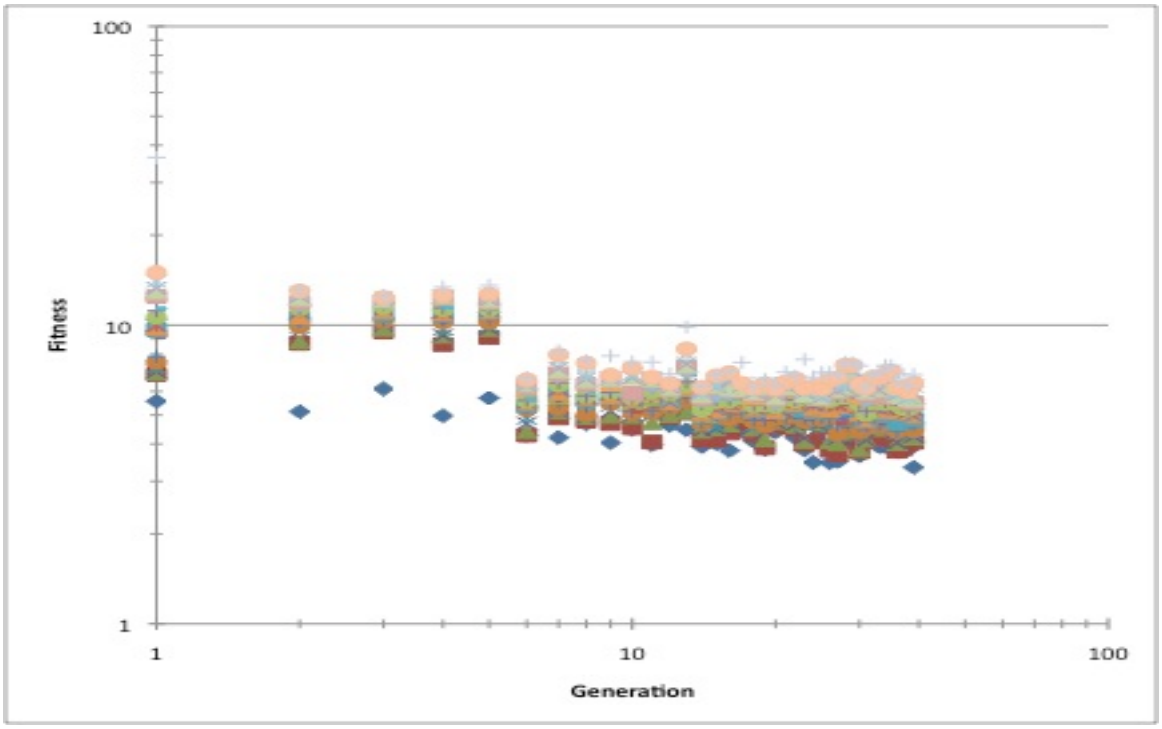

Figure 22: Log-log plot of the 25 individuals in the population for one experiment over 38 generations.

The best individual in the first generation scored 5.61% and the worse individual try to finish with around 11%. In general, the improvement in scores is significant, with more 50% in the average of the first and the 38th generation.

In the last 10 generations, the elites' scores are between 3.08% and 4%. Furthermore, the learning rate of these elites also covers 0.3 that is the one score the best results in last previous sections. In addition, the initial margins are in range of 0.1 to 0.6 for all the top individuals of the last generations.

In order to check improvements, the first elite and the last elite were performed in the same supply demand curve but they have more trading period to adjust

the equilibrium price. The tests also raise a question about stability of the top individuals in a longer trading days.

