Analysis of robot traders within the framework of the Bristol Stock Exchange

Adam Coales (39683)[1] and Ved Uttamchandani (36811)[2]

University of Bristol
[1]ac0541@bristol.ac.uk
[2]vu0232@bristol.ac.uk

Abstract. In the present paper, we explore the differences between various (eleven) robot traders, some of which have been implemented by external sources whilst the rest are either our own creations, or variations upon existing implementations. We found that our bespoke trader (CSNP) outperforms all the other traders it has been pitted against with the majority of supply and demand schedules.

1 The traders employed

1.1 Giveaway (GVWY)

The Giveaway trader is a straight-forward, naive trader which submits orders at the limit price it is provided with. This means that, although it will never incur a loss, its profits are incidental, a mere by-product of current market conditions.

1.2 Zero Intelligence Constrained (ZIC)

ZIC is a trivial trader with, as its name suggests, a limited quantity of logic: it picks a random number between the limit order price and the worst possible bid/ask in the limit order book (LOB). More often than not, its bididing strategies are sub-optimal, with bids laying well under the current market equilibrium.

1.3 Shaver (SHVR)

Shaver improves on current offers by shaving one off of the best ask or adding one onto the best bid in order to only minimally reduce its potential profit whilst outbidding its competitors.

1.4 Sniper (SNPR)

Sniper is a variation of Shaver. As opposed to Shaver, this trader lies dormant for the first 80% of the trading day after which it "shaves" the best bid/ask by increasing amounts. This trader would be more profitable in an environment where he could accumulate orders as he receives them. In this environment, all the orders received before his active phase are discarded.

1.5 Zero Intelligence Plus (ZIP)

ZIP's offers are based around the limit order price scaled by some margin. This margin is updated regularly based upon the state of the current market via the LOB and the history of trades.

ZIP's offers are based around the limit price scaled by some margin. This margin is constantly kept up to date within the respond function. Respond monitors the state of the current market, using the limit order book (lob) and history of trades to determine how the current margin should be modified.

1.6 Match (MTCH)

This trader matches the current best offers. The behaviour implemented is an attempt to reflect a social heuristic which is not uncommon: herd behaviour. The purpose of this quote is to highlight matches predisposition to rely on alien intelligence (other traders) in order to determine its own course of action. In the context of stock markets, this behaviour is most evident when stock market bubbles occur: Trends in large stock markets tend to begin and end with bubbles or crashes. These tend to be episodes where the "herd" appears to be greedy or afraid for no rational reason. "Insanity in individuals is rare - but in groups, parties, nations, and epochs, it is the rule." - Nietzsche.

1.7 Random (RAND)

Random is based upon ZIC but, unlike it, this trader constrains its random numbers between the limit order price and the best bid/ask if they are available. In the case where they are not, it falls back to ZIC's default behaviour.

1.8 Adaptive Aggressive (AA)

The AA trader makes use of a bidding strategy which enables it to respond efficiently and quickly to fluctuations in the market but also to longer-term trends in the market. It does this by way of two distinct learning behaviours: the first, the short-term one, modifies a factor which reflect how aggressive it is and the second, the longer-term one, observes the influence of the factor on the trader's behaviour and the current market trend. The aggression factor controls how much the trader is willing to sacrifice profit for quantity of trades: the more aggressive it is, the more it is willing to lower its profit margin and accept worse offers.

1.9 Profit Away (PFWY)

An improvement on Giveaway, Profit Away generates offers which give it a 5% profit margin. In essence, this trader sacrifices on the number of trades with the aim to increase profits by consequential amounts through variations in market equilibrium.

1.10 Middle (MIDL)

This trader offers prices halfway between the best bid and the best ask. Although it never accepts offers itself, it endeavours to entice traders to accept its by promptly skipping ahead of the competition.

1.11 Custom Sniper (CSNP)

The main and final trader we worked on. The strategy revolves around secrecy: it limits its footprint in the LOB by simply accepting deals it regards as profitable enough. It computes its profit margin by analysing the standard deviation of the market during its recent history when the new order comes in. The intuition is that the standard deviation is a representation of the fluctuations in the market, however, to limit the ambition on the trader, we set an upper bound on the profit margin percentage. The larger it is, the more room there is for variation in prices and therefore, potentially better deals. In addition to this behaviour, it also drastically lowers its profit margin during what it estimates to be the last 10% of time before a new order will come in (calculated by taking the average of the time between all previous orders).It is worth noting that, given its limited history memory, CSNP looks at local variations in the market.

