Reinforcement Learning for Optimized Trade Execution

Yuriy Nevmyvaka yuriy.nevmyvaka@lehman.com
Lehman Brothers, 745 Seventh Av., New York, NY 10019, USA
Yi Feng fengyi@cis.upenn.edu
Michael Kearns[1] mkearns@cis.upenn.edu
University of Pennsylvania, Philadelphia, PA 19104, USA

Abstract

We present the first large-scale empirical application of reinforcement learning to the important problem of optimized trade execution in modern financial markets. Our experiments are based on 1.5 years of millisecond time-scale limit order data from NASDAQ, and demonstrate the promise of reinforcement learning methods to market microstructure problems. Our learning algorithm introduces and exploits a natural "low-impact" factorization of the state space.

1. Introduction

In the domain of quantitative finance, there are many common problems in which there are precisely specified objectives and constraints. An important example is the problem of *optimized trade execution* (which has also been examined in the theoretical computer science literature as the *one-way trading* problem). Previous work on this problem includes (Amgen and Chriss, 2002), (Bertsimas and Lo, 1998), (Coggins et al., 2003), (El-Yaniv et al., 2001), (Kakade et al., 2004), and (Nevmyvaka et al., 2005). In this problem, the goal is to sell (respectively, buy) V shares of a given stock within a fixed time period (or horizon) H, in a manner that maximizes the revenue received (respectively, minimizes the capital spent). This problem arises frequently in practice as the result of "higher level" investment decisions. For example, if a trading strategy – whether automated or administered by a human – decides that Qualcomm stock is likely to decrease in value, it may decide to short-sell a block of shares. But this "decision" is still underspecified, since various trade-offs may arise, such as between the speed of execution and the prices obtained for the shares (see Section 2). The problem of optimized execution can thus be viewed as an important horizontal aspect of quantitative trading, since virtually every strategy's profitability will depend on how well it is implemented.

For most of the history of the U.S. equity markets, the only information available to an agent attempting to optimize trade execution was the sequence of prices of already-executed trades, and the current bid and ask (the best outstanding buy and sell prices offered). But in recent years, electronic markets such as NASDAQ have begun releasing, in real time, all of the outstanding buy and sell limit order prices and volumes (often referred to as order book or market microstructure data). It is not difficult to see that such data might be extremely valuable for the problem of optimized execution, since the distribution of outstanding limit orders may help predict short-term price direction, likelihood of execution at a given price, buy or sell side imbalances, and so on.

In this paper, we report on the first extensive empirical application of reinforcement learning (RL) to the problem of optimized execution using large-scale NASDAQ market microstructure data sets. Our basic experimental methodology (detailed in Section 3) consists of the following steps:

1. The identification of a set of (observed) state variables that can be derived from the microstructure data and the RL algorithm's own actions. These variables are divided into "private" variables (such as the amount of time and shares remaining for the algorithm in the execution problem) and "market" variables (which reflect various features of the trading activity in the stock).

2. The application of a customized RL algorithm, which exploits the structure of the execution optimization problem to improve computational efficiency, to extremely large datasets. Our empirical results are based on 1.5 years of very high-frequency (millisecond time scale) microstructure data for several NASDAQ stocks.

3. The empirical comparison of the RL-optimized execution policy to a variety of natural baseline execution strategies.

Portions of this work were conducted while the authors were in the Equity Strategies department of Lehman Brothers in New York City. Appearing in Proceedings of the 23 rd International Conference on Machine Learning, Pittsburgh, PA, 2006.

Our main contributions are:

1. A careful experimental demonstration that RL approaches are well-suited for optimized execution, and can result in substantial improvements in performance. For instance, we show that the execution policies learned by RL can improve relative performance by as much as 50%.

2. An efficient RL algorithm that is carefully crafted to take advantage of the structural features of order book trade execution. It fuses Q-learning and dynamic programming and exploits the approximate independence between private and market variables.

3. A study of the relative value of a variety of market variables, including several that require and exploit the recent revelation of microstructure data. An example of a valuable microstructure variable is the current cost of a market order submission.

4. A detailed analysis of the policies learned by RL, and how they depend on the observed state variables. We show that these policies can be interpreted naturally in terms of the state variables and the constraints of the execution problem.

