A Multiagent Approach to Q-Learning for Daily Stock Trading

Jae Won Lee, Jonghun Park, *Member, IEEE*, Jangmin O, Jongwoo Lee, and Euyseok Hong

***Abstract*—The portfolio management for trading in the stock market poses a challenging stochastic control problem of significant commercial interests to finance industry. To date, many researchers have proposed various methods to build an intelligent portfolio management system that can recommend financial decisions for daily stock trading. Many promising results have been reported from the supervised learning community on the possibility of building a profitable trading system. More recently, several studies have shown that even the problem of integrating stock price prediction results with trading strategies can be successfully addressed by applying reinforcement learning algorithms. Motivated by this, we present a new stock trading framework that attempts to further enhance the performance of reinforcement learning-based systems. The proposed approach incorporates multiple Q-learning agents, allowing them to effectively divide and conquer the stock trading problem by defining necessary roles for cooperatively carrying out stock pricing and selection decisions. Furthermore, in an attempt to address the complexity issue when considering a large amount of data to obtain long-term dependence among the stock prices, we present a representation scheme that can succinctly summarize the history of price changes. Experimental results on a Korean stock market show that the proposed trading framework outperforms those trained by other alternative approaches both in terms of profit and risk management.**

***Index Terms*—Financial prediction, intelligent multiagent systems, portfolio management, Q-learning, stock trading.**

I. Introduction

BUILDING an intelligent system that can produce timely stock trading suggestions has always been a subject of great interest for many investors and financial analysts. Nevertheless, the problem of finding out the best time to buy or sell has remained extremely hard since there are too many factors that may influence stock prices [1]. The famous "efficient market hypothesis" (EMH), which was tested in the economics over a 40-year period without definitive findings, states that no investment system can consistently yield average returns exceeding the average returns of a market as a whole. Throughout many years, finance theoreticians argue for EMH as a basis of denouncing the techniques that attempt to find useful information about the future behavior of stock prices by using historical data [2].

However, the assumptions underlying this hypothesis turns out to be unrealistic in many cases [3], and in particular, most approaches taken to testing the hypothesis were based on linear time series modeling [4]. Accordingly, as claimed in [4], given enough data and time, an appropriate nonparametric machine learning method may be able to discover more complex nonlinear relationships through learning from examples. Furthermore, if we step back from being able to "consistently" beat the market, we may find many interesting empirical results indicating that the market might be somehow predictable [5].

Indeed, the last decade has witnessed the abundance of such approaches to financial analysis both from academia and industry. Application of various machine learning techniques to stock trading and portfolio management has experienced significant growth, and many trading systems have been proposed in the literature based on different computational methodologies and investment strategies [6]–[10]. In particular, there has been a huge amount of interest in the application of neural networks to predict the stock market behavior based on current and historical data, and this popularity continues mainly due to the fact that the neural networks do not require an exact parametric system model and that they are relatively insensitive to unusual data patterns [3], [11].

More recently, numerous studies have shown that even the problem of integrating stock price prediction results with dynamic trading strategies to develop an automatic trading system can be successfully addressed by applying reinforcement learning algorithms. Reinforcement learning provides an approach to solving the problem of how an autonomous agent that senses and acts in its environment can learn to choose optimal actions to achieve its goals [12]. Compared with the supervised learning techniques such as neural networks, which require input and output pairs, a reinforcement learning agent learns behavior through trial-and-error interactions with a dynamic environment, while attempting to compute an optimal policy under which the agent can achieve maximal average rewards from the environment.

Hence, considering the problem characteristics of designing a stock trading system that interacts with a highly dynamic stock market in an objective of maximizing profit, it is

Manuscript received August 5, 2005; revised February 21, 2006. This work was supported by a research grant (2004) from Sungshin Women's University. This paper was recommended by Associate Editor R. Subbu.

J. W. Lee and E. Hong are with the School of Computer Science and Engineering, Sungshin Women's University, Seoul 136-742, Korea (e-mail: jwlee@sungshin.ac.kr; hes@sungshin.ac.kr).

J. Park (corresponding author) is with the Department of Industrial Engineering, Seoul National University, Seoul 151-742, Korea (e-mail: jonghun@snu.ac.kr).

J. O was with the School of Computer Science and Engineering, Seoul National University, Seoul 151-742, Korea. He is now with NHN Corporation, Seongnam 463-811, Korea (e-mail: rupino11@naver.com).

J. Lee is with the Department of Multimedia Science, Sookmyung Women's University, Seoul 140-742, Korea (e-mail: bigrain@sookmyung.ac.kr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMCA.2007.904825

worth considering a reinforcement learning algorithm such as Q-learning to train a trading system. There have been several research results published in the literature along this line. Neuneier [8] used a Q-learning approach to make asset allocation decisions in financial market, and Neuneier and Mihatsch [13] incorporated a notion of risk sensitivity into the construction of Q-function. Another portfolio management system built by use of Q-learning was presented in [14] where absolute profit and relative risk-adjusted profit were considered as performance functions to train a system. In [15], an adaptive algorithm, which was named recurrent reinforcement learning, for direct reinforcement was proposed, and it was used to learn an investment strategy online. Later, Moody and Saffell [16] have shown how to train trading systems via direct reinforcement. Performance of the learning algorithm proposed in [16] was demonstrated through the intraday currency trader and monthly asset allocation system for S&P 500 stock index and T-Bills.