2 Statistical Analysis

2.1 Testing consistency and performance of all traders

Experiment In this experiment, we tested the performance of all the traders running simultaneously in an increasingly unstable market. The market varies in a sinusoidal fashion with a reducing wavelength and increasing amplitude, and a constant range of ten (spread in the y-axis). The market settings had an order schedule range between 95-105 with schedule_offsetfn and fixed step mode. Customer intervals set to 10s interval with a drip-poisson time mode.

Results As is illustrated by Figure 1 the traders can be broadly split into four categories with respect to their profit margins. The first category comprises of ZIC and SNPR. The second of MTCH, AA and MIDL. GVWY, RAND, SHVR and ZIP make up the third, whilst PFWY and CSNP make up the final one, that of those with the highest income.
A two-tailed U-Test, at a significance level of 0.05, between CSNP and PFWY gives the following results:
U-value=1142
p=0.4413
p is greater than 0.05, therefore neither the data sets obtained by running CSNP or PFWY is stochastically greater than the other.
On the other hand, a U-Test with the same parameters between CSNP and ZIP gives the following results:

U-value=2500
p=0
p is less than 0.05, therefore the data sets obtained by running CSNP and ZIP are significantly different. Using these results along with the means of the analysed data sets (CSNP_mean=389.49, PFWY_mean=390.58, ZIP_mean=327.99), we can infer that, CSNP has performed better than ZIP. We can not, however, draw any conclusions with respect to the difference in performance between CSNP and PFWY.
We observed that the top tier traders' profit gradients marginally increased as the market became more unstable as each session progressed, reflecting the increased chances of better deals occurring as well as their design to look for better deals.

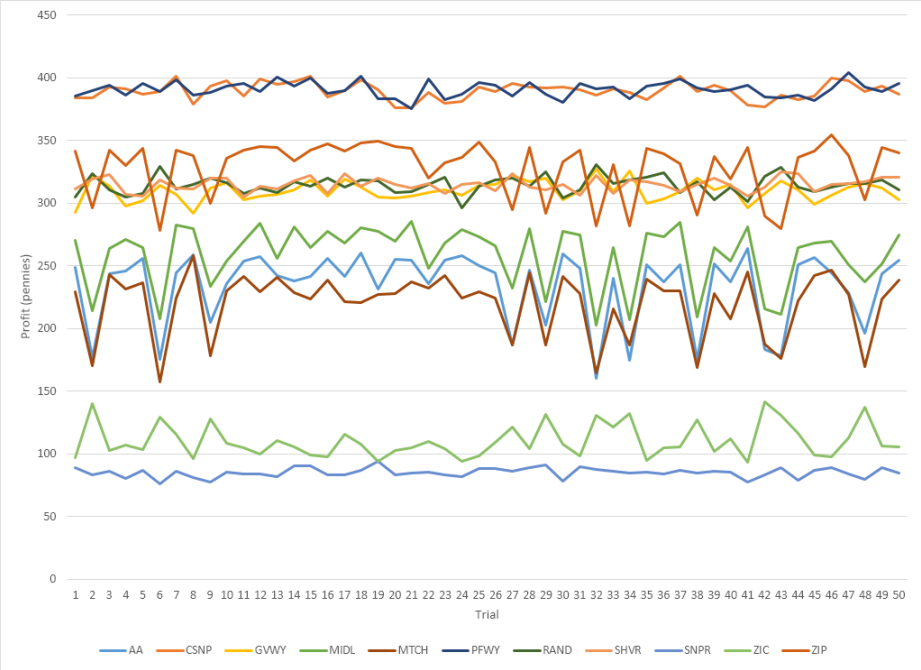

Fig. 1: Graph representing the performance of all the traders in a market with increasing fluctuations.

Experiment In this experiment, we ran all 11 traders with a constant range, varying from 50 - 100 up to 50 - 250. The stepmode was fixed. The market settings had an order schedule range varied between 50-100 and 50-250 and fixed stepmode. Customer intervals set to 10s interval with a drip-poisson timemode.

Results Contrary to our preconceptions, the traders which were significantly better than others were different from the previous experiment. However, varying ranges within this test mainly seemed to affect ZIC's performance. ZIC was consistently 10th and 6th respectively when the supply-demand schedule ranges were 50-150 and 50-250 respectively. This is most likely due to the range of valid bids (bids which can be accepted) increasing whilst the range of invalid bids (unacceptable bids up to the market worst) decreased. Other traders of note were AA and PFWY: AA improved from one of the worst to joint second-best trader and PFWY fell from joint first-best down to joint fifth.