We note that while several previous works have proposed the use of RL to various problems in quantitative finance, including optimized execution and market-making – see (Bertsimas and Lo, 1998), (Chan et al, 2001), (Tesauro and Bredin, 2002), and (Kim and Shelton, 2002) – this is the first large-scale empirical application, and the first to employ microstructure data.

2. Limit Order Trading and Market Simulation

In this section we provide only the briefest background on limit orders and microstructure necessary to understand our results. Modern financial markets such as the NASDAQ are *limit order* markets. By this we mean that buyers and sellers specify not only their desired volumes, but their desired prices as well. A limit order to buy (respectively, sell) V shares at price p may partially or completely execute at prices at or below p (at or above p).

For example, suppose that NVIDIA Corp. (NVDA) is currently trading at roughly $27.22 a share (see Figure 1, which shows an actual snapshot of an NVDA order book), but we are only willing to buy 1000 shares at $27.13 a share or lower. We can choose to submit a limit order with this specification, and our order will be placed in the buy order book, which is ordered by price, with the highest price at the top (this price is referred to as the *bid*; the lowest sell price is called the *ask*). In the example provided, our order would be placed immediately after the extant order for 109 shares at $27.13; though we offer the same price, this order has arrived before ours. If an arriving limit order can be immediately executed with orders on the opposing book, the executions occur. For example, a limit order to buy 2500 shares at $27.27 will cause execution with the limit sell orders for 713 shares at $27.22, and the 1000-share and 640-share sell orders at $27.27, for a total of 2353 shares executed. The remaining (unexecuted) 147 shares of the arriving buy order will become the new bid at $27.27. It is important to note that the prices of executions are the prices specified in the limit orders *already in the books*, not the prices of the incoming order that is immediately executed. Thus we receive successively worse prices as we consume orders deeper in the opposing book.


INET home system stats help

inet

NVDA

GET STOCK

NVDA go

Aggregate by Price

LAST MATCH		TODAY'S ACTIVITY	
Price	27.2200	Orders	59,437
Time	16:00:41	Volume	1,336,147

BUY ORDERS		SELL ORDERS	
SHARES	PRICE	SHARES	PRICE
9	27.1900	713	27.2200
100	27.1800	1,000	27.2700
9	27.1800	640	27.2700
109	27.1300	500	27.4300
1,843	27.0400	500	27.4500
100	26.7500	20	27.8000
700	26.4900	700	27.8200
1,000	25.0000	1,000	27.9300
700	24.8000	400	28.0000
2,300	24.4800	75	28.0000

Figure 1. Snapshot of NVDA order book.

In our study, we used historical records from the INET Electronic Communication Network, an electronic, completely automated stock exchange, which accounts for a significant volume of trading in NASDAQ stocks. We have developed a simulator (or a generative model in RL terminology) that combines the real world INET order flow with artificial orders generated by our execution strategies. It executes orders and maintains priorities in order books in the way we have just described. Such a setup allows us to run simulations in historical order books, and capture all costs and uncertainties of trade execution: the bid-ask spread, market impact, and the risk of non-execution. Our experiments are based on three stocks: Amazon (AMZN), Qualcomm (QCOM), and NVIDIA (NVDA). These stocks were chosen to examine how the performance and policies found by RL vary with stock properties such as liquidity, volume traded, and volatility. Because of the frequency and detailed nature of our data, market datafiles are extremely large – transaction records in each stock take up several GBs of disk space. We separated the available data into disjoint training and test sets (12 and 6 months respectively).

3. An RL Formulation of Optimized Execution

From the discussion of the last section, the implications of order book dynamics for the problem of optimized trade execution should be immediately apparent. In particular,

if our goal is to sell V shares in time horizon H, one option is to submit a market order immediately at the beginning of the time interval – that is, effectively place a limit order to sell all V shares at price 0, which will have the effect of giving us the V best prices in the buy book at that moment. However, if V is large we will be receiving progressively worse prices for subsequent shares as we churn through the buy book – a visceral form of market impact.

