QUANTUM REINFORCEMENT LEARNING FOR HIGH FREQUENCY TRADING

Alexander Kirnasov

Head of Quant Dev in Znamenka Capital

ABSTRACT

We introduce a new approach of Reinforcement Learning Application for High Frequency Trading called Quantum Reinforcement Learning as our agent learns to react on 'quantum' individual events in Limit Order Book – single Limit Order Book updates and single trades (and optionally single Orders if provided by Exchange). We claim that such level of learning granularity allows our agent to find optimal trading strategies by on-line modeling of Market Microstructure with a maximum rate and precision.

KEYWORDS

Deep Reinforcement Learning, High Frequency Trading

1. INTRODUCTION

There are 2 main approaches in Modeling of High Frequency Trading – traditional statistical approaches such as Market Microstructure modeling and more recent Machine Learning approaches detecting such Microstructure models 'on-line' without statistical modeling of various distributions in Limit Order Book data. Machine Learning models gained popularity on higher frequencies for it's dynamic nature as static statistical modeling is much more difficult on such trading frequencies. Machine Learning approaches can be further categorized in 2 main groups – supervised predictive models and unsupervised Reinforcement Learning models. Supervised models attempt to predict short term price spikes based on some history in Limit Order Book events preceding such spikes, manually marked by a supervisor from price history and presented to the model for further learning – typically to make 3 decisions on each tick – buy, sell or stay out of position. Such predictive models have some drawback of poor ability to generalize on unseen market data. On the other hand unsupervised Reinforcement Learning models allow agent to autonomously learn trading strategies by first acting randomly but then correcting itself trying to maximize final PnL. Such approach tends to generalize much better even in strongly stochastic market environment.

As RL in general can be treated as an optimization approach, there is a number of research papers, investigating how RL can be applied for algorithmic trading. Applications of RL in trading can be categorized in the following groups:

1. RL for optimal portfolio management [6]
2. RL for low frequency directional trading [1, 2]
3. RL for high frequency directional trading [3, 4]
4. RL for Optimal Trade Execution [5]
5. RL for Market Making [7]

David C. Wyld et al. (Eds): SIGI, CSTY, AIMLNET, AI – 2024
pp. 77-86, 2024. CS & IT - CSCP 2024 DOI: 10.5121/csit.2024.141307

Our paper is most closely related to group 3 – RL for high frequency directional trading and especially to recent work [4] "Deep reinforcement learning for active high frequency trading".

Typically one of the main characteristics of RL agent is a frequency of environment resampling and action decisions making. In above mentioned applications of RL in Finance typical frequency is based on fixed time intervals and usually these resampling intervals dt >> 1 second. In High Frequency Trading settings where latencies eg. on NYSE are now measured in nanoseconds RL agent should try to minimize its interval of decision making. In work [4] authors used dt = 0.1s and to our knowledge this interval is currently minimal in research papers devoted to applications of RL for HFT. In our paper we also use interval dt ~ 0.1s, however the main difference in our approach is that we allow our agent to make decisions not just based on regular time intervals and aggregated Limit Order Book information (Level 2 data) as for example in [4], rather our agent can make decisions based on individual Limit Order Book events – Market Orders submissions, Limit Orders submissions and Limit Order Cancellations (essentially Level 3 data) which theoretically can be done with dt close to nanoseconds which makes it more appropriate in actual High Frequency Trading. We use term 'Quantum' to specify that our agent tries to react on trading events with theoretically infinitely small latencies.

We claim that such Quantum Reinforcement Learning can find optimal trading strategies with maximal precision. We then investigate three major Reinforcement learning approaches in such learning environment – namely Cross Entropy, Deep Q-Learning and Policy Proximal Optimization. As yet another contribution of the current research we have observed importance of high level information for making trading decisions even on such trading frequencies which to our knowledge has not been used in previous publications on Deep Learning applications for High Frequency Trading. Namely, our trading agent is analyzing both micro events in Limit Order Books along with higher level information represented as time series of price candlesticks on various frequencies up-to 15 minutes. We claim that such information is important for agent as in some sense HFT is itself a computerized form of well know 'scalping' human trading strategies – hence we attempt to model our agent's behavior to closely follow 'scalping' which utilizes high level information as yet another indicator to enter positions. Eg – if there is a combination of 2 factors - 15 minutes based bearish trend and Limit Order Book is currently 'empty' on a buy side – it's a good indicator to enter short position. Adding lower frequency information also can be treated as accounting for Market state information in addition to Market microstructure information. Finally our models also accurately account for both maker and taker's fees on each transaction which makes this work quite practical.