Experiment We opted to ascertain the performance of the traders in following unexpected market behaviour (simulating the influence of world events on financial markets). To this end, we used swapped the supply schedules twice during the simulation. This led to a leap in stock price and the results showcase how the traders were influenced by the market and how they responded. The market settings had an order schedule consisting of two supply ranges, 10 - 190 and 200 - 300, the customers swapped to the second range during the middle third of the market session. Customer interval was once again set to 10 seconds with a drip-poisson time mode.

Results Figure 2 shows the fluctuations in the market. The steep variations correspond to the moments when the schedules were swapped. Figure 3 shows the profits of the traders over time. We notice the stabilising of profits while the traders try to adapt to the sudden change in stock price and the speedup when the schedule is swapped back to what it was originally.

Experiment In this experiment we compared the performance of two traders with ratios of 1:30, 30:30 and 30:1 with both traders having equal numbers of bidders and askers. The market settings consisted of an order schedule with one supply range of 50 - 250 and a fixed step mode. New customer orders were created at an interval of 10 with time mode set to drip-poisson.

Results The results for the ratio of 1:30 (1 buyer and seller of CSNP) are shown in the 2nd column of Figure 4. For the ratio 1:30 CSNP out performs all other traders as expected, this is because CSNP thrives in an environment with lots of traders outbidding each other, allowing it to cherry pick the best deals for itself. When the tables are reversed however and there are 30 CSNP buyers and sellers to 1 other buyer and seller, the market is virtually empty due to CNSP's nature to only accept deals and never make counter offers. This empty market dries up the potential for both market fluctuations and sales causing CSNP to perform terribly. The one peculiarity we have been unable to explain is CSNP's performance vs AA, where trade numbers seemed to far exceed the number of customers the one AA buyer and AA seller were given. Finally the 30:30 column performed as expected, with profit differences increasing against traders who

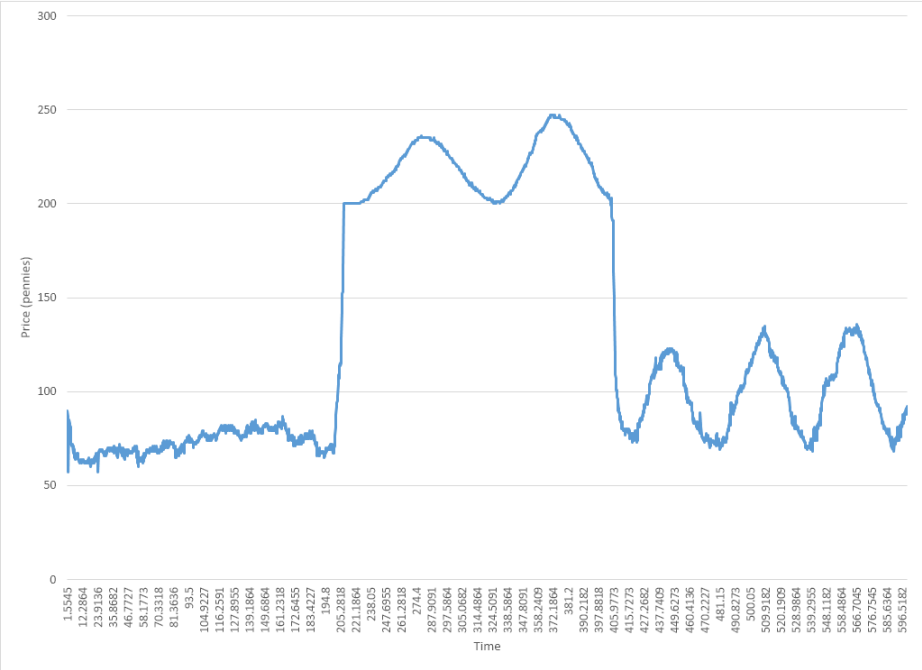

Fig. 2: Graph showing the artifically induced fluctuations in the stock price.

allow the market to fluctuate more and decreasing versus those who cause only slow changes.

Experiment This experiment aims to test traders performance as either 20 sellers or 20 buyers versus another trader of the opposite type. The combinations tried were CSNP vs AA, CSNP vs SHVR and CSNP vs ZIP. The market's order schedule had one supply range of 50 - 250 and a fixed step mode. New customer orders arrival times were modelled as a poisson process with a mean of 10.