A potentially more intelligent class of strategies are the *submit and leave* (S&L) policies. Here we pick a fixed limit order price p, and place a sell order for all V shares at price p. After H minutes, we go to the market with any remaining (unexecuted) shares. Such strategies provide a simple framework for trading off the likelihood of execution with the prices obtained. And even though S&L appears simplistic, it captures the important reality of costly monitoring. Most real-world traders deal with large diversified portfolios and thus do not have the luxury of devoting all their attention to a single stock. In the S&L setting, orders cannot be monitored continuously and revised instantaneously, which realistically exposes them to market fluctuations and adverse selection. See (Nevmyvaka et al., 2005) for an extensive empirical study of this restricted but interesting class of strategies. Of course, we would like to be able to entertain considerably richer classes of execution strategies. It is very natural to consider *state-based* strategies – strategies that can examine salient features of the current order books and our own activity – in order to decide what to do next. We now give the RL formulation of optimized execution we employed in our experiments, describing the state, action and reward structure, as well as the algorithm.

States. Each *state* $x \in X$ is a vector of attributes (state variables) that describes the current configuration of our system. We note that where we say *state* we more precisely mean *observed state*; we in no way suggest that our state representations are sufficient to render a system as complex as modern financial markets truly Markovian. In what follows we will essentially be treating a partially observable environment as if it were fully observable to us; the test of this assumption will lie in our empirical evaluation.

While we will explore a number of different state representations, there are two important variables that appear in all of them. These are the *elapsed time t* and the *remaining inventory i,* which represent how much time of the horizon H has passed and how many shares we have left to execute in the target volume V. Different resolutions of accuracy will be investigated for these variables, which we refer to as *private* variables, since they are essentially only known and of primary concern to our execution strategy. More precisely, we pick I and T, which are the resolutions (maximum values) of the private variables i and t, respectively. I represents the number of inventory units our policy can distinguish between – if V = 10,000 shares, and I = 4, our remaining inventory is represented in rounded units of V/I = 2,500 shares each. Similarly, we divide the time horizon H into T distinct points at which the policy is allowed to observe the state and take an action; for H = 2 min and T = 4 we can submit a revised limit order every 30 seconds, and the time-remaining variable t can assume values from 0 (start of the episode) down to 4 (last decision point of the episode).

We will also examine a number of additional state variables we refer to as *market* variables, which compute various properties of interest of the limit order books and recent activity in the stock. Thus, a state has the form $x_m = <t, i, o_1, \ldots, o_R>$, where the o_j are market variables. We describe our selection of market variables in Section 4.

Actions. As is standard, policies will map states to actions, but we also need to place some limits on the action space. Our action space can be thought of as a simple limit order price at which to reposition all of our remaining inventory, relative to the current ask or bid. In other words, for the problem of selling V shares, action a corresponds to placing a limit order for all of our unexecuted shares at price *ask - a*. Thus we are effectively withdrawing any previous outstanding limit order we may have (which is indeed supported by the actual exchanges), and replacing with a new limit order. Thus a may be positive or negative, with $a=0$ corresponding to coming in at the current ask, positive a corresponding to "crossing the spread" towards the buyers, and negative a corresponding to placing our order in the sell book. For the problem of buying, a is interpreted relative to the current bid. For either buying or selling, more positive actions mean movement towards the opposing book, and more negative actions mean placement deeper within our own book.

Rewards. An action from a given state may produce immediate rewards, which are essentially the proceeds (cash inflows or outflows, depending on whether we are selling or buying) from any (partial) execution of the limit order placed. Furthermore, since we view the execution of all V shares as mandatory, any inventory remaining at the end of time H is immediately executed at market prices --- that is, we eat into the opposing book to execute, no matter how poor the prices, since we have run out of time and have no choice.

In order to permit comparisons of policies and performance across stocks, which may have very different per-share prices, we always measure the execution prices achieved by a policy relative to the mid-spread price (namely, *(ask+bid)/2*) at the start of the episode in question. Thus, we are effectively comparing performance to the idealized policy which can execute all V of its shares immediately at the mid-spread, and thus are

assuming infinite liquidity at that point. Since this idealized policy cannot in general be realized, we always expect to do worse, but this provides a convenient common baseline for the comparison of multiple policies that can be realized. We define the *trading cost* of a policy as the underperfomance compared to the mid-spread baseline, measured in the standard financial unit of basis points, or 1/100 of a percent. Our reward function captures the most important aspects of execution – bid-ask spread, market impact, opportunity cost, etc. As our study is primarily aimed at large institutional investors, we assume that commissions and exchange fees are negligible. We also assume direct and fast access to exchanges – i.e. we do not account for possible order arrival delays.