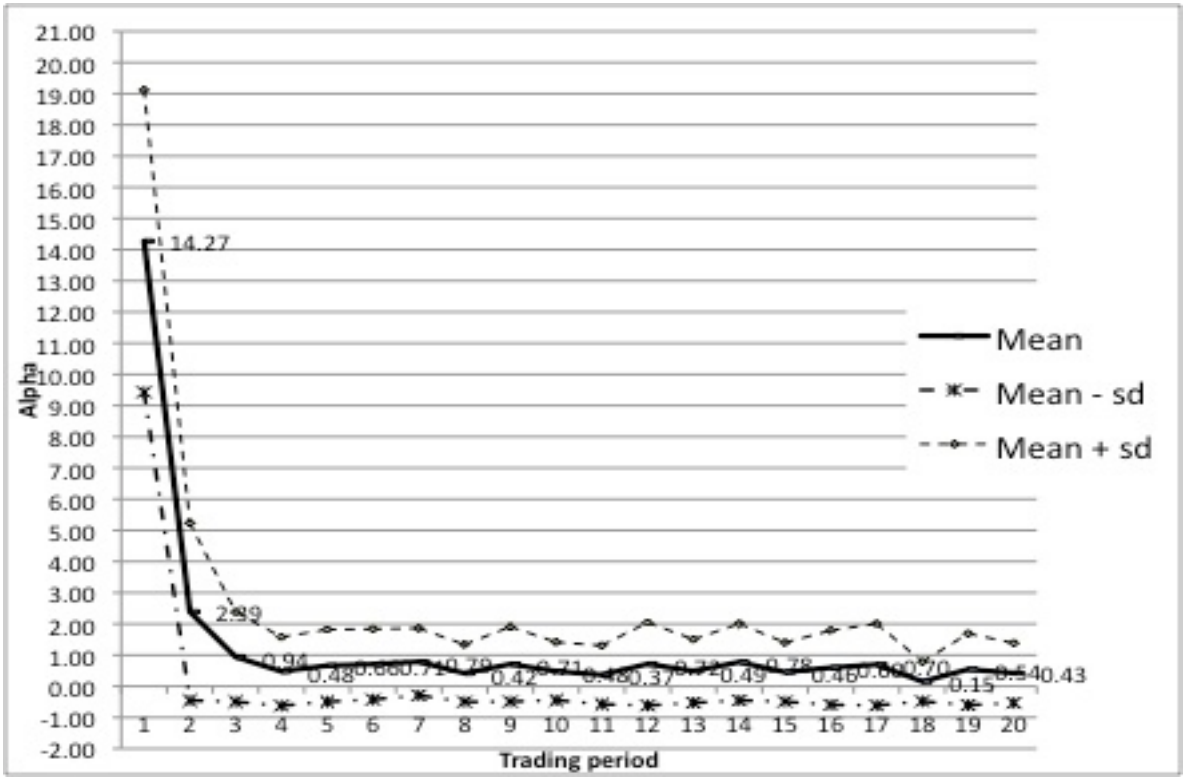

Figure 23:Alpha of average over 50 PS agents' experiments (20 trading periods per each experiment) with the elite of the first generation (μ = 0.14, μ_Δ =0.55, β = 0.28, β_Δ =0.37, γ = 0.18, γ_Δ = 0.3, ca = 0.01, cr = 0.06); a random starting order and all agents has to wait for an open market time.

Figure 23 and figure 24 show that during the first 6 periods, all the elite did quickly reach near the equilibrium price and the 38th elite seems to have the better results, which are just around 0.4% after the second days. The results of the first elite are stable and quite high at beginning periods. Hence, they loose against the last elites in term of fitness scoring. However, at the end of the experiments periods, the first one seems to have compatible results or even better one. Therefore, further work should consider a long period of trading rather than increasing the number of test in one curve like Cliff (2001) to solve the issue.

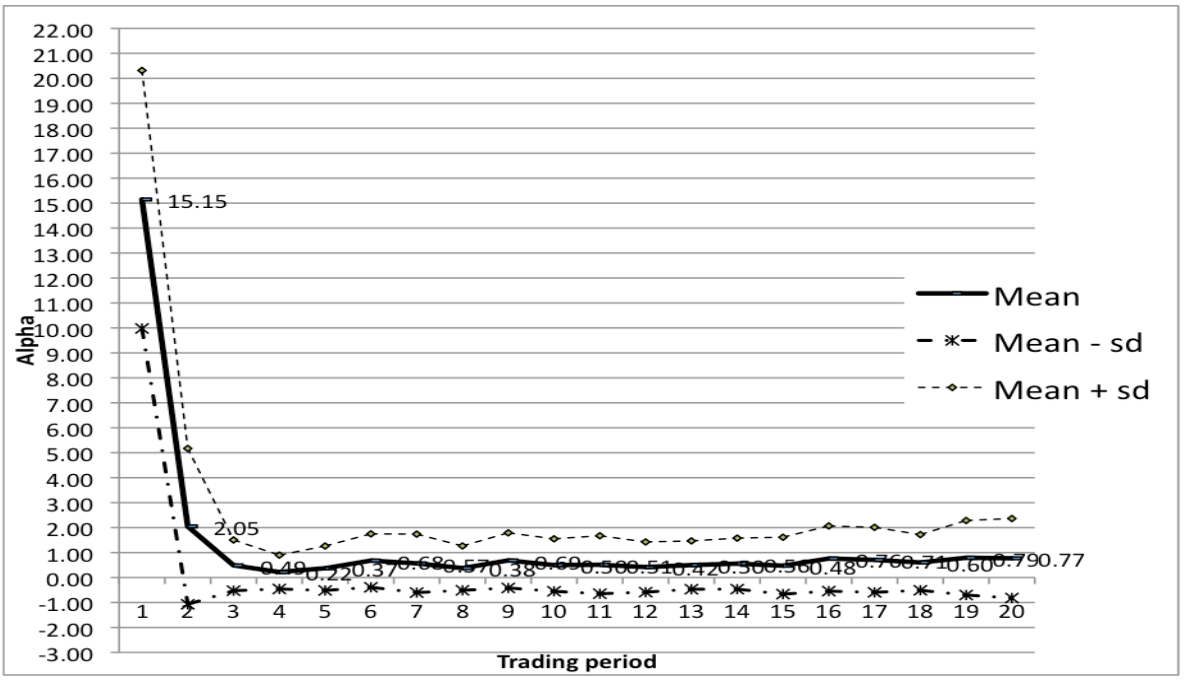

Figure 24:Alpha of average over 50 PS agents' experiments (20 trading periods per each experiment) with the elite of 38th generation ((μ = 0.06, μ_Δ =0.6, β = 0.23, β_Δ =0.32 , γ = 0.02, γ_Δ = 0.41, ca = 0.11 , cr = 0.01)); a random starting order and all agents has to wait for an open market time.

-

FURTHER WORK

As mention, a further work will carry out with the setting for the experiment in genetic algorithm. Increasing the number of individuals in population can give a wider range of solution space. Furthermore, it is necessary to have some changes in the fitness function in order to check the stability of any testing strategies. Basically, the longer of the trading time is the correctness of the experiment of resistance. In addition, any strategies should be tested against the random supply demand sets, so the best ones have more chance to apply in a real application.