Results As expected CSNP performed badly against both AA and SHVR as both a buyer or a seller (see Figure 5). This is because the AA starts out not very aggressive and SHVR only changes the best ask/bid by one, both of which will lead to a small standard deviation in the market. This small standard deviation leads to CSNP accepting offers that give it only the smallest of profit margins. However, ZIP makes larger changes to the best asks/bids allowing CSNP to demand a high profit on sales and thus generate more profit. In this kind of market CSNP would fare much better by simply waiting as long as possible before accepting offers, as this would allow the traders of the opposite type maximum time to out bid each other and reduce their own profits by as much

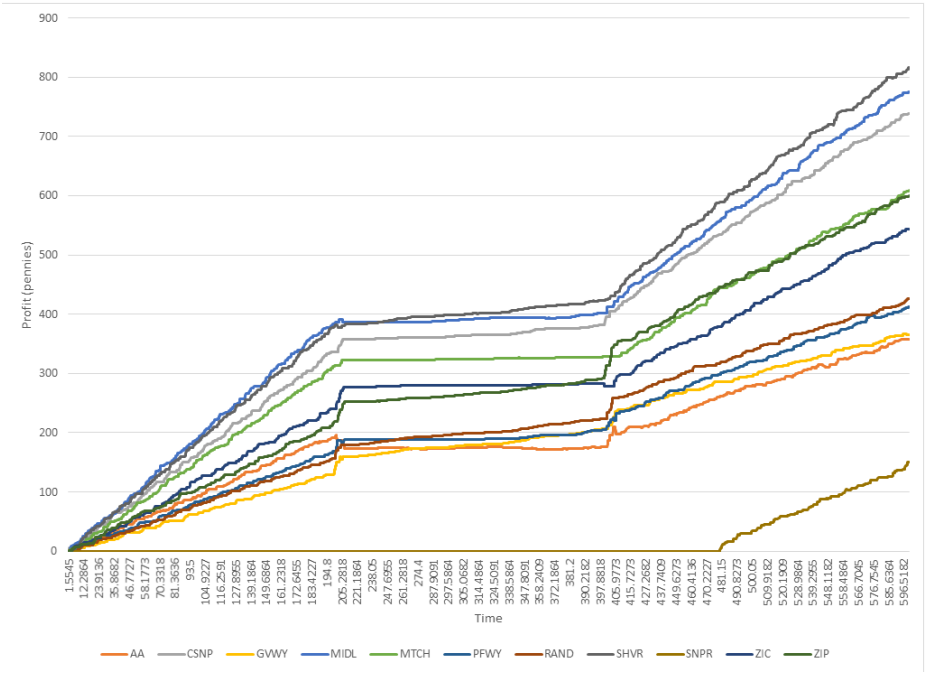

Fig. 3: Graph showing the reactions of the traders to the artificial fluctuations in the stock price.

Trader	1:30	30:1	30:30
ZIP	3.61	0.12	4.65
AA	2.60	0.72	0.68
SHVR	1.62	0.04	0.14

Fig. 4: Table listing the ratios of profit earned between CSNP and opposing traders in multiple scenarios.

as possible. The skews in results between whomever is buyer and whomever is seller is due to all three of ZIP, AA and SHVR making stub quotes at the 'worst' price if none has already been posted, and the buyers stub of 1 is far closer to the market's range than the sellers stub of 1000.

3 Future Improvements

3.1 Merging CSNP and PFWY

CSNP relies on giving away no information, has intelligence for choosing profit percent and, when accepting a deal, is quite likely to provide more profit than

Trader	CSNP Buyers	CSNP Sellers
ZIP	2.37	0.46
AA	0.09	0.05
SHVR	0.06	0.05

Fig. 5: Table representing the ratio of profit between CSNP and opposing traders when it or its opponents are either buyers or sellers.

the minimum. CSNP however struggles more and more the less time it's given to sell each deal.
PFWY posts offers, this means it will only earn a greater return if it accepts an offer as it's posted, however as it is in the market from then on it has the benefit of being accepted at any time after that. Profit away loses out on profit more often as time with a single customers offer increases.
Given more time we predict merging these two could lead to an even more successful trader. A trader which has its profit margin based on fluctuations in the market and uses its knowledge of how long it likely has left before a new offer. A trader which also posts an offer when it deems it better than holding out in the hopes that the market will be profitable when it next gets to bid/ask.

3.2 Using knowledge of the limit price's distance from the mean

One thing CSNP fails to take into account is how many standard deviations its limit is from the average trading price and from the current best ask/bid. These can cause issues when the trader expects a large profit because the market is fluctuating but its limit means the chances of the market allowing it to make its profit margin is remote. Therefore, our second suggested improvement would be to add some logic which influences the profit margin based on these variables.