Algorithm. We experimented with a variety of standard and modified RL algorithms, but all results are reported from a very efficient algorithm that exploits the particular properties of optimized execution and financial markets. Speed and efficient (re-)use of data make our approach particularly attractive.

The feature that allows us to reduce the number of state optimizations is the (approximately) Markovian nature of trade execution: if our state space is properly defined, the optimal action at any given point in time is (approximately) independent of any previous actions. By the Markov property, the optimal action in a state at elapsed time $t = \tau$ is independent of the actions in all states with elapsed time $t \leq \tau$. Extending this logic, optimal actions in states with $t = T$ (no time remaining) are completely independent from all other actions. Indeed, when time runs out, we are forced to submit a market order for all unexecuted shares to bring our inventory to the target level V (no mater what else we do between $t = 0$ and $t = T-1$). Now we can solve the problem inductively: having assigned optimal actions for all states with $t = T$ (mandatory market order), we now have all the information we need to determine the optimal action for all states with $t = T-1$; having done that, we move one time step back to $t = T-2$, and so on until $t = 0$. Having arrived at $t = 0$, we have a globally optimal policy (under our Markovian assumption).

Another significant assumption we shall make is that our own actions do not affect the behavior of other market participants. This means that we do not explicitly model strategic order submission by other traders, like "stepping in front" of large orders. According to our formal notation, we will assume that our actions do not affect market state variables $o_1 \ldots o_R$, but only the private variables t (with every action that we take t increases by 1) and i (as we gradually execute our order, i decreases monotonically). We exploit this independence property of public and private variables to use the available data more efficiently (and thus reduce overfitting), and to make sure that we optimize every state that can be ever encountered within our dataset. For every interval H/T in our data, we can ask (and get an answer to) a question: what is the optimal action in this state if we were to encounter this particular state with $t = 0, 1, 2 \ldots T$ periods remaining? Then we use the exact same "trick" with the other private variable – remaining inventory i. We try every possible value of $i \in [0, V/I]$ in every state we are optimizing. This ensures that every possible state that can be generated from our dataset $\{t, i\}^* \{o_1 \ldots o_R\}$ gets visited.

Finally, we explain our methodology for determining an optimal action in a given state. Our approach is similar to Q-learning: in every state we encounter, we try all possible actions and update the expected cost associated with taking each action and following the optimal strategy afterwards. Taking an action results in an immediate payout if any number of shares get executed, and it transfers us into a new state one time step later. Since our learning moves backwards in time, this new state has already been optimized and we know the expected cost of following the optimal strategy from that state. More formally, our cost update rule is the following:

$$c(x, a) = \frac{n}{n+1} c(x, a) + \frac{1}{n+1} [c_{im}(x, a) + \arg\max c(y, p)],$$

where $c(x, a)$ is the cost of taking action a in the state x and following the optimal strategy in all subsequent states; $c_{im}(x, a)$ is the immediate (1-step) cost of taking action a in state x; y is the new state where we end up after taking a in x; n is the number of times we have tried a in x, and p is the action taken in y.

In order to learn an optimal strategy, we need to go through the dataset $T*I*L$ times (L is the number of actions in each state). This is a much smaller number than the worst-case scenario, and it has several interesting properties. First, the running time of our algorithm is independent of the number of market variables R, and, second, we can increase T and I arbitrarily without risking overfitting because of the efficient use of our data. Figure 2 plots running times as functions of T, I, and R.