The other methods such as genetic programming or swarm intelligent to improve normal algorithm, are considerable choice for my future plan. Moreover, I will try to explore a deeply knowledge about the order book in order to build a robust testing market for new strategies.

CONCLUSION

To sum up, the project has done successful works on evaluation and optimisation the performance of PS algorithm in persistent shout double auction. Firstly, the results in chapter "PROBLEMS WITH ORDER BOOK" show that PS algorithm with a new implementation performed well in a more realistic market model. Secondly, the modification of PS algorithm seems to be very robust in symmetric supply and demand curves and flat supply and demand curves.

Furthermore, in order to have a clear test results, I have introduced a new implementation of order book, which is very close to a real market model. In addition, because of the outperformed and stability of PS agents in the new order book, more research can carried out to about the algorithm.

Finally, although I have done a good implementation of GA for producing the best parameter sets as possible, the result seems to be limited by the trading time and number of population. Hence, to improve the performance of PS and MPS, the work with GA should modify as the suggestion on the "further work" section.

BIBLIOGRAPHY

Bäck, T., Fogel, D. B., Michalewicz Z., 2000. "Evolutionary Computation 1: Basic Algorithms and Operators," Institute of Physics Publishing.

Brewer, P.J., Huang, M., Nelson, B., Plott, C.R. (2002). "On the behavioral foundations of the law of supply and demand: Human convergence and robot randomness". Experimental Economics 5, pp 179–208.

Cliff, D. (2001) "Evolution of market mechanism through a continuous space of auction-types," Presented at the Artificial Societies and Computational Markets (ASCMA98) workshop at the Second International *27* Conference on Autonomous Agents, Minneapolis/St. Paul, May 1998. Also available as HP Labs Technical Report HPL-2001-99

Cliff, D. (2003) "Explorations in Evolutionary Design of Online Auction Market Mechanisms," Journal of Electronic Commerce Research and Applications, vol.2 (2), pp. 162-175.

Cliff, D. and Bruten, J., 1997. "Minimal-Intelligence Agents for Bargaining Behaviors in Market-Based Environments," Hewlett Packard Laboratories Paper HPL-97-91, Bristol, England.

Cliff, D. and Bruten, J., 1997a. "Zero is not enough: On the lower limit of agent intelligence for continuous double auction markets," Hewlett Packard Laboratories Paper 97-141, Bristol, England.

Cliff, D. and Bruten, J. 1997b. "More than zero intelligence needed for continuous double-auction trading", Hewlett Packard Laboratories Paper HPL-97-157, Bristol, England.

Duffy, J., 2006. "Agent-Based Models and Human Subject Experiments," Handbook of Computational Economics," in: Leigh Tesfatsion & Kenneth L. Judd (ed.), Handbook of Computational Economics, edition 1, volume 2, chapter 19, pp. 949-1011.

Gjerstad, S. and Dickhaut, J., 1998. "Price formation in double auctions," Games and Economic Behaviour, vol. 22, pp. 1-29.

Gode, D. and Sunder, S. (1993). "Allocative efficiency of market with zero-intelligent traders: Market as a partial substitute for individual rationality," Journal of Political Economy, vol. 101, pp. 119-137.

Hayek, F. A., 1945. "The Use of Knowledge in Society," American Economic Review, Vol. 35(4), p. 519-530.

He M., Leung H. -F. and Jennings N.R., 2003. "A fuzzy-logic based bidding strategy for autonomous agents in continuous double auctions," IEEE Transactions on Knowledge and Data Engineering 15 (6), pp. 1345–1363.

Herbert D., 2000. "On the emergence of exchange and mediation in a production economy," Journal of Economic Behavior & Organization, Elsevier, vol. 41(1), January , pages 27-53

Koza, J. R. and Poli, R., 2003. “A genetic programming tutorial,” In Burke, E., editor,Introductory Tutorials in Optimization, Search and Decision Support.

Mankiw, N., 2008. “Principles of microeconomics,” Cengage Learning, Inc, 5nd Eds, pp.62.

McAfee, R. P. and McMillan, J., 1987. "Auctions and Bidding,” Journal of Economic Literature, American Economic Association, vol. 25(2) June, pp. 701.

Phelps, S., Parsons, S., Sklar, E. and McBurney,P. , 2003. “Using genetic programming to optimise pricing rules for a double auction market,” In: Proceedings of the Workshop on Agents for Electronic Commerce, Pittsburgh.

Preist, C. and Van Tol, M., 1998. “Adaptive agents in a persistent shout double auction,” ICE’98: Proceedings of the First International Conference on Information and Computation Economies, ACM Press, New York, NY, USA (1998), pp. 11—18.

Posada, M., 2006. “Strategic Software Agents in Continuous Double Auction' under Dynamic Environments,” Lecture Notes in Computer Science 4224, pp. 1223-1233.

Posada, M., Hernández, C., and Lopez-Paredes, A., 2005. “Learning in a Continuous Double Auction Market,” Lecture Notes in Economics and Mathematical Systems vol.564, pp. 41-51.

Posada, M. and Lopez-Paredes A., 2007. "How to Choose the Bidding Strategy in Continuous Double Auctions: Imitation Versus Take-The-Best Heuristics," Journal of Artificial Societies and Social Simulation, Journal of Artificial Societies and Social Simulation, vol. 11.

Rust, J., Miller, J. and Palmer, R., 1993. “Behaviour of trading automata in computerized double auctions” in Friedman and Rust (eds.), The double auction markets: Institutions, theories and evidence Addison-Wesley, pp. 155-198.