As was noted above essentially our approach tends to minimize an interval of agent's decision making and hence we plan to extend the applicability of our approach to other applications of RL in algorithmic trading – for example to High Frequency Market Making. As opposed to directional trading Market Maker's main strategy is to 'earn the spread' and it is achieved typically via optimally quoting it's bid and ask prices. Typical applications of RL for Market Making such as [7] suggest requoting intervals dt ~ 1s. As discussed above it could be beneficial to requote with minimal latency in order to react quicker on Market events which otherwise can lead to a well known problem of adverse selection where Market Maker accumulates large inventories with losses while trading with better informed traders. In our next paper we plan to present results of applications of Quantum RL for high Frequency Market Making.

In addition in our next plans we consider integration of the 'news analysis' as part of Market State information for RL trading agent by utilizing modern Language models such as GPT3/GPT4. It is assumed that such analysis would be done with relatively lower frequency yet it could provide an agent with more data for making accurate trading decisions.

2. TRADING MODEL

Every Reinforcement Learning Model starts by specifying Environment States and Agent's actions along with description of how Environment State changes after agents undertakes certain actions and which reward agent gains by choosing specific actions.

Below is a description of three types of Microstructure events, that lead to the change in the trading environment.

1 Book Level Update event.

Typically most exchanges provide a Market Data channel that sends such update events: We represent it as a tuple ('update_type','price_level_absolute','price_level_relative','size_ absolute', 'size_delta', 'timestamp')

update_type can be either Insert, Delete or Update. Absolute Price Size specifies actual level's price and relative price's size specifies a number of an updated level in the book (eg best ask would be level 0 at sell side). Size's delta of the update specifies how much level is modified, absolute size specifies new size of the level after update is applied.

As can be noted we also add a timestamp as a part of event information which allows agent to learn potentially important information from current frequency of updates.

2 Trade event.

This is also a common channel where exchanges broadcast information on committed trades.
Trade occurs when exchange's matching engine matches incoming Market order with the queued limit orders based on Price / Submission time priority.

Trade is represented as a tuple: ('price_absolute','size', 'trade_direction', 'timestamp')

3 Individual trading order

Some exchanges also broadcast information on individual trading orders. In such case our model incorporates such events as well in the form of the following tuple: ('order_type','price_level_absolute','price_level_relative','order_size','timestamp')
Our agent makes trading decisions upon arrival of each individual event from the above list. It makes such decisions by supplying it's model's NN (Neural Network) with a current state of trading environment (coded in a way, described below).
Below is a description of the coding scheme for the environment state.
Environment State is represented as a combination of two States: Market Microstructure State (State_MM) + Agent's Positions State (State_AP).
Let us first describe State_MM.
In turn this state can be split into three parts: State_LOB + State_micro_structure_events + State_high_level_prices
State_LOB is a current state of a Limit Order Book with 10 best sell and buy price levels with sizes
+ a timestamp.
State_micro_structure_events is represented as 3 time series of events of each type (book update, trade, individual order). Each time series contains a 100 of last events of corresponding type, coded in a way described above.

State_high_level_prices represents 3 time series, composed of 10 candlesticks each on the following time frequencies (15 minutes, 1 minute, 1 second) + 3 pairs of Support, Resistance Levels
– more specifically Min and Max prices for each series. Support and Resistance levels information is considered to be a hint to the model to act differently if price is approaching one of such levels compared to times when price is in the middle of the corresponding price range. Candlestick information is coded as a following tuple: ('price_open', 'price_max','price_min','price_close')
Let us now describe State_AP – a state, representing agent's current trading position This state is split into two following parts: State_Open_Position + State_Limit_Orders
State_Open_Position represents a total current open position as an open position's size – positive for buy or negative for sell.