Here is our algorithm in pseudo-code:

```
Optimal_strategy (V, H, T, I, L)
    For t = T to 0
      While (not end of data)
        Transform (order book) → o1 ... oR
        For i = 0 to I {
            For a = 0 to L {
                Set x = {t, i, o1 ... oR}
                Simulate transition x → y
                Calculate cim(x, a)
                Look up argmax c(y, p)
                Update c(<t,v, o1 ... oR>, a)
    Select the highest-payout action argmax c(y, p) in
    every state y to output optimal policy
```

We want to emphasize that the main assumption exploited is the approximate independence of private and market variables. More precisely, during *training only*, we assumed that the actions of our policy had *no effects* on the subsequent evolution of order books. In other words, after simulating the number of shares we would have executed from a given order book state (thus allowing the appropriate updates of the private variables), we next advance to the order books as they evolved *without our trading*. This is tantamount to the independence assumption just articulated. The great advantage of this assumption is that it renders our computation time and sample complexity nearly *independent* of the number of market (but not private) variables, since our actions do not influence market variable evolution. Of course, this assumption must be validated, and it is: all of our results are reported on *test data* in which this assumption was *not* made --- all test set order book simulations maintain the impacts of any policy actions. The strong empirical results obtained on the test data despite the independence assumption made in training verifies its approximate validity.

Experimental Methodology. For every class of policies we investigate, we went through the following sequence of steps:

1. We investigated three stocks: AMZN, NVDA, and QCOM; two order sizes V: 5,000 shares and 10,000 shares; and two execution horizons H: 2 minutes and 8 minutes. We tried every combination of stock, V, and H for every state space representation.

2. A choice for the parameters I and T was made, which are the resolutions of the private variables i and t, respectively. These resolutions were generally kept small for purposes of a reduced state space.

3. Market variables were selected. Market variables summarized a variety of information from the order books (current and past) into a number of low-resolution features. For example, executed market volume can take on any value between 0 and 1,000,000 shares, but we convert it into a proxy variable that takes values of 0 (low), 1 (medium), and 2 (high). In a sense, market variables are a "blurry" representation of the real world. Choices for I, T, and the market variables completely specify a state space representation. The size of state space can vary from a dozen or so states (just T and I with very low resolution) to over a thousand (several market variables, high resolution).

4. We partitioned our training data into episodes, and applied our RL algorithm to learn an optimized execution policy over the chosen state space. For instance, when we set H to 2 minutes, we partitioned our 1-year INET training data into approximately 45,000 episodes. In general we enjoyed the luxury of large training and test sets, which is an advantage of using microstructure data.

5. Policies learned via RL were then compared on the 6-month test set with several baseline strategies (always measured by trading costs in basis points over the mid-spread price at the opening of the episode; see above). The most basic comparison is with the optimized submit-and-leave strategy. However, also of interest are comparisons between RL policies obtained using different state variables, since this can shed light on the relative value of these variables.

4. Experimental Results

We begin our presentation of empirical results with a brief discussion of how our choice of stock, volume, and execution horizon affects the RL performance across all state space representations. All other factors kept constant, the following facts hold (and are worth keeping in mind going forward):

1. NVDA is the least liquid stock, and thus is the most expensive to trade; QCOM is the most liquid and the cheapest to trade. 2. Trading larger orders is always more costly than trading smaller ones. 3. Having less time to execute a trade results in higher costs. In simplest terms: one has to accept the largest price concession when he is selling a large number of shares of NVIDIA in the short amount of time.

4.1 Learning with Private Variables Only

The first state space configuration that we examine consists of only the two private variables t (decision points remaining in episode) and i (inventory remaining). Even in this simple setting, RL delivers vastly superior performance compared to the class of optimized submit-and-leave policies (which is already a significant improvement over a simple market order). Figure 3 shows trading costs for all stocks (AMZN, NVDA, QCOM), execution times (8 and 2 minutes), and order sizes (5K and 10K shares) for the submit-and-leave strategy (S&L), the learned policy where we update our order 4 times and distinguish among 4 levels of inventory ($T=4, I=4$) , and the learned policy with 8 updates and 8 inventory levels ($T=8, I=8$). We find that learned execution policies significantly outperform submit-and-leave policies in all settings. Relative improvement over S&L ranges from 27.16% to 35.50% depending on our choices of T and I (increasing either parameter leads to strictly better results).

How can we explain this improvement in performance across the board? It of course comes from the RL's ability to find optimal actions that are conditioned on the state of the world. In Figure 4 we display the learned strategies for a setting where we have 8 order updates and 8

inventory levels. We show all three stocks, but fix trading volume at 10,000 shares and execution time at 2 minutes. Each panel in Figure 4 corresponds to a single bar in Figure 3, which indicates that we execute our trades with much more precision now. The more inventory and less time remains, the more aggressively we price the orders to avoid submitting a costly market order at the end. We can observe that RL learns the same general "shape" in all cases, but the exact policy specification is stock-dependent.

