Optimizing stock market execution costs using reinforcement learning

Abdulrahman A. Ahmed, Ayman Ghoneim, Mohamed Saleh
Faculty of Computers and Artificial Intelligence
Cairo University
Cairo, Egypt
a.elsayed@fci-cu.edu.eg, a.ghoneim@fci-cu.edu.eg, m.saleh@fci-cu.edu.eg

***Abstract*—Stock market trading is a complex process where traders aim to maximize their expected return while minimizing associated risks. With the increasing availability of digital historical records, using automated agents for stock market trading becomes of a significant interest. Reinforcement learning is a machine learning branch which circumvents the problem of defining explicit targets and tackles problems which require sequential decisions. Reinforcement learning has been applied in finance problems, yet execution costs optimization problem among others still gets little attention. The optimization of execution costs in stock markets is a vital problem, where a trader wants to minimize the cost of buying a predefined amount of shares over a fixed time horizon. In this study, we propose a novel reinforcement learning Q-trade model to address the execution costs optimization problem. We tested the Q-trade model using historical data of the Egyptian stock market and Nasdaq stock market, and it showed in both markets a significant improvement (more than 60% for some securities) over the compared strategies.**

***Index Terms*—Reinforcement learning, Q-learning, Quantitative finance, Stock markets**

I. Introduction

Due to the increasing complexity of trading in stock markets, automating the trading operations becomes a major aim for individual traders, portfolio managers and financial organizations. Further, the increasing availability of stock markets digital records (i.e., prices and trading volumes) has motivated and directed more research efforts towards automated trading. In spite the fact that forecasting future prices is the fundamental problem in stock markets, other problems are also of essence, since they combine future prices with associated actions to be consequently taken.

Optimizing execution costs (OEC) of shares (also known as optimizing trading costs) is such a problem of significance importance and applicability, where a trader wants to buy a predefined number of shares over a specified time horizon, and this needs to be achieved in minimum costs. The OEC problem was introduced in [1] where several dynamic programming models with ascending factors under consideration were proposed. The first model assumes that the future prices are only affected by past prices and the trader's self-trading activities. The second and third models add components in the future price function to reflect information about other traders' trading activities in the market and external noise. Since then, different modifications and extensions were introduced in subsequent studies. In [2]–[4], different types of risks were incorporated. In [5], the work in [2] was extended to incorporate trade durations and different types of risks in trading. In [6], both discrete and continuous type of trades were combined. The OEC problem has been studied also using other approaches such as differential equations (e.g., [7]–[10]). The dynamic programming approach has its own limitations, as it assumes complete knowledge of the problem model (prices and actions) which doesn't usually holds in real-situations and suffers from the curse of dimensionality as the problem grows in size. However, it is still the most effective way to model the OEC problem when there is no equation-based model for the market dynamics and we depend on historical data.

Reinforcement learning (RL) has been used in computational finance as an alternative approach. RL was combined with neural networks in [11] and genetic algorithms in [12] to develop a trading system. In [13], RL was partially used to learn trading American options. All previously mentioned studies used synthetic data. In [14], RL was used to train agent in the market making problem. The first large scale implementation of RL to solve the OEC problem was in [15], where an algorithm mixing techniques of dynamic programming and RL was proposed for limit order book [16] while using a minimal set of assumptions about the limit order book market structure. The trading policies reached in [15] are better when compared against the buy and hold trading strategy. In [17], RL was applied (similar to [15]) to order book markets based on the model proposed in [2].

To the best of our knowledge, RL was only used to address the OEC problem in order book trading, and no other RL models were presented to address the OEC problem in stock markets using historical prices. In this study, we present a novel RL model (named Q-trade model) to compute trading policies for the OEC problem in stock markets. We tested the Q-trade model using historical records of the Egyptian stock market and the Nasdaq stock market. The proposed Q-trade model results in trading policies which have shown significant improvement (more than 60% for some stocks) over the buy-

and-hold strategy and random strategy.

The paper is organized as follows. In the next section preliminary concepts are covered. In Section III, the proposed Q-trade RL model for the OEC is presented. Experimental results and analysis are given in Section IV. Finally Section V concludes the study and highlights future research directions.

II. Preliminary concepts

In this section, we cover preliminary concepts related to the OEC problem, Markov decision process and RL. Such concepts serve as the foundation for the Q-trade model we propose in this study.