In general it would be tempting to represent State_Limit_Orders as a vector of certain length of orders on every book level less than vector's length. This way agent could maintain multiple limit orders on different book levels. However in this work simplifying assumption was made to maintain only 2 limit orders for best ask and best bid assuming that maintaining a net of limit orders maybe beneficial only in actual trading environment where it is important to add an order in advance to prevent price 'slipping' effect. In this work however we do not analyze such effects and hence limit our orders to just best bid and best ask. Hence State_Limit_Orders is represented as a pair ('size_best_bid', 'size_best_ask').
Now, once a Trading Environment is specified let us proceed to the description of agent's possible actions and rewards for those actions.
We assume that agent's action can be represented as a tuple: ('best_ask','best_bid','episode_close_indicator').

Specifying these delta's instructs and agent to properly maintain his two current limit orders. Depending on the output from the model's NN – he can decide to keep same order sizes or possibly increase some of them, or decrease or completely cancel.

Episode_close_indicator when set to 1 instructs an agent to close his open position and finalize episode of Reinforcement Learning.

Let us now describe how agent is being rewarded for actions.

There are two possible ways to simulate how Book Level is being traded. It is common to assume market orders arrive with exponential times with some rate r and it can be possible to estimate when potentially agent's order can be consumed partly or fully. Potentially exponential arrival rates could be calibrated from information on Limit Order Book updates presented above (for details please refer to work [8]), however in this work another approach is used for estimation of time when agent's order is consumed. Namely internally simulation maintains approximate location of agent's order inside level's orders FIFO and thus it is assumed that first trade, that consumes level's size up-to agent's location would also consume agent's order. In this case agent's reward is calculated as follows:

R = maker_fee – trade_price*size, maker_fee > 0
If size is positive and that is a limit buy – we assume that agent spends money to buys specified size, if size is negative and that is limit sell – we assume – agent is payed back for specified trades size.

In both cases agent is rewarded with maker_fee.

The moment when position is filled is not directly related to trader's current action – rather it depends on his previous actions and that is a typical scenario for Reinforcement Learning.
Another scenario when agent is reward in our model is when episode is closed. In this case we compute reward as follows:

R = taker_fee + trade_price * open_size, taker_fee < 0
In this case when open_size is positive and episode is closed – agent sells his current position and gets rewarded with the sold amount. If open size is negative – agent is closing his short position and pays corresponding amount for that. In both cases agent pays a taker's fee as he is using a market order.

Finally, let us describe how environment is modified upon every step

There are two changes that happen in the environment upon event's arrival: first of all a time series of corresponding event's type is shifted by 1 event in State_MM. In addition – State_MM's Limit Order Book is properly updated. Finally, possible changes in State_high_level_prices state is monitored based on timestamp of the event – eg next second's candlestick may be closed if event's timestamp crosses previous second. In addition to that State_AP can be modified as follows: if agent's action specifies changes in his limit orders – that is being done. Also if it is estimated that agent's order is consumed – his limit order is updated properly as well as his open position.

3. AGENT'S NEURAL NETWORK

Depending on the Reinforcement Learning Approach used – weather it is DQN or Proximal Policy Optimization or Cross Entropy – output of the network differs. For DQN – it is estimated value of a pair (S,a) – where S is environment's state and a is an agent's action. Agent selects and action which maximizes Q(S,a). If it is Proximal Policy Optimization or Cross Entropy – output is a probability distribution of agent's possible actions in a given state. In both cases however there is a common part of network architecture which consumes state of the environment as an input and is described below.

LOB part of the input state is processed by 2 Linear Layers of 64 neurons each, producing output of length 10
Each of 3 events time time series is processed by LSTM with 64 neurons – each producing an output vector of length 10
Each of 3 candlestick time series is processed by LSTM with 64 neurons – each producing an output vector of length 10
3 pairs of Support / resistance Levels are processed with single Linear Layer with 64 neurons with out put of length 3
All output vectors are then concatenated to produce a single output of length 73 which is then processed by another Linear Layer with 64 neurons to produce either Q-value for DQN or a probability distribution for Proximal Policy Optimization and Cross Entropy approaches.