Smith V. L., 1962. "An Experimental Study of Competitive Market Behavior,“ Journal of Political Economy, University of Chicago Press, vol. 70, No. 2., pp. 111-137.

Smith, Vernon L., Markets as Economizers of Information: Experimental Examination of the "Hayek Hypothesis", Economic Inquiry, 20:2 (1982:Apr.) p.165

Smith V. L and Williams A., 1983. “An experimental study of alternative rules for competitive market exchange”. In: R. Englebrecht-Wiggins, M.

Schubik and R. Stark, Editors, Auctions, bidding and contracting: Uses and theory, New York University Press, New York (1983), pp. 307–334.

Tesauro G. and Bredin J.L., 2002. "Strategic sequential bidding in auctions using dynamic programming," AAMAS'02: Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems, ACM Press, New York, NY, USA, pp. 591–598.

Tesauro, G., Das, R., 2001. "High performance bidding agents for the continuous double auction," Proceedings of the Third ACM Conference on Electronic Commerce, pp. 206-209.

Van Boening, M.V., Wilcox, N.T., 1996. "Avoidable cost: Ride a double auction roller coaster," American Economic Review vol.86 (3), pp. 461–477.

Appendix A: code

Due to the restriction , there are only two class codes, which also is not fully printed.

```
public class MBuyer extends ZIPTraders implements Runnable {

	private String buyerID;
	private boolean active = true;
	private double out_standing_ask = Common.OUT_STANDING_ASK;
	private double out_standing_bid = Common.OUT_STANDING_BID;
	private double profit = 0;
	private double limit_price = 0;
	private double transactionPrice = 0;
	private long openMarketTime = 0;

	public long getOpenMarketTime() {
		return openMarketTime;
	}

	public void setOpenMarketTime(long openMarketTime) {
		this.openMarketTime = openMarketTime;
	}

	public double getTransactionPrice() {
		return transactionPrice;
	}

	public void setTransactionPrice(double transactionPrice) {
		this.transactionPrice = transactionPrice;
	}

	public double getLimit_price() {
		return limit_price;
	}

	public void setLimit_price(double limitPrice) {
		limit_price = limitPrice;
	}

	private DoubleAuction doubleAuction;
	private ArrayList<Shout> historyShout;
	private Shout lastShout = null;

	public Shout getLastShout() {
		return lastShout;
	}

	public void setLastShout(Shout lastShout) {
		this.lastShout = lastShout;
	}

	private boolean endOfRound = false;

	// DecimalFormat df = new DecimalFormat();

	public boolean isEndOfRound() {
		return endOfRound;
	}

	public void setEndOfRound(boolean endOfRound) {
		this.endOfRound = endOfRound;
	}

	// private double
	Random r = new Random();
	double initMargin, learningRate, momentum, mMargin, bMargin, mLearningRate,
			bLearningRate, mMomentum, bMomentum, c_a, c_r;
	Thread t;
	double targetPrice = 0;
	double delta = 0;
	double wfValue = 0;
	double momentum_based = 0;
```

```
public double getProfit() {
        return profit;
}

public double getOut_standing_ask() {
        return out_standing_ask;
}

public void setOut_standing_ask(double outStandingAsk) {
        out_standing_ask = outStandingAsk;
}

public double getOut_standing_bid() {
        return out_standing_bid;
}

public void setOut_standing_bid(double outStandingBid) {
        out_standing_bid = outStandingBid;
}

public boolean isActive() {
        return active;
}

public void setActive(boolean active) {
        this.active = active;
}

public String getBuyerID() {
        return buyerID;
}

int numberOfInvalidShout = 0;

public void setBuyerID(String buyerID) {
        this.buyerID = buyerID;
}

boolean gotDeal = false;

public boolean isGotDeal() {
        return gotDeal;
}

public void setGotDeal(boolean gotDeal) {
        this.gotDeal = gotDeal;
}

double tmpMomentum;
double tmpLearningRate;
double cr;
double ca;

public MBuyer(String buyerID, double limit_price,
                DoubleAuction currentAuction, MarketParameters marketParameters) {
        this.buyerID = buyerID;
        this.limit_price = limit_price;
        doubleAuction = currentAuction;
        historyShout = new ArrayList<Shout>();
        this.mMargin = marketParameters.getmMargin();
        this.bMargin = marketParameters.getbMargin();
        this.mLearningRate = marketParameters.getmLearningRate();
        this.bLearningRate = marketParameters.getbLearningRate();
        this.mMomentum = marketParameters.getmMomentum();
        this.bMomentum = marketParameters.getbMomentum();
        this.c_a = marketParameters.getC_a();
        this.c_r = marketParameters.getC_r();
        initMargin = mMargin + r.nextFloat() * bMargin;
        learningRate = mLearningRate + r.nextDouble() * bLearningRate;

        momentum = mMomentum + r.nextFloat() * bMomentum;
```

```
		cr = r.nextDouble() * c_r;
		ca = r.nextDouble() * c_a;
	}

	public void start() {
		t = new Thread(this, buyerID);
		t.start();
	}

	public void reset() {
		numberOfInvalidShout = 0;
		setGotDeal(false);
		out_standing_ask = Common.OUT_STANDING_ASK;
		out_standing_bid = Common.OUT_STANDING_BID;
	}

	/*
	 * (non-Javadoc)
	 *
	 * @see java.lang.Runnable#run()
	 */
	int numberOfRound = 0;
	double initShoutValue = 0;
	boolean accepted = false;

	@Override
	public void run() {
		try {
			while (doubleAuction.isWait()) {
				synchronized (doubleAuction.look) {
					try {
						doubleAuction.getLook().wait(1);
					} catch (InterruptedException e) {
						e.printStackTrace();
					}
				}

			}
			if (!isActive()) {
				while (!doubleAuction.isEndOfRound()
						&& !doubleAuction.isEndOfTradingDay()) {
					if (out_standing_bid >= limit_price) {
						break;