Along with the trading price of a certain security, there are other affecting factors such as commissions, bid/ask spread (i.e., availability of shares) and impact of current trading decisions on future prices. Such costs are known as execution costs as they are associated with executing a specific trading strategy either for buying or selling [1]. In [1], a basic formulation of the OEC problem (excluding the commissions and bid/ask spread factors) states that a trader wants to minimize the costs associated with buying/selling a total amount of shares $\bar{S}$ during a predefined time interval T. The mathematical formulation is given as follows:

$$\begin{array}{ll} \text{Minimize} & \sum_{t=1}^{T} P_t S_t \\ \text{s.t.} & \sum_{t=1}^{T} S_t = \bar{S} \\ & S_t \geq 0, \forall t \in [1,T] \end{array} \tag{1}$$

where the objective is to minimize the sum of multiplication of share price P_t at time step t by the amount of traded shares S_t at time step t. The objective function is constrained that the sum of shares is equal to the total number of shares $\bar{S}$ required to be traded as shown in equation (1). The price P_t is computed using the following equation

$$P_t = P_{t-1} + \theta S_t + \varepsilon_t, \theta > 0, E[\varepsilon_t | S_t, P_{t-1}] = 0, \tag{2}$$

where current value of P_t is calculated in terms of the price P_{t-1} at the previous time step $t-1$ plus a scaled parameter θ multiplied by the shares traded at time step t plus white noise ε_t. The first term in equation (2) captures how P_t is affected by the price in the previous time step. The second term captures the effect of self-trading at time step $t-1$ on the next price at t by multiplying parameter θ (represents the effect of self-trading) by the amount of shares S_t traded at time t. The third term ε_t is a stochastic parameter which represents shocks and unpredicted events in stock market and it has small effect since its expected value is 0.

A. Markov decision process

Markov decision process (MDP) is a discrete stochastic process which expresses a given problem by states, possible associated actions with each state, and a probability of transition from one state to another given the action taken. Formally, an MDP is defined by the tuple $\{S, A, T, R, \gamma\}$ where S is the set of possible states, A is the set of possible actions (i.e., decisions), T represents the transition probabilities, R represents the rewards associated with actions, and γ is a discount factor for future rewards. Given an input tuple describing a certain problem, MDP outputs a policy which dictates what action should an agent take in each particular state in order to maximize the expected reward [18]. The Markov property states that the current state depends only on the previous one, i.e., the next state is determined based only on the actions taken in the previous state. Given the current state $s \in S$ and action $a \in A$ which will be taken in state s, the transition probability $t \to [0,1] \in T$ specifies the probability of moving from the current state s to the next state s' after action a has been taken. The agent will get a reward r where $r = R(s,a,s')$ discounted by $\gamma \to [0,1]$ to prefer near gains over distant ones.

For example, consider a trader who wants to buy n shares of a certain security. In MDP terms, the states will represent the prices of the share (e.g., \$49, \$50, \$51), and assume there are two actions $A = \{0,1\}$ (whether to hold 0 or to buy 1). Assume that the current state/price is $s = \$50$, the trader's action (along with other factors) has a transition probability to whether the market will move to a higher price state or a lower price state. The reward is maximized when the trader buys at the lowest price, however, this reward will be discounted by γ the more the trader postpones the buying action from one time step to the next. Given the transitions probability, the MDP will return a policy which indicates the action to be taken in each price state in order to maximize the future expected reward.

B. Reinforcement Q-learning

Reinforcement learning (RL) is one of the main branches of machine learning (alongside with supervised and unsupervised learning). In supervised learning, the data consists of input vectors and associated target output labels or values, and the goal is to develop a model that given an input vector it results in an output with the minimum deviation (i.e., error) from the target output. In unsupervised learning, data contains input vectors only (no target output), and the goal is to discover the hidden relation patterns of the data points and cluster them into groups depending on similarity measures. On the other hand in RL, the MDP serves as a foundation, where there is an input (represented by actions) and an associated grade of output (represented by the reward). Using the reward received, the RL model learns incrementally the best action to be taken given a particular state [19].

There are two approaches in RL known as model-based RL and model-free RL. Model-based RL aims to learn both the state space and the reward function, while model-free RL learns only the reward function. On-policy and off-policy are two types of algorithms used within model-free RL. On-policy algorithms follow a specific policy at every step (e.g., SARSA algorithm), while off-policy algorithms with ε-greedy policy may choose random action with probability ε and greedy action with probability $1-\varepsilon$. The Q-learning algorithm is one of the widely used model-free off-policy RL algorithms which was first introduced in [20], and we will use it in this study. Algorithm 1 illustrates the steps of the Q-learning algorithm [19].