It is interesting to examine more closely how actions relate to trading costs. In Figure 5 we show how optimal policies are derived from q-values. We fix either the time remaining at 1 (left panel), the last decision point; or the inventory remaining at 4 (right panel), the maximum amount. We then vary the other private variable and plot the Q-values (= trading costs) of each action. In each state the learned action is of course the minimum of the corresponding curve. The shapes of the Q-value functions explain the relationship between state variables and optimal actions: for large inventories and little time remaining, the entire Q-value function shifts upwards, (reflecting higher expected cost), but the minimum of the curve shifts to the right, which indicates that optimal action must be more aggressive in such situations.

4. 2 Introducing Market Variables

We now add market variables to our state space to investigate whether optimal actions should be contingent on market conditions. We summarize our findings in Table 1, which shows the improvement over using just the private variables (averaged over all stocks, sizes, execution periods and private variable resolutions *T* and *I*) for each market variable we tested. *Market order cost* is a measure of liquidity beyond the bid-ask spread – how much would it cost to submit a market order for the balance of inventory immediately instead of waiting until *T*. *Bid-ask volume misbalance* is the signed difference between volumes quoted at the bid and at the ask. *Transaction volume* is the "signed" volume (buy orders are positive, sell orders are negative) of all trades within last 15 seconds. Each variable can take a small number of values. We also explored combinations of 2 and 3 market variables. We note that while we have succeeded in identifying factors that can improve RL performance significantly, we have explored many others that did not result in more efficient policies, which is a testimony to market efficiency. Overall, we can achieve improvement in execution of 50% or more over S&L policies and several times over market orders when we employ RL with both private and market variables.

Bid-Ask Spread	7.97%
Bid-Ask Volume Misbalance	0.13%
Spread + Immediate Cost	8.69%
Immediate Market Order Cost	4.26%
Signed Transaction Volume	2.81%
Spread+ImmCost+Signed Vol	12.85%

Table 1. Additional trading cost reduction when introducing market variables

In order to provide this improvement in performance, the RL algorithm must learn different actions for different values of market variables. Figure 6 is similar in spirit to Figure 4: it shows how optimal actions depend on the market variables *bid-ask spread* and *market order cost*, while keeping the other parameters (*V, H, T,* and *I*) fixed. The learning dictates that larger spreads and lower costs of submitting a market order dictate more aggressive actions --- in the first case because one must go further to "chase" the opposing book, and in the second because an opportunity for reduced price is present. Other useful market variables induce similar dependencies.

As it was the case with *T* and *I* in Figure 5, it is the shape of Q-value functions that is responsible for difference of optimal actions across states. In the left panel of Figure 7, we see how the location of the optimal action on the cost curve shifts to the right as the spread size increases. In the right panel, we show why some market variables do *not* improve the performance of our algorithm: While volume misbalance is most certainly a predictor of future trading costs (as indicated by the difference in Q-value functions), we cannot take advantage of this predictability, since the cost-*minimizing* actions are identical in all three cases.

5. Conclusion and Future Work

We have reported on the first large-scale application of reinforcement learning to the common and challenging finance problem of optimized trade execution. Via careful choice of market state variables and the design of an efficient algorithm exploiting the independence and structure of the problem, we have shown that RL can indeed result in significant improvements over simpler forms of optimization (such as submit and leave policies).

In future work we plan to relax many assumptions made here adapt the methodology and state variables employed here for other common but precisely-defined finance problems, such as market-making and the optimization of constrained strategy classes.

References

Almgren, R., Chriss, N., *Optimal Execution of Portfolio Transactions*, Journal of Risk, 2002.

Bertsimas, D., A, Lo, A., *Optimal Control of Execution Costs*. Journal of Financial Markets 1, 1-50, 1998.

Chan, N., Shelton, C., Poggio, T., *An Electronic Market-Maker*. AI Memo, MIT, 2001.

Coggins, R., Blazejewski, A., Aitken, M., *Optimal Trade Execution of Equities in a Limit Order Market*, International Conference on Computational Intelligence for Financial Engineering, pp. 371-378, March, 2003.