					}
					double suggestPrice = notActiveTraining();
					if (out_standing_bid >= limit_price
							|| suggestPrice == limit_price) {
						break;
					}
					lastShout.setShoutValue(suggestPrice);
					Thread.sleep(10);
				}
			} else {
				ArrayList statusList = null;
				Shout newShout;

				if (doubleAuction.getTradingDay() < 1) {
					if (doubleAuction.getRound() < 1 || lastShout == null) {
						initShoutValue = new Util().getRightFormat(limit_price
								* (1 - initMargin));
						newShout = new Shout(Common.BID, initShoutValue);
					} else {
						newShout = new Shout(Common.BID, ZIPStrategy());
					}

				} else {
					if (doubleAuction.getRound() < 1) {
						if (isGotDeal()) {
							newShout = new Shout(Common.BID, ZIPStrategy());
						} else {
							if (initShoutValue < limit_price) {
								initShoutValue = new Util().getRightFormat(
```

```
                                                                (1 + r.nextDouble() * 0.2)
                                                                                *
lastShout.getShoutValue()
                                                                                +
r.nextDouble() * 0.2,
                                                                RoundingMode.UP);
                                                        if (initShoutValue > limit_price) {
                                                                initShoutValue = limit_price;
                                                        }
                                                } else {
                                                        initShoutValue = limit_price;
                                                }
                                                newShout = new Shout(Common.BID, initShoutValue);
                                        }
                                } else {
                                        newShout = new Shout(Common.BID, ZIPStrategy());
                                }
                        }
                        statusList = doubleAuction.receiveShout(this, newShout);
                        do {

                                out_standing_ask = Double.parseDouble(statusList.get(2)
                                                .toString());
                                out_standing_bid = Double.parseDouble(statusList.get(1)
                                                .toString());
                                if (statusList.get(0).equals(Common.INVALID_SHOUT)) {
                                        newShout.setShoutType(Common.INVALID_SHOUT);
                                        lastShout = newShout;
                                        historyShout.add(newShout);
                                        setGotDeal(false);
                                } else if (statusList.get(0).equals(Common.BID)) {
                                        newShout.setShoutType(Common.BID);
                                        lastShout = newShout;
                                        historyShout.add(newShout);
                                        setGotDeal(false);
                                } else if (statusList.get(0).equals(Common.DEAL)) {
                                        newShout.setShoutType(Common.DEAL);
                                        lastShout = newShout;
                                        historyShout.add(newShout);
                                        setEndOfRound(true);
                                        setGotDeal(true);
                                        break;
                                } else {
                                        if (isActive()) {
                                                lastShout = newShout;
                                                historyShout.add(newShout);
                                        }
                                        setEndOfRound(true);
                                        break;
                                }
                                if (out_standing_bid >= limit_price
                                                || newShout.getShoutValue() == limit_price) {
                                        break;
                                }
                                Thread.sleep(10);
                                if (isActive() || !isGotDeal()) {
                                        newShout.setShoutValue(ZIPStrategy());
                                } else {
                                        setGotDeal(true);
                                        break;
                                }

                                statusList = doubleAuction.receiveShout(this, newShout);
                        } while (!isEndOfRound() || isActive() || !isGotDeal());
                        numberOfRound++;
                        if (isActive()) {
                                // reset();
                        } else {
                                setGotDeal(true);
                        }
                }
        } catch (InterruptedException e) {
```

```
                e.printStackTrace();
            }
        }

        public void updateProfit() {
            profit = new Util().getRightFormat(limit_price - transactionPrice);
        }

        double previousShout = 0;

        public double notActiveTraining() {
            return ZIPStrategy();
        }

        /*
         * (non-Javadoc)
         *
         * @see ZIPTraders#ZIPStrategy()
         */
        @Override