Algorithm 1 Q-learning Algorithm.

```
1:  Input parameters: step size $\alpha \in (0,1]$, $\varepsilon > 0$.
2:  Initialize $Q(s,a)$, for all $s \in S, a \in A(s)$ arbitrarily except
    that $Q(\text{terminal},\cdot) = 0$.
3:  for episode do
4:      Initialize $s$.
5:      for each step of episode do
6:          Choose $a$ from $A(s)$ using policy derived from
            $Q$ (e.g., $\varepsilon$-greedy).
7:          Take action $a$, observe $r$, $s'$.
8:          $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_a Q(s',a) -$
            $Q(s,a)]$.
9:          $s \leftarrow s'$.
10:     end for
11: end for
```

Recall that $s \in S$ is a state, $s' \in S$ is the next state, S is the set of all states, $a \in A(s)$ is an action to be taken in state s, $A(s)$ is the set of all possible actions at state s, and γ is the discount factor. At each state s, an agent chooses action a which maximizes the Q function given in Step 8 of Algorithm 1, then observe the next state s' and the reward r. The agent then moves to next state s', and repeat the previous for a specified number of iterations. Step 8 of Algorithm 1 is the main part in the Q-learning algorithm which updates the Q value. The new Q value is the the current Q value $Q(s,a)$ plus the second term $\alpha[r+\gamma \max_a Q(s',a) - Q(s,a)]$. The second term is the difference between selecting the action which returns the maximum Q value minus the current Q value multiplied by the discount factor γ (i.e., $\gamma \max_a Q(s',a) - Q(s,a)$) plus the immediate reward r. The second term is multiplied by learning rate α to balance between old experiences and the new ones.

III. Q-trade Model

In this section, we illustrate the details of the proposed Q-trade model to solve the OEC problem. First we discuss the MDP of the Q-trade model in terms of the states, actions, reward and transition probability matrix. We then discuss the modification we introduced to the Q-Learning algorithm to solve the Q-trade model followed by the experimental design.

A. *Q-trade MDP*

Recall that an MDP is defined by the tuple $\{S, A, T, R, \gamma\}$, we will define the MDP of the proposed Q-trade model.

States. A state is described using three variables, which are the price p of the share, the quantity q of purchased shares, and the time step t. The quantity of purchased shares determines the quantity of remaining shares to be purchased. The time step variable is used to indicate where a certain execution (i.e., buying a quantity of shares) stands in the execution horizon. Without loss of generality, we assume that a trader wants to buy a total quantity of $\bar{S} = 1000$ shares. The trader can buy nothing, 500 shares or 1000 shares at a time, thus the quantity of purchased shares has the values $q = \{0, 500, 1000\}$. We assume that the trader can make an early execution or a late execution within the time horizon available to buy the shares, thus the time step has two values $t = \{0, 1\}$. For example, if we have three prices p = \$49, \$50, and \$51 for the share, the set of possible states will be $(p,q,t) \in$ { (49,0,0), (49,500,0), (49,1000,0), (49,0,1), (49,500,1), (49,1000,1),(50,0,0), (50,500,0), (50,1000,0), (50,0,1), (50,500,1), (50,1000,1),(51,0,0), (51,500,0), (51,1000,0), (51,0,1), (51,500,1), (51,1000,1) } . Figure 1 shows the MDP for the example with starting state $(50,0,0)$, and how it is connected to other states through actions.

Actions. The set of action A holds the execution actions (i.e., the amount of shares to be purchased at a time). For simplicity and without loss of generality, in this study we assume discrete action space with three actions which are buy nothing (denoted by a_0), buy 500 shares (denoted by a_1) and buy 1000 shares (denoted by a_2). Figure 1 shows the possible actions that each state can take and its effect. As shown in Figure 1, if the current state is $(50,0,0)$ and action a_0 is executed, then we will move to states where $q = 0$. If action a_1 is executed, then we will move to states where $q = 500$. While if action a_2 is executed, then we will move to states where $q = 1000$. All three actions are allowed in all states, however we discourage infeasible actions through negative rewards as we will illustrate later (e.g., 500 shares are already bought in the current state, the action a_2 (buy 1000 shares) must be discouraged since the trader wants to buy a total quantity of $\bar{S} = 1000$ shares).

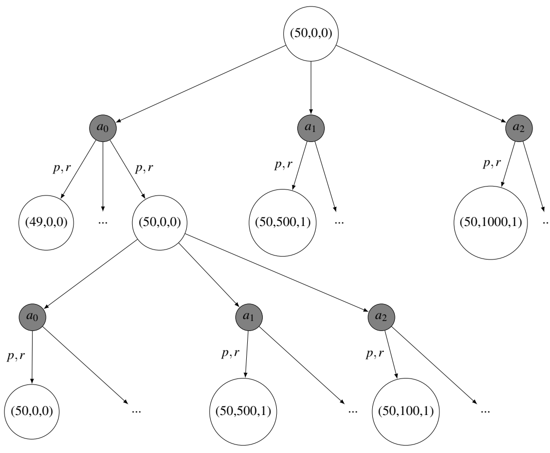

Fig. 1: MDP where white circles represent states, gray circles represent actions, p represents transition probability and r represents the reward.