El-Yaniv, R., Fiat, A., Karp, R., Turin, G. *Optimal Search on One-Way Trading Online Algorithms*. Algorithmica 30:101-139, 2001.

Kakade, S., Kearns, M., Mansour, Y., Ortiz, L., *Competitive Algorithms for VWAP and Limit Order Trading*. Proceedings of the ACM Conference on Electronic Commerce, 2004.

Kim, A., Shelton, C., *Modeling Stock Order Flows and Learning Market-Making from Data*. AI Memo 2002-009, MIT, 2002.

Mitchell, T., *Machine Learning*, McGraw Hill, 1997.

Nevmyvaka, Y., Kearns, M., Papandreou, A., Sycara, K, *Electronic Trading in Order-Driven Markets: Efficient Execution*. In the Proceedings of IEEE International Conference on E-Commerce Technology, 2005.

Tesauro, G., and Bredin, J., *Strategic Sequential Bidding in Auctions Using Dynamic Programming*. Proceedings of AAMAS-02.

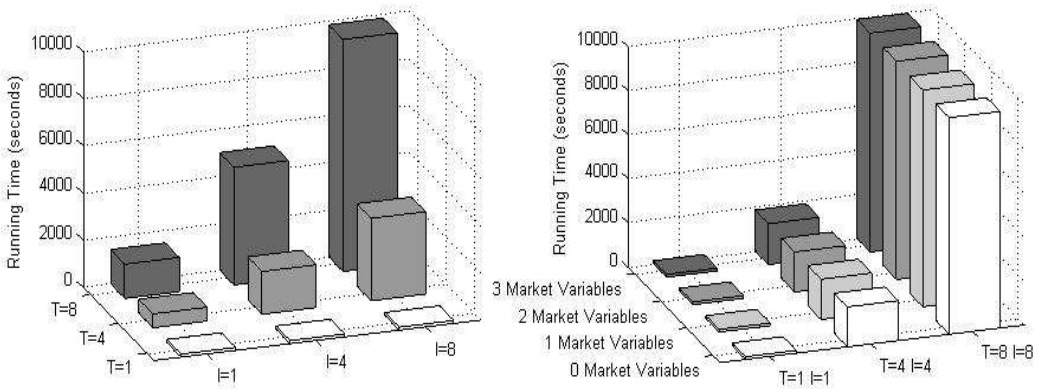

Figure 2. Running time as a function of *T*, *I*, and *R*: independent of the number of market variables

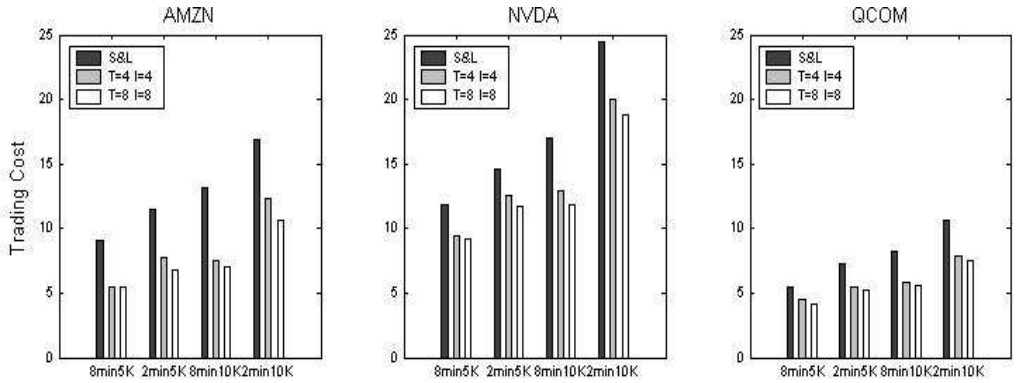

Figure 3. Expected cost under S&L and RL: adding private variables T and I decreases costs