4 Conclusion

In this paper, we have found that our suggested trader, CSNP, performs well in markets where the frequency of new orders is reasonably paced and the other traders induce market fluctuations. CSNP's desire for secrecy is also its main downfall however, as it thus needs far stronger logic than traders who actively participate in the exchanges as it has to decide whether it is worth waiting, hoping a better trade will come along.
In general, within BSE, we found traders that aimed for slightly higher profit margins over sales numbers performed better, however in the real world traders cannot ignore number of sales.

Appendix: Trader Code

```
class Trader_Custom_Sniper(Trader):

        def __init__(self, ttype, tid, balance):
                self.ttype = ttype
                self.tid = tid
                self.balance = balance
                self.blotter = []
                self.orders = []
                self.untraded = 0
                self.willing = 1
                self.able = 1
                self.lastquote = None
                self.time = 0
                self.duration = 0
                self.order_records = []
                self.profit_percent = 5
                self.profit_limit = 5
                self.new = False
                self.price_history = []
                self.time_history = []
                self.history_length=50
                self.call_times=[]

        def add_order(self, order):
                # in this version, trader has at most one order,
                # if allow more than one, this needs to be self.orders.append(order)
                if len(self.orders) > 0:
                        self.untraded += 1
                self.orders=[order]
                self.willing = 1
                self.new = True
                self.profit = order.price / float(100)
                if len(self.call_times) > 0 and len(self.time_history) > 0:
                        i=0
                        no_trades=False
                        while self.time_history[i] < self.call_times[0]:
                                i+=1
                                if i>=len(self.time_history):
                                        no_trades=True
                                        break
                        if not no_trades:
                                total=0
                                j=i
                                diff=[]
```

```
                                while i<len(self.price_history):
                                        total+=self.price_history[i]
                                        diff.append(self.price_history[i])
                                        i+=1
                                average=total/float(i-j)
                                diff=[(x-average)**2 for x in diff]
                                sigma=((sum(diff)/float(i-j))**0.5)
                        else:
                                average = order.price
                                sigma=1
                        self.profit_percent = (sigma / float(average)) * 100
                #print self.profit_percent
                if self.profit_percent > self.profit_limit:
                        self.profit_percent = self.profit_limit
                self.profit = self.profit * self.profit_percent

        def getorder(self, time, countdown, lob):
                self.add_time(time)
                if (len(self.orders) < 1):
                        order = None
                else:
                        #self.profit = self.profit-1
                        #if self.profit < 5:
                        #        self.profit = 5otype = self.orders[0].otype
                        limitprice = self.orders[0].price
                        quoteprice = self.orders[0].price
                        otype = self.orders[0].otype
                        #modifier=0
                        if len(self.order_records) > 0:
                                avgduration = sum(self.order_records)/len(self.order_records
                                if ((avgduration/100) * 90) < self.duration:
                                        self.profit = (limitprice/float(100)) * 0.1
                        #if len(self.price_history) == self.history_length:
                        #        grad = linalg.lstsq(array([self.time_history, ones(self.his
                        #        modifier=11*grad
                        if otype == 'Bid':
                                acceptprice = limitprice - self.profit
                        #        acceptprice += modifier
                                if lob['asks']['best'] < acceptprice:
                                        self.willing = 0
                                else:
                                        self.willing = 1
                        else:
                                acceptprice = limitprice + self.profit
                        #        acceptprice += modifier
```

```
                        if lob['bids']['best'] > acceptprice:
                                self.willing = 0
                        else:
                                self.willing = 1
                if self.willing == 0:
                        self.lastquote = quoteprice
                        order=Order(self.tid, otype, quoteprice, self.orders[0].qty
                else:
                        order = None
        return order

def add_history(self, element):
          #if len(self.price_history)==self.history_length:
          #        self.price_history.pop(0)
          #        self.time_history.pop(0)
          #if len(self.price_history)<self.history_length:
          self.price_history.append(element['price'])
          self.time_history.append(element['time'])
          #        return True
          #return False

def add_time(self, time):
          if len(self.call_times)==self.history_length:
                  self.call_times.pop(0)
          if len(self.call_times)<self.history_length:
                  self.call_times.append(time)

def respond(self, time, lob, trade, verbose):
        if self.new == True:
                if self.time > 0:
                        self.order_records.append(self.duration)
                self.time = time
                self.new = False
        self.duration = time - self.time
        if trade != None:
                self.add_history(trade)
```