Rewards. Each action will result into a specific reward. The reward is calculated according to equation (3) which multiplies the quantity purchased $q(a)$ according to action a (where $q(a_0) = 0$, $q(a_1) = 500$, $q(a_2) = 1000$) times negative the share price $-p$ (i.e., this captures the amount paid by the trader to purchase the shares). For example, if the trader chose action a_1 to buy 500 shares and the share price was \$50, then the reward for carrying this action is -25000. Selling can be captured by assuming positive reward (i.e., the amount gained from selling the shares).

$$r = q(a) \times -p \tag{3}$$

All rewards are calculated according to equation (3) except in the following cases:

1) Terminal states which belong to the set $\{(p, 1000, t) | p \in \{prices\}, t \in \{0, 1\}\}$, i.e., the trader already bought the required total quantity of $\bar{S} = 1000$ shares. The reward of a_0 is a large positive value, while the other two actions a_1 and a_2 have large negative reward value to discourage executing these actions in the terminal state and stop buying more shares than $\bar{S}$ shares.
2) States which belong to the set $\{(p, 500, t) | p \in \{prices\}, t \in \{0, 1\}\}$, i.e., the trader already bought 500 shares. The reward of a_2 is a large negative value to discourage executing the action and stop buying more shares than $\bar{S}$ shares.
3) States which belong to the set $\{(p, q, 1) | p \in \{prices\}, q \in \{0, 500, 1000\}\}$, all rewards are reduced by subtracting a predefined value to penalize the delay in buying the required amount of shares $\bar{S}$.

Transition Probability Matrix. To implement the MDP structure explained above, we need to convert historical data to prices and transition probabilities. Algorithm 2 explains how to construct the transition probability matrix (TPM). After data extraction and cleaning (step 1), a security has a sequence of historical prices. In step 2, this sequence of prices is mapped to a predefined interval of $[1, 10]$, where the lowest price of the share is mapped to 1 and the highest price of the security is mapped to 10. In step 3, the interval $[1, 10]$ is discretized into n intervals, and n prices for the security are extracted (each price is the midpoint of an interval). We then create an $n \times n$ matrix M (step 4). In step 5, if a price belongs to interval i is followed in the historical data by a price that belongs to interval j, then we add one in the cell (i, j) of matrix M. We then normalize each row in matrix M, so that the sum of all entries in each row adds up to 1 (step 5), and such matrix M will serve as the TPM matrix (step 6).

Algorithm 2 Building TPM matrix.

```
1: Data extraction and cleaning.
2: Map prices to a defined interval.
3: Discretized the interval and extract $n$ prices.
4: Create $n \times n$ matrix m.
5: Tally matrix $m$ with price movement according to security
   historical prices.
6: Normalize each row in matrix $m$.
7: Use matrix $m$ as a transition probability matrix for the
   MDP.
```

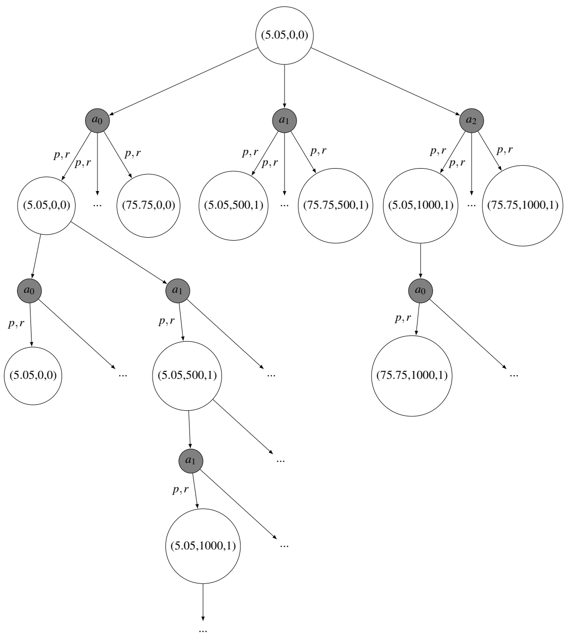

Fig. 2: MDP of the SWDY security.

B. RL Q-learning and experimental design

We use the RL Q-learning algorithm [19] illustrated in subsection II.B. We modify the algorithm to control the number of states per a given sequence, i.e., how long the trader will move from a state to another according to the TPM before stopping and a state is chosen randomly again. This allows the trader to get more exposure to different states instead of being stuck most of iterations in terminal states and learns nothing. We decrease the number of states per sequence in the algorithm to 8 states. We used the Q-learning algorithm implementation provided by the Mdptoolbox [21]. For each security, we carried out 25 runs (i.e., determined 25 trading policies), and each run lasts for 150000 episode with discount factor $\gamma = 0.96$.