Graphically architecture is depicted below (Figure 1):

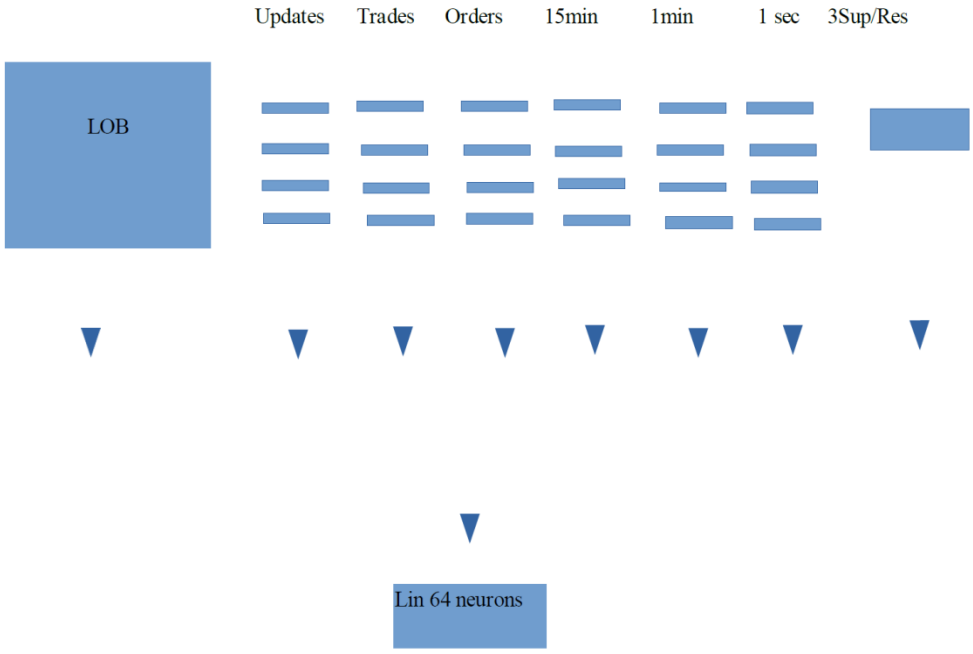

Figure 1. Graph architecture

4. EXPERIMENTAL SETUP AND RESULTS

In order to evaluate the above Quantum RL scheme, data was collected using WebSocket Binance Api. Network was trained on the following parsed data from Binance exchange:

It represents about 8 hours of High Frequency data (Limit Order Book combined with Trade History) with mean interval between events ~ 0.1s.

Testing was done on a different ~ 8 hours data file from Binance: The trained model was evaluated on the following day.

We checked all three above mentioned alternative for RL – DQN, PPO and CE The best results of trading simulation were for DQN RL and are discussed below. PyTorch implementation is available on GitHub at following URL: https://github.com/akirnasov/Quantum-RL Train and test data can be found in the same repository (please check for *.txt files).

In order to evaluate benefits of the proposed Quantum RL approach we compare it with closest approach from work [4] “Deep reinforcement learning for high frequency trading”. In this work authors use Policy Proximal Optimization for RL implementation and they also use only Level 2 data with 0.1 s sampling frequency. Hence in our experiments we compare learning results of Quantum RL approach (DQN implementation with event based trading data and lower frequency market state data) separately with PPO implementation and with DQN based only on Level 2 data.

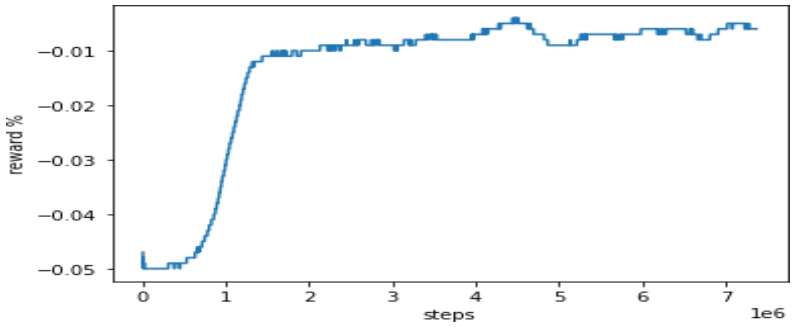

Figure 2. Graph reward % (Quantum RL)

On the above chart (Figure 2) we plotted mean reward for the last 100 steps of education. Reward assumes 0.025% commission on each position's open / close – so chart starts with -0.050% and then between 1M and 1.5M steps rapidly grows hitting -0.004% after around 2M steps (~12 hours training on RTX3090).

Below we provide two similar charts – in the first case for PPO implementation and in the second case when only using Level 2 data.