        public double ZIPStrategy() {
            RoundingMode rd;
            if (doubleAuction.getOut_standing_bid() != Common.OUT_STANDING_BID
                    && doubleAuction.getOut_standing_ask() != Common.OUT_STANDING_ASK) {
                out_standing_bid = doubleAuction.getOut_standing_bid();
                out_standing_ask = doubleAuction.getOut_standing_ask();
            }
            if (out_standing_ask <= out_standing_bid) {
                targetPrice = (1 - r.nextDouble() * c_r) * out_standing_ask
                        - r.nextDouble() * c_a;
                rd = RoundingMode.DOWN;
            } else {
                targetPrice = (1 + r.nextDouble() * c_r) * out_standing_bid
                        + r.nextDouble() * c_a;
                rd = RoundingMode.UP;
            }
            targetPrice = new Util().getRightFormat(targetPrice, rd);
            // targetPrice = new Util().getRightFormat(targetPrice);
            // Widrow-Hoff delta value
            wfValue = (learningRate * (new Util().getRightFormat(targetPrice
                    - lastShout.getShoutValue())));
            // BP network
            momentum_based = momentum * momentum_based + (1 - momentum) * wfValue;

            // M_i(t+1) = (p_i(t) + delta_i(t))/limit_price - 1
            double profitMargin = new Util().getRightFormat((lastShout
                    .getShoutValue() + momentum_based)
                    / limit_price, rd) - 1;
            double shoutPrice = new Util().getRightFormat(limit_price
                    * (1 + profitMargin), rd);
            if (shoutPrice > limit_price) {
                return limit_price;
            }
            return shoutPrice;
        }

        @Override
        public void addPrivateValueList(ArrayList<Double> privateValues) {

        }
}
```

```
public class DoubleAuction {
	DecimalFormat df = new DecimalFormat();
	private int num_seller = 11;
	private int num_buyer = 11;
	private double out_standing_ask = Common.OUT_STANDING_ASK;
	private double out_standing_bid = Common.OUT_STANDING_BID;
	private ArrayList<Buyer> buyerList = new ArrayList<Buyer>();
	private ArrayList<Seller> sellerList = new ArrayList<Seller>();
	private ArrayList<Buyer> notActiveBuyerList = new ArrayList<Buyer>();
	private ArrayList<Seller> notActiveSellerList = new ArrayList<Seller>();
	public ArrayList<Double> buyer_PVL = new ArrayList<Double>();
	public ArrayList<Double> seller_PVL = new ArrayList<Double>();
	private double surplus = 0;
	Buyer buyer_hold_osb;
	MarketParameters mp;
	Seller seller_hold_osa;
	Timer timer;
	DayRecord dayRecord = null;
	public final static String DAY_RECORD = "dayrecord";
	private boolean endOfTradingDay = false;

	double mMargin, bMargin, mLearningRate, bLearningRate, mMomentum,
			bMomentum, c_a, c_r;
	private double numberOfDeal = 0;

	/**
	 * @param args
	 */
	public void wait(int seconds) {
		new TimeControl(seconds, this);
	}

	public boolean isEndOfTradingDay() {
		return endOfTradingDay;
	}

	public void setEndOfTradingDay(boolean endOfTradingDay) {
		this.endOfTradingDay = endOfTradingDay;
	}

	public double getOut_standing_ask() {
		return out_standing_ask;
	}

	public void setOut_standing_ask(double outStandingAsk) {
		out_standing_ask = outStandingAsk;
	}

	public double getOut_standing_bid() {
		return out_standing_bid;
	}

	public void setOut_standing_bid(double outStandingBid) {
		out_standing_bid = outStandingBid;
	}

	public double getNumberOfDeal() {
		return numberOfDeal;
	}

	public void setNumberOfDeal(double numberOfDeal) {
		this.numberOfDeal = numberOfDeal;
	}

	public DoubleAuction() {
		df.setRoundingMode(RoundingMode.UP);
		df.setMaximumFractionDigits(2);
	}

	public void marketInit(MarketParameters marketParameters) {
		this.mp = marketParameters;
		this.mMargin = marketParameters.getmMargin();
```

```
		this.bMargin = marketParameters.getbMargin();
		this.mLearningRate = marketParameters.getmLearningRate();
		this.bLearningRate = marketParameters.getbLearningRate();
		this.mMomentum = marketParameters.getmMomentum();
		this.bMomentum = marketParameters.getbMomentum();
		this.c_a = marketParameters.getC_a();
		this.c_r = marketParameters.getC_r();
	}

	public MarketParameters getMp() {
		return mp;
	}

	public void setMp(MarketParameters mp) {
		this.mp = mp;
	}

	boolean endOfRound = false;

	public boolean isEndOfRound() {
		return endOfRound;
	}

	public void setEndOfRound(boolean endOfRound) {
		this.endOfRound = endOfRound;
	}

	boolean bg = false;

	@SuppressWarnings("unchecked")
	public ArrayList processBuyerMessage(Buyer buyer, Shout newShout) {
		ArrayList list = new ArrayList();
		if (newShout.getShoutValue() <= out_standing_bid) {
			list.add(Common.INVALID_SHOUT);
			list.add(out_standing_bid);
			list.add(out_standing_ask);
			return list;
		} else if (out_standing_bid < newShout.getShoutValue()
				&& newShout.getShoutValue() < out_standing_ask) {
			out_standing_bid = newShout.getShoutValue();
			buyer_hold_osb = buyer;
			list.add(Common.BID);
			list.add(out_standing_bid);
			list.add(out_standing_ask);
			return list;
		} else {
			out_standing_bid = newShout.getShoutValue();
			buyer_hold_osb = buyer;
			buyer.setActive(false);
			Seller seller = seller_hold_osa;
			double tradeValue = 0;
			tradeValue = new Util().getRightFormat(tradeValue);
			buyer.setTransactionPrice(tradeValue);
			seller.setTransactionPrice(tradeValue);
			buyer.updateProfit();
			seller.updateProfit();
			seller.setActive(false);
			seller.setOut_standing_bid(out_standing_bid);
			seller.setGotDeal(true);
			buyer.setGotDeal(true);