IV. Results and Discussion

In this section, we present the results of testing the proposed Q-trade model using Egyptian and Nasdaq stock markets' historical prices. The former is a developing (i.e., emerging) market while the later is a developed market. Since that - and up to our knowledge - there are no RL models for the OEC problem in stock markets available in the literature, we compared the performance of the proposed Q-trade model against two trading strategies, which are the buy and hold strategy, and the random strategy. Both strategies served as benchmarks in previous studies when there are no more sophisticated models to compare against.

A. Results for the Egyptian stock market

We tested the proposed Q-trade model with Egyptian stock market prices dataset. The dataset is comprised of daily records for 37 securities from January 2013 until August 2019. Each record contains the price, highest price reached during the day, lowest price reached during the day, volume of shares traded and the percentage of change in the price. We tested Q-trade model using securities that were traded daily only. To demonstrate the dimensionality of Q-trade model after using securities historical prices, Figure 2 shows the MDP structure for the SWDY security where prices range from 5.05 LE (Egyptian pounds) to 75.75 LE. After discretization, we have 48 prices for the SWDY share, and the total number of states is 288, which is 48 prices multiplied by 3 purchasing quantities (0, 500, 1000) multiplied by two time steps (0,1). Due to the large number of states, Figure 2 illustrates just a portion of the SWDY MDP structure.

In order to evaluate the performance of the trading strategies resulting from the Q-trade model, we compare the trading policies resulting from the Q-trade model against two different strategies. The first one is the buy and hold (B&H) strategy. The B&H strategy starts by buying a quantity of shares at the start of a specified time period, then evaluates the total value at the end of another specified time period (i.e., the remaining amount of money plus the number of shares purchased multiplied by the share current price). For example, if we have \$50,000 in cash and at time step 0 we bought 1000 shares with a price of \$10 per each share. At time step 0, the total value is $\$(1000*10)+\$40000=\$50000$. However, assume at the time step 10, the price per share becomes \$5, thus the total value will be $\$(5*1000)+\$40000=\$45000$.

The second strategy is a random strategy, where shares are bought when a randomly generated number is lower than a predefined threshold. After preliminary analysis, we choose a threshold of 0.75 since lower threshold values led to not buying any shares in many testing periods. The final random strategy value is evaluated at the end of the tested period as the shares value plus the remaining amount of money provided in the beginning. For example, if we have \$50,000 in cash at time step 0, then at every time step a random value is generated less than 0.75, one thousand shares are bought at the time step according to the current share price. At the end of the testing period, one thousand shares are calculated using the current share price plus the remaining cash from the starting time step. The random strategy performance illustrated in the results is the average over 30 different runs.

Table I demonstrates how Q-trade model performs compared to the B&H and random strategies for three different securities, which are SWDY, ORWE, and MTIE. We choose these particular three securities to demonstrate three scenarios where the Q-trade model performed good (for the SWDY security), had unstable performance (for the ORWE security) and performed poorly (for the MTIE security) compared mainly to B&H. The percentage column shows the average performance of the Q-trade model over 25 runs (25 trading policies) compared to B&H and random strategies across different testing periods. A positive percentage means that the Q-trade model performed better than the other strategy, while a negative percentage means that Q-trade model performed worse than the other strategy. Each testing period represents a one day in trading. Q-trade model was evaluated over different testing period which are 8, 15, 30, 60, 90, 120, and 150 testing periods, where number of testing periods represents how many days were considered in each test.

Security	Test period	Average RL	B&H	Imp. %	Random	Imp. %
SWDY	8	96259.54	59923.66	60.64	61111.98	57.51
	15	95305.56	50000	90.61	60823.29	56.69
	30	95319.15	53617.02	77.78	61088.79	56.03
	60	89434.66	50000	78.87	64342.61	39
	90	91155.38	51494.02	77.02	54218.06	68.13
	120	91476.99	57208.59	59.9	54114.62	69.04
	150	91711.94	64104.48	43.07	65182.79	40.7
Average				69.7		55.3
ORWE	8	131000	119230.8	9.87	117019.2	11.95
	15	134800	90000	49.78	117478.9	14.74
	30	100711.5	76923.08	30.92	79761.54	26.27
	60	131496.6	126530.6	3.92	130542.4	0.73
	90	107068.2	65056.82	64.58	71795.45	49.13
	120	105631.4	89701.9	17.76	85947.24	22.9
	150	105173.4	96476.96	9.01	98881.3	6.36
Average				26.55		18.87
MTIE	8	118241.4	82758.62	42.88	114392.4	3.36
	15	130952.4	145238.1	-9.84	131721.9	-0.58
	30	130496.2	140601.5	-7.19	139386.8	-6.38
	60	127493.1	117011.5	8.96	116622.2	9.32
	90	135448.9	141618.5	-4.36	140823.4	-3.82
	120	135920.4	141884.8	-4.2	142246.1	-4.45
	150	140293.2	143874.3	-2.49	142674.3	-1.67
Average				3.39		-0.6

TABLE I: Q-trade model performance compared against the B&H and random strategy for SWDY, ORWE, and MTIE securities.