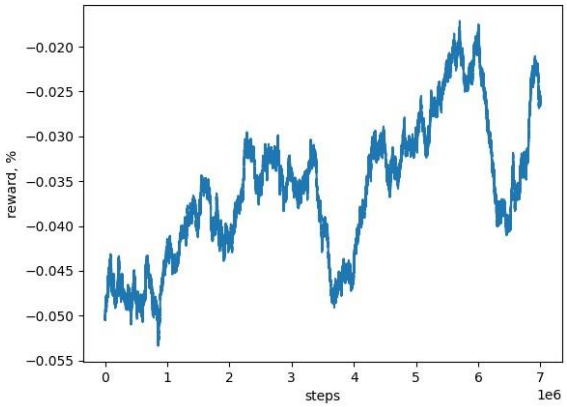

Figure 3. Graph reward % (PPO)

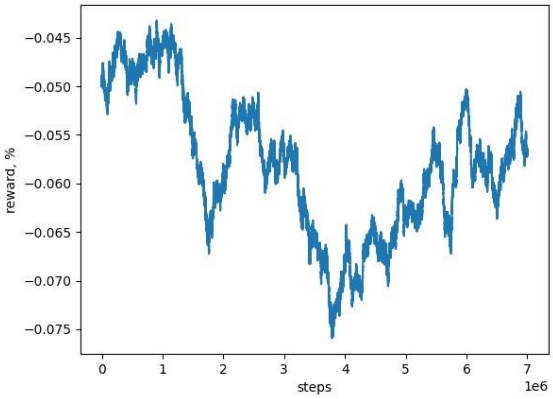

Figure 4. Graph reward % (DQN with L2 data only)

As we can see in both cases trading agent was not able to reach positive PnL even after 7 Mln steps of learning (~48 hours using RTX3090). We believe that DQN is better suitable for algo trading RL as it is typically used for discrete action spaces, while adding trading events to Level 2 as well as lower frequency market state information allows our agent to make better trading decisions.

On the next chart (Figure 5) we plot how long the position is kept on average.

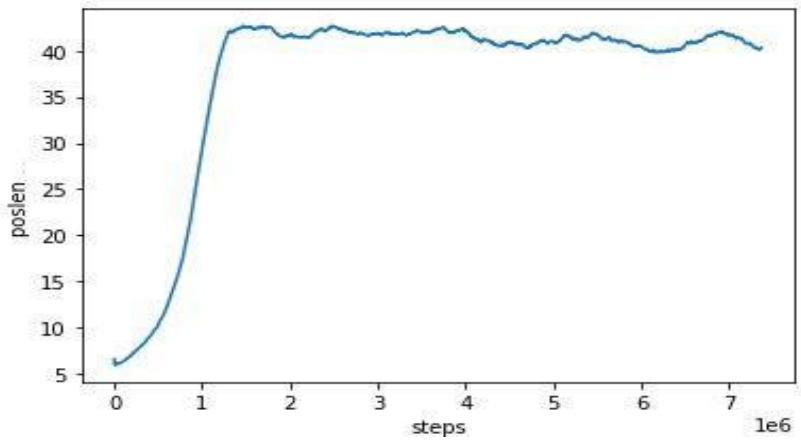

Figure 5. Graph poslen

It can be seen that Position Length grows to around 40 ticks and stabilizes. That corresponds to around 4 seconds on average. That can be treated in a way, that network learned to catch short term spikes based on 500 previous ticks history.

Finally, on the below chart (Figure 4) we plotted the PnL result of our HFT strategy on the test ~8 hours' time frame. It should be noted that in this case we set actual commission – 0.1% on Binance.

As can be seen our strategy was able to generate ~ 4% PnL in ~ 8 hours and at max level it reached almost 8% PnL.

Should be noted, that test run was done with reward function set to percentage of the price change from the first to the last tick. While during education reward function was set to add small percentage change of the current tick. That is why even though the reward graph only goes up-to -0.004% and with lower commission 0.025%, still – the test run goes to the positive PnL area as percentage change between last and first ticks is always higher than sum of percentage changes on every tick that can be easily derived mathematically.

So, we can conclude using RTX 3090 it is possible to train network on previous day data at night time and then use it for profitable trading the very next day and so on. In the current setup with event mean interval of 0.1 s and average inference on CPU taking ~0.01s that is quite feasible to implement in actual HFT platform.