			setEndOfRound(true);
			numberOfDeal++;
			hadDeal = true;
			notActiveBuyerList.add(buyer);
			notActiveSellerList.add(seller);
			list.add(Common.DEAL);
			list.add(out_standing_bid);
			list.add(out_standing_ask);
			dayRecord.addDealValue(tradeValue, buyer.getBuyerID(), seller
					.getSellerID());
			return list;
		}
```

```
}

@SuppressWarnings("unchecked")
public ArrayList processSellerMessage(Seller seller, Shout newShout) {
        ArrayList list = new ArrayList();
        if (newShout.getShoutValue() >= out_standing_ask) {
                list.add(Common.INVALID_SHOUT);
                list.add(out_standing_bid);
                list.add(out_standing_ask);
                return list;
        } else if (out_standing_bid < newShout.getShoutValue()
                        && newShout.getShoutValue() < out_standing_ask) {
                out_standing_ask = newShout.getShoutValue();
                seller_hold_osa = seller;
                list.add(Common.ASK);
                list.add(out_standing_bid);
                list.add(out_standing_ask);
                return list;
        } else {
                seller.setActive(false);
                out_standing_ask = newShout.getShoutValue();
                seller_hold_osa = seller;
                Buyer buyer = buyer_hold_osb;
                double tradeValue = 0;
                try {
                        tradeValue = df.parse(
                                        df.format((out_standing_ask + out_standing_bid) / 2))
                                        .doubleValue();
                } catch (ParseException e) {
                        e.printStackTrace();
                        System.out.println("wrong number" + tradeValue);
                }
                buyer.setTransactionPrice(tradeValue);
                buyer.updateProfit();
                buyer.setOut_standing_ask(out_standing_ask);
                seller.setTransactionPrice(tradeValue);
                seller.updateProfit();
                buyer.setActive(false);
                seller.setGotDeal(true);
                buyer.setGotDeal(true);
                hadDeal = true;
                list.add(Common.DEAL);
                list.add(out_standing_bid);
                list.add(out_standing_ask);
                notActiveBuyerList.add(buyer);
                notActiveSellerList.add(seller);
                dayRecord.addDealValue(tradeValue, buyer.getBuyerID(), seller
                                .getSellerID());
                numberOfDeal++;
                setEndOfRound(true);
                return list;
        }

}

@SuppressWarnings("unchecked")
public synchronized ArrayList receiveShout(ZIPTraders trader, Shout newShout) {
        ArrayList list;
        if (isEndOfRound()) {
                list = new ArrayList();
                list.add(Common.WAIT_NEXT_ROUND);
                list.add(out_standing_bid);
                list.add(out_standing_ask);
                return list;
        }
        if (isEndOfTradingDay()) {
                list = new ArrayList();
                list.add(Common.END_OF_TRADING_DAY);
                list.add(out_standing_bid);
                list.add(out_standing_ask);
                return list;
```

```
		}
		if (trader instanceof Buyer) {
			Buyer buyer = (Buyer) trader;
			ArrayList status = processBuyerMessage(buyer, newShout);
			return status;
		} else {
			Seller seller = (Seller) trader;
			ArrayList status = processSellerMessage(seller, newShout);
			return status;
		}

	}

	boolean hadDeal = false;

	private void reset() {
		out_standing_ask = Common.OUT_STANDING_ASK;
		out_standing_bid = Common.OUT_STANDING_BID;
		setEndOfRound(false);
		buyer_hold_osb = null;
		seller_hold_osa = null;
		round = 0;
		wait = true;
		hadDeal = false;
	}

	public void generateAgentTables() {
		for (int i = 0; i < num_buyer; i++) {
			buyerList.add(new Buyer("buyer " + i, buyer_PVL.get(i), this, mp));
		}
		for (int i = 0; i < num_seller; i++) {
			sellerList.add(new Seller("seller " + i, seller_PVL.get(i), this,
					mp));
		}

	}

	double alpha = 0;
	// double equilibriumPriceModel1 = 3.0;
	double maxSurplus = 0;
	DecimalFormat dataFormat = new DecimalFormat();
	String mode = "";
	EquiliriumPrice equilibrium = null;

	public EquiliriumPrice getEquilibrium() {
		return equilibrium;
	}

	public void setEquilibrium(EquiliriumPrice equilibrium) {
		this.equilibrium = equilibrium;
	}

	public void setEquilibrium() {
		this.equilibrium = new Util().getEquilirium(buyer_PVL, seller_PVL);
	}

	double avgTransaction = 0;

	/**
	 * calculate Smith's alpha value
	 */
	public void calculateCoefficientConvergence() {
		double tmp = 0;
		double transPriceTmp = 0;
		double numberOfDeal = 0;
		// double surplus = 0;
		dataFormat.setMaximumFractionDigits(2);
		dataFormat.setRoundingMode(RoundingMode.FLOOR);
		double equilibriumPrice = getEquilibrium().getEquiliriumPrice();
		double tunnelPrice = getEquilibrium().getTunnelWide();
		maxSurplus = getEquilibrium().getMaxSurplus();
		double diffPrice = 0;
```

```
		int i = 0;
		for (Buyer buyer : notActiveBuyerList) {
			transPriceTmp = buyer.getTransactionPrice();
			avgTransaction = avgTransaction + transPriceTmp;
			i++;
			if (transPriceTmp > 0) {
				surplus = surplus + buyer.getProfit();
				numberOfDeal++;
				if (transPriceTmp <= (equilibriumPrice + tunnelPrice)
						&& transPriceTmp >= equilibriumPrice) {
					diffPrice = 0;
				} else if (transPriceTmp < equilibriumPrice) {
					diffPrice = Math.abs(transPriceTmp - equilibriumPrice);
				} else {