Table I shows the percentage of improvement for Q-trade over B&H and random strategies. For the SWDY security, Q-trade model has a high improvement compared to the B&H for short testing periods, but as the testing period increases, the Q-trade model reaches the lowest improvement percentage over the B&H at testing period 120. The Q-trade had an average improvement (averaged over the seven testing periods) of 69.70% compared to B&H. For the ORWE security, Q-trade model has low improvement over B&H at 8 testing period and have a small advantage over B&H except in test periods 15 and 90 (where Q-trade had good improvement) to reach an average improvement of 26.55%. For the MTIE security, as the testing period increases, the advantage of Q-trade model over B&H decreases gradually until it drops below the B&H starting from period 15. While Q-trade gained advantage over B&H at testing period 8 and 60, it is beaten again for later testing periods to reach a total average of 3.39%. The behavior of random strategy in most periods is similar to B&H.

While table I illustrates detailed Q-trade model performance for seven testing periods for three securities, Table II shows the average improvement (averaged over the seven different testing periods) for Q-trade model compared to B&H and random strategies for another 34 securities. Q-trade model outperformed B&H and random in most securities, but the improvement is poor in few (e.g. the CIRA security).

Figures 3 and 4 show the Q-trade model performance compared with B&H and random strategies over seven different test periods. In most cases, the Q-trade model has an advantage over the two other strategies. Moreover, the Q-trade model shows more steady performance across testing

Security	Improvement % compared to B&H	Improvement % compared to Random
AMER	42.80	31.92
CIRA	11.50	18.00
ABRD	45.98	40.92
COMI	28.46	18.89
SUGR	54.29	41.04
PHAR	44.37	30.75
ABUK	57.56	37.14
EAST	31.29	15.50
HRHO	23.81	14.37
EGAS	43.08	32.90
TMGH	55.63	35.78
ESRS	63.77	46.39
EHDR	52.20	35.74
ELWA	50.78	31.64
NCCW	60.71	45.98
NEDA	37.66	35.39
NINH	48.25	45.15
OIH	31.40	21.83
ORAS	48.52	45.03
ZMID	41.22	30.20
PORT	41.92	25.16
PRCL	40.41	35.49
QNBA	47.20	43.94
RAKT	57.60	38.95
RREI	66.00	50.56
RTVC	66.99	46.48
SAUD	46.63	34.00
SCEM	45.04	33.35
SDTI	46.61	30.08
SKPC	66.68	47.64
TORA	67.41	49.74
POUL	38.13	31.77
MOIL	38.38	27.11
MNHD	50.54	35.50

TABLE II: Total average (over seven testing periods) of Q-trade model improvement compared to the B&H and random strategy for 34 securities.

periods compared to the steep movements shown by the other strategies. This shows that the Q-trade model is less susceptible to price abrupt changes. For example, COMI security had price lose at testing period 15 compared to testing period 8. While both B&H and random strategies have lost values, Q-trade curve is less steep downward compared to B&H and random curves. On the other hand, when the price had a peak upward for the NINH security, Q-trade curve had much smoother (from test periods 15 to 30) upward movement compared to B&H and random curves.