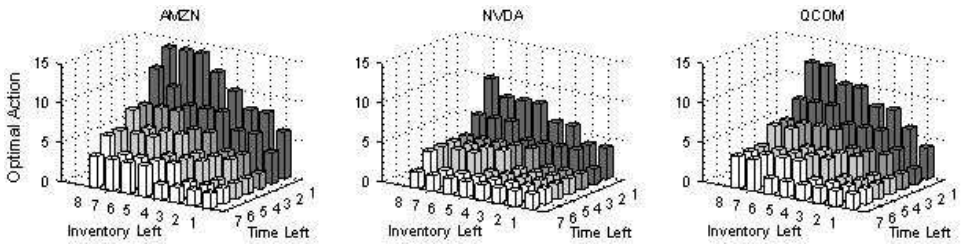

Figure 4. Visualization of learned policies: place aggressive orders as time runs out, significant inventory remains

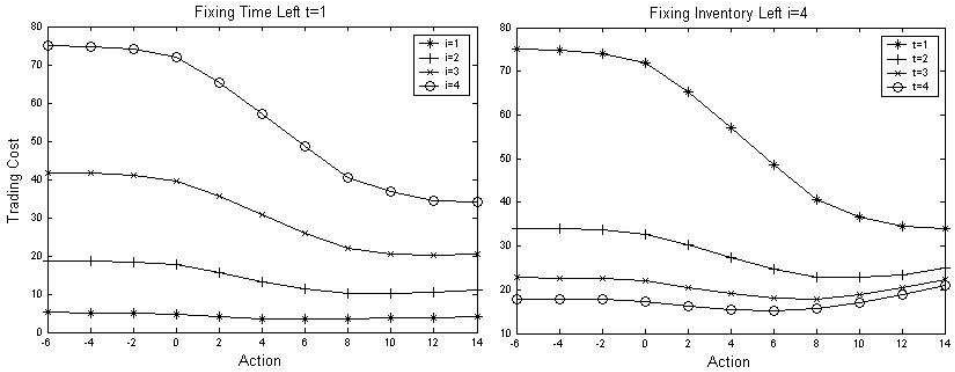

Figure 5. Q-values: curves change with inventory and time (AMZN, T=4, I=4)

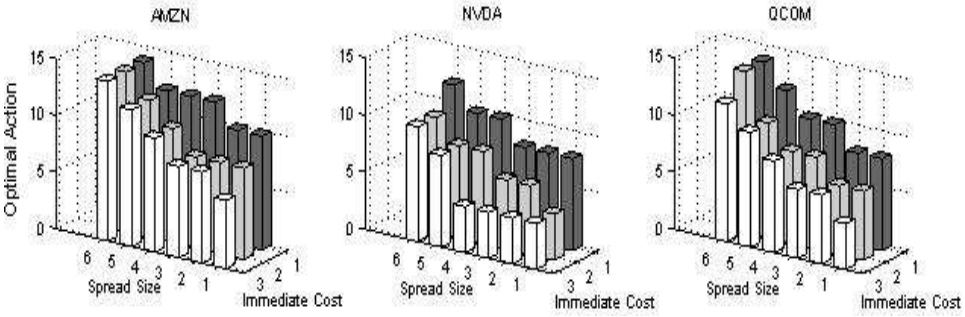

Figure 6. Large spreads and small market order costs induce aggressive actions

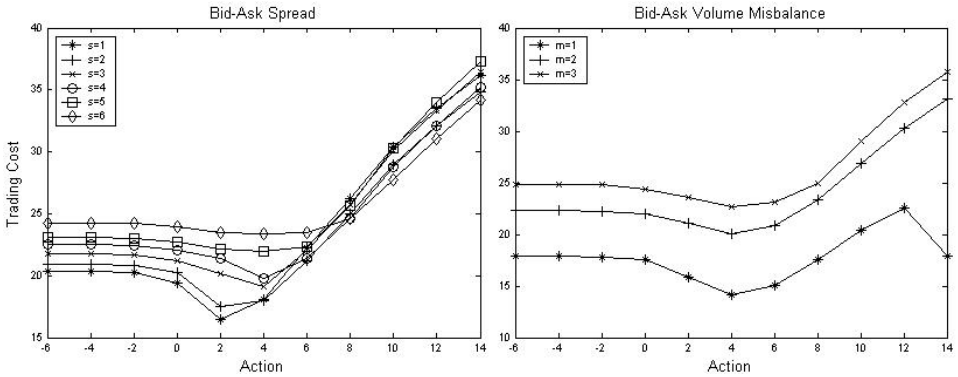

Figure 7. Q-values: cost predictability may not affect the choice of optimal actions