					diffPrice = Math.abs(transPriceTmp - equilibriumPrice
							- tunnelPrice);
				}

				diffPrice = new Util().getRightFormat(diffPrice);

				double diffPrice2 = diffPrice * diffPrice;
				try {
					diffPrice2 = dataFormat

.parse(dataFormat.format(diffPrice2)).doubleValue();
				} catch (ParseException e) {
					e.printStackTrace();
					// System.out.println("wrong number" + nfmn);
				}

				tmp = new Util().getRightFormat(tmp + diffPrice2);
			}

		}
		for (Seller seller : notActiveSellerList) {
			transPriceTmp = seller.getTransactionPrice();
			if (transPriceTmp > 0) {
				surplus = surplus + seller.getProfit();
			}

		}
		avgTransaction = new Util().getRightFormat(avgTransaction / i);
		double sigma = Math.sqrt(tmp / numberOfDeal);
		alpha = new Util().getRightFormat((100 * sigma) / equilibriumPrice);
		System.out.println("alpha " + alpha);
	}

	/**
	 * reset the market after one trading day
	 */
	public void resetAgentTables() {
		for (Buyer buyer : buyerList) {
			buyer.setActive(true);
			buyer.setTransactionPrice(0);

		}
		for (Seller seller : sellerList) {
			seller.setActive(true);
			seller.setTransactionPrice(0);
		}
		notActiveBuyerList.clear();
		notActiveSellerList.clear();
		surplus = 0;
		maxSurplus = 0;
		round = 0;
		tradingDay++;
		numberOfDeal = 0;
		wait = true;
		avgTransaction = 0;
	}
	int round = 0;
```

```
public int getRound() {
	return round;
}

double tradingDay = 0;

/**
 * randomise order book
 */
private void randomJoinTheMarket() {
	ArrayList<Buyer> inBuyerTheList = new ArrayList<Buyer>();
	ArrayList<Seller> inSellerTheList = new ArrayList<Seller>();
	int remainBuyers = buyerList.size();
	int remainSellers = sellerList.size();
	int minRemainTraders = remainBuyers < remainSellers ? remainBuyers
			: remainSellers;
	for (int i = 0; i < minRemainTraders; i++) {
		Buyer buyer = new Util().getRandomBuyer(buyerList,
				notActiveBuyerList, inBuyerTheList);
		inBuyerTheList.add(buyer);
		buyer.start();
		Seller seller = new Util().getRandomSeller(sellerList,
				notActiveSellerList, inSellerTheList);
		inSellerTheList.add(seller);
		seller.start();
	}
	if (remainBuyers > minRemainTraders) {
		for (int i = 0; i < remainBuyers - minRemainTraders; i++) {
			Buyer buyer = new Util().getRandomBuyer(buyerList,
					notActiveBuyerList, inBuyerTheList);
			inBuyerTheList.add(buyer);
			buyer.start();
		}
	} else if (remainSellers > minRemainTraders) {
		for (int i = 0; i < remainSellers - minRemainTraders; i++) {
			Seller seller = new Util().getRandomSeller(sellerList,
					notActiveSellerList, inSellerTheList);
			inSellerTheList.add(seller);
			seller.start();
		}
	}
	setWait(false);
	synchronized (look) {
		look.notifyAll();
	}

}
boolean wait = true;
public synchronized boolean isWait() {
	return wait;
}

public synchronized void setWait(boolean wait) {
	this.wait = wait;
}

public Object look = new Object();

public Object getLook() {
	return look;
}

public void setLook(Object look) {
	this.look = look;
}

private void InOrderJoinTheMarket() {
	for (Buyer buyer : buyerList) {
		if (buyer.isActive()) {
			buyer.start();
		} else {
			// buyer.start();
```

```
                }
        }
        for (int i = 0; i < sellerList.size(); i++) {
                if (sellerList.get(i).isActive()) {
                        sellerList.get(i).start();
                } else {
                        // sellerList.get(i).start();
                }
        }
        setWait(false);
        synchronized (look) {
                look.notifyAll();// notify all thread
        }
}

/**
 * create and run a trading day
 */
public void simulateTradingDay() {
        setEndOfTradingDay(false);
        TimeControl timeControl = new TimeControl(Common.MARKET_OPEN_TIME, this);

        while (!isEndOfTradingDay()) {
                setEndOfRound(false);
                wait = true;
                randomJoinTheMarket();
                try {
                        for (Buyer buyer : buyerList) {
                                buyer.t.join();
                        }
                        for (Seller seller : sellerList) {
                                seller.t.join();
                        }
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
                if (!hadDeal) {
                        setEndOfTradingDay(true);
                }
                reset();
                round++;
        }
        calculateCoefficientConvergence();
}

public double getTradingDay() {
        return tradingDay;
}

public void setTradingDay(double tradingDay) {
        this.tradingDay = tradingDay;
}

/**
 * do one experiment in fix number of days
 */
public ArrayList<DayRecord> experiment() {
        ArrayList<DayRecord> experimentRecord = new ArrayList<DayRecord>();
        for (int i = 0; i < Common.NUM_OF_TRADING_DAY; i++) {
                dayRecord = new DayRecord(mp, buyer_PVL, seller_PVL);
                simulateTradingDay();
                System.out.println("Finish trading day " + i);
                dayRecord.setActualSurplus(surplus);
                dayRecord.setMaxSurplus(maxSurplus);
                dayRecord.setAlphaSmith(alpha);
                dayRecord.setAllocativeEfficiency(new Util().getRightFormat(surplus
                                / maxSurplus));
                dayRecord.summaryTradingDay();
                dayRecord.setAvgTransaction(avgTransaction);
                experimentRecord.add(dayRecord);
                resetAgentTables();
        }
```

```
                System.gc();
                return experimentRecord;
        }
}
```