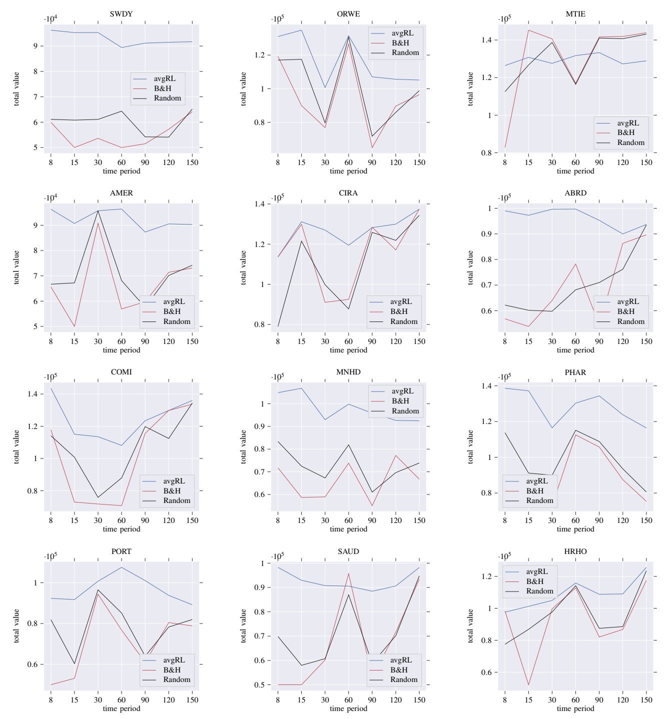

Fig. 3: Average Q-trade model performance in blue compared to B&H strategy in red and random strategy in black over 7 testing periods for SWDY, ORWE, MTIE, AMER, CIRA, ABRD, COMI, MNHD, PHAR, PORT, SAUD and HRHO securities, where the y-axis shows the total value.

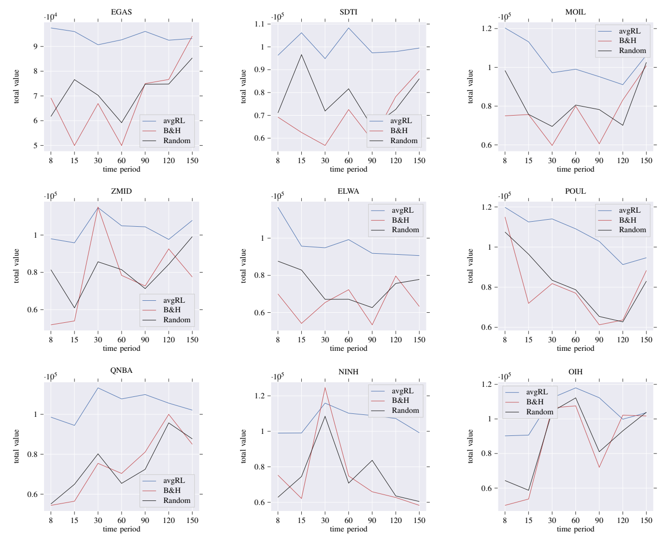

Fig. 4: Average Q-trade model performance in blue compared to B&H strategy in red and random strategy in black over 7 testing periods for EGAS, SDTI, MOIL, ZMID, ELWA, POUL, QNBA, NINH and OIH securities, where the y-axis shows the total value.

B. Results for the Nasdaq stock market

For further investigation, we tested the Q-trade on one of the main global stock markets. We selected the top ten valued companies by market share in Nasdaq stock market. Nasdaq securities are shown in Figure 5 where Q-trade model performance is compare against the B&H and random strategies over seven different test periods. As shown in figure 5 for MSFT security, Q-trade kept advantage over other strategies except for a tangency at 30 and 90 period with B&H strategy. While for AAPL security, Q-trade did very well compared to other strategies until testing period 90 where the differences among the strategies became much narrower. For AMZN security, Q-trade had the same performance of the B&H strategy until period 60, then Q-trade superseded and kept advantage until period 120 to lose its advantage at period 150. Table III shows the total average for the ten securities over seven testing periods for Q-trade improvement compared to B&H and random strategies. The proposed Q-trade model has shown positive average improvement over the B&H and random strategies for all the ten tested securities.

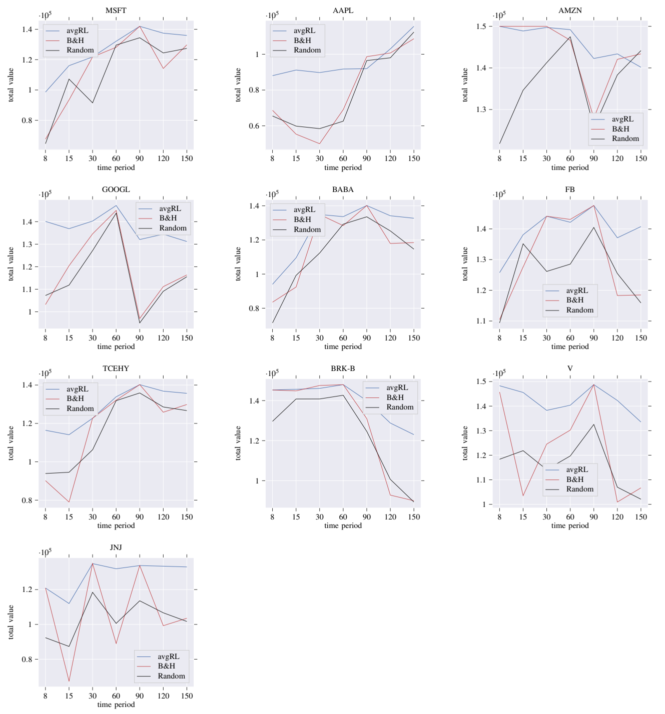

Fig. 5: Average Q-trade model performance in blue compared to B&H strategy in red and random strategy in black over seven testing periods for MSFT, AAPL, AMZN, GOOGL, BABA, FB, TCEHY, BRK-B, V, and JNJ securities where the y-axis shows the total value.