It is also possible to add on-the fly training during next day. Eg: every M minutes weights of the trading Neural Network can be updated with the weights from the trainable Neural Network that runs in parallel. This can be used to better account for the changing trading environment. Parameter M could be considered as a hyper parameter and tuned separately for each traded instrument.

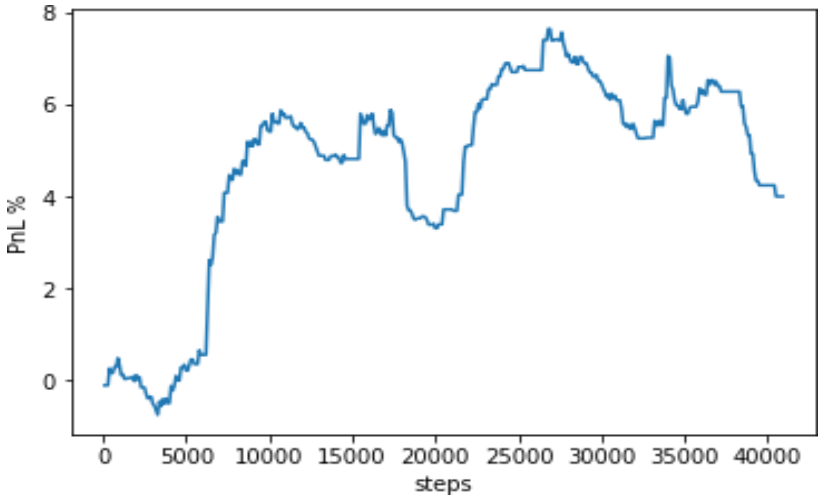

Figure 6. Graph PnL%

5. LIMITATIONS

One of the key limitations of Quantum RL approach is a performance consideration within actual High Frequency Trading platform as in practice we should consider inference latency when making RL decisions via Neural Network. However it is not rare these days to use specialized hardware in High Frequency Trading such as ASIC chips (Application Specific Integration circuit) and GPU (Graphics Processing Units) which makes such approach quite feasible. Even using just modern High Performance CPUs it is still possible to apply the described approach on Crypto Exchanges as we have shown in the above research. Also it is possible to consider Attention based architectures instead of LSTM which could make inference much faster along with other possible optimizations for the Neural Networks including binary code optimizations.

6. CONCLUSIONS

In this paper we presented a novel approach called Quantum Reinforcement Learning for High Frequency Trading. We evaluated approach using simulation on Binance Exchange and showed that it can be used for profitable HFT strategies. We also provided a link to PyTorch implementation for interested researchers. We consider possible extensions of the proposed approach for High Frequency Market Making in the future plans.

REFERENCES

[1] "Learning to trade via direct reinforcement," IEEE transactions on neural Networks, vol. 12, no. 4, pp. 875–889, 2001.

[2] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai, "Deep direct reinforcement learning for financial signal representation and trading," IEEE transactions on neural networks and learning systems, vol. 28, no. 3, pp. 653–664, 2016.

[3] H. Wei, Y. Wang, L. Mangu, and K. Decker, "Model-based reinforcement learning for predictions and control for limit order books," arXiv preprint arXiv:1910.03743, 2019.

[4] A. Briola, J. Turiel, R. Marcaccioli, and T. Aste, "Deep reinforcement learning for active high frequency trading," arXiv preprint arXiv:2101.07107, 2021.

[5] Y. Nevmyvaka, Y. Feng, and M. Kearns, "Reinforcement learning for optimized trade execution," in Proceedings of the 23rd international conference on Machine learning, 2006, pp. 673–680.

[6] Z. Wang, B. Huang, S. Tu, K. Zhang, and L. Xu, "Deeptrader: A deep reinforcement learning approach for risk-return balanced portfolio management with market conditions embedding," in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, no. 1, 2021, pp. 643–650.

[7] Gasperov and Z. Kostanjcar, "Market making with signals through deep reinforcement learning," IEEE Access, vol. 9, 2021, pp. 61 611–61 622

[8] Rama Cont, Sasha Stoikov and Rishi Talreja A stochastic model for order book dynamics, Operations Research, Volume 58, No. 3, 2010, pp. 549-563.