V. Conclusion

OEC problem was addressed in different studies, however most of the proposed solutions lacked practical applicability. RL appears as an alternative approach to solve the OEC problem. While previous RL studies that addressed the OEC

Security	Improvement % compared to B&H	Improvement % compared to Random
MSFT	13.69	17.67
AAPL	29.6	27.17
AMZN	1.52	7.86
GOOGL	17.85	20.22
BABA	8.52	14.16
FB	7.5	9.33
TCEHY	13.35	12.90
BRK-B	12.42	16.03
V	16.73	19.19
JNJ	24.53	23.25

TABLE III: Total average (over seven testing periods) of Q-trade model improvement compared with the B&H and random strategy for the top ten Nasdaq securities.

problem had used limit order book data structure to build their models, in this study we proposed a novel Q-trade model to solve the OEC problem for stock markets. We showed that Q-trade is a promising model while using a basic RL Q-learning algorithm with a simple MDP structure. We tested the Q-trade model using the Egyptian stock market and Nasdaq stock market historical prices and showed that the Q-trade model has a significant improvement (in both markets) over B&H and random strategies trading strategies in most of the tested securities (up to 60% for some securities). There are fruitful avenues to build upon the current model including: introducing new variables to the MDP such as traded volume to capture more closely stock markets dynamics, enhancing the design of the reward function, and using approximate Q function as a better representation (e.g., deep learning techniques).

References

[1] D. Bertsimas and A. Lo, "Optimal control of execution costs," *Journal of Financial markets*, 1998.

[2] R. Almgren and N. Chriss, "Optimal execution of portfolio transactions," *Journal of Risk*, 2001.

[3] R. Almgren, "Optimal execution with nonlinear impact functions and trading enhanced risk," *Applied Mathematical Finance*, 2003.

[4] J. Lorenz and R. Almgren, "Mean-variance optimal adaptive execution," *Applied Mathematical Finance*, 2011.

[5] G. Huberman and W. Stanzl, "Optimal liquidity trading," *Review of Finance*, 2005.

[6] A. Obizhaeva and J. Wang, "Optimal trading strategy and supply/demand dynamics," *Journal of Financial markets*, 2013.

[7] A. Schied and T. Schoneborn, "Risk aversion and the dynamics of optimal liquidation strategies in illiquid markets," *Finance and Stochastics*, 2009.

[8] R. Almgren, "Optimal trading with stochastic liquidity and volatility," *SIAM Journal on Financial Mathematics*, 2012.

[9] G. Huberman and W. Stanzl, "Optimal trade execution: a mean quadratic variation approach," *Quantitative Finance*, 2009.

[10] O. Gueant, "Optimal execution and block trade pricing: a general framework," *Applied Mathematical Finance*, 2015.

[11] J. Moody and M. Saffell, "Learning to trade via direct reinforcement," *IEEE Transactions on Neural Networks*, 2001.

[12] A. Hryshko and T. Downs, "System for foreign exchange trading using genetic algorithms and reinforcement learning," *International Journal of Systems Science*, 2004.

[13] Y. Li, C. Szepesvari, and D. Schuurmans, "Learning exercise policies for american options," in *International Conference on Artificial Intelligence and Statistics (AISTATS09)*, 2009.

[14] T. Sponeer, J. Fearnley, R. Savani, and A. Koukorinis, "Market making via reinforcement learning," in *In proceedings of AAMAS*, 2018.

[15] Y. Nevymvaka, Y. Feng, and M. Kearns, "Reinforcement learning for optimized trade execution," in *23rd international conference on machine learning*, 2006.

[16] M. D. Gould, M. A. Porter, S. Williams, M. McDonald, D. J. Fenn, and S. D. Howison, "Limit order books," 2013.

[17] D. Hendricks and D. Wilcox, "A reinforcement learning extension to the almgren-chriss framework for optimal trade execution," in *IEEE Conference on Computational Intelligence for Financial Economics and Engineering*, 2014.

[18] M. Puterman, *Markov Decision Process*. Wiley-Interscience, 1994.

[19] A. G. B. R. Sutton, *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[20] C. Watkins, "Learning from delayed rewards," Ph.D. dissertation, Cambridge University, 1989.

[21] Chades, Iadine, G. Chapron, M. Cros, F. Garcia, and R. Sabbadin, "Mdptoolbox: a multi-platform toolbox to solve stochastic dynamic programming problems," in *Ecography*, 2014.