In this paper, we propose a new stock trading framework that attempts to further enhance the performance of reinforcement learning-based systems. The proposed framework, which is named MQ-Trader, aims to make buy and sell suggestions for investors in their daily stock trading. It takes a multiagent approach in which each agent has its own specialized capability and knowledge, and employs a Q-learning algorithm to train the agents. The motivation behind the incorporation of multiple Q-learning agents is to enable them to effectively divide and conquer the complex stock trading problem by defining necessary roles for cooperatively carrying out stock pricing and selection decisions. At the same time, the proposed multiagent architecture attempts to model a human trader's behavior as closely as possible.

Specifically, MQ-Trader defines an architecture that consists of four cooperative Q-learning agents: The first two agents, which were named buy and sell signal agents, respectively, attempt to determine the right time to buy and sell shares based on global trend prediction. The other two agents, which were named buy and sell order agents, carry out intraday order executions by deciding the best buy price (BP) and sell price (SP), respectively. Individual behavior of the order agents is defined in such a way that microscopic market characteristics such as intraday price movements are considered. Cooperation among these proposed agents facilitates efficient learning of trading policies that can maximize profitability while managing risks effectively in a unified framework.

One of the important issues that must be addressed when designing a reinforcement learning algorithm is the representation of states. In particular, the problem of maintaining the whole raw series of stock price data in the past to compute long-term correlations becomes intractable as the size of considered time window grows large. Motivated by this, we propose a new state representation scheme, which is named turning point (TP) matrix, that can succinctly summarize the historical information of price changes. The TP matrix is essentially a binary matrix for state representation of the signal agents. Furthermore, in MQ-Trader, various technical analysis methods such as short-term moving averages (MAs) and Japanese candlestick representation [17] are utilized by the order agents.

In Section II, we present the architecture of the proposed framework, describe how cooperation among the trading agents in MQ-Trader is achieved, and subsequently define the state representation schemes. Section III presents learning algorithms for the participating agents after briefly introducing basic concepts of Q-learning. Experimental setup and results on a real Korean stock market, i.e., Korea Composite Stock Price Index (KOSPI), are described in Section IV. Finally, Section V concludes this paper with discussion on future research directions.

II. Proposed Framework for Multiagent Q-Learning

In this section, we first present the proposed MQ-Trader framework that employs cooperative multiagent architecture for Q-learning. After describing the behavior of individual agents during the learning process, this section proceeds to define the necessary state representations for the agents. Detailed learning algorithms are presented in Section III.

A. Proposed Learning Framework

In an attempt to simulate a human investor's behavior and at the same time to divide and conquer the considered learning problem more effectively, MQ-Trader defines four agents. First, a stock trading problem is divided into the timing and the pricing problem of which the objectives are, respectively, to determine the best time and the best price for trading. This naturally leads to the introduction of the following two types of agents: 1) the signal agent and 2) the order agent.

Second, motivation for the separation of the buy signal agent from the sell signal agent comes from the fact that an investor has different criteria for decision making depending on whether she/he buys or sells a stock. When buying a stock, the investor usually considers the possibility of rising and falling of the stock price. In contrast, when selling a stock, the investor considers not only the tendency of the stock price movements but also the profit or loss incurred by the stock. Accordingly, the separation is necessary to allow the agents to have different state representations. That is, while the buy signal agent maintains the price history information as its state to estimate future trend based on the price changes over a long-term period, the sell signal agent needs to consider the current profit/loss obtained in addition to the price history.

Finally, the buy order and the sell order agents, respectively, generate orders to buy and sell a stock at some specified price. These are called bid and offer. The objective of these order agents is to decide the best price for trading within a single day in an attempt to maximize profit.

Fig. 1 shows the overall learning procedure defined in MQ-Trader. It aims to maximize the profit from investment by considering the global trend of stock price as well as the intraday price movements. Under this framework, each agent has its own goal while interacting with others to share episodes throughout the learning process.

More specifically, given a randomly selected stock item, an episode for learning is started by randomly selecting a

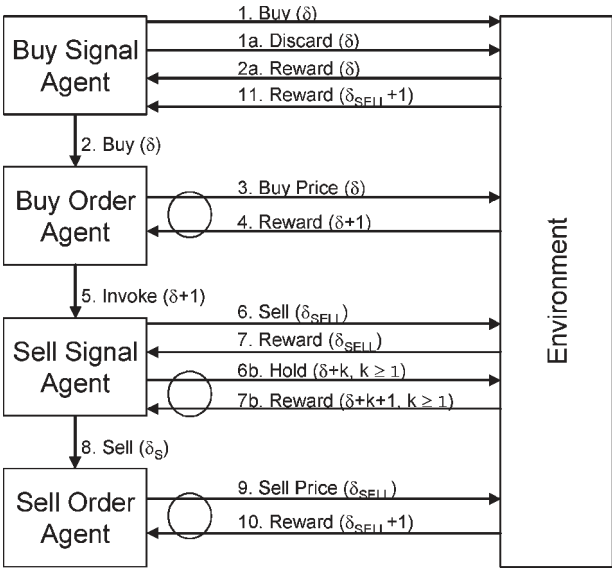

Fig. 1. Learning procedure of MQ-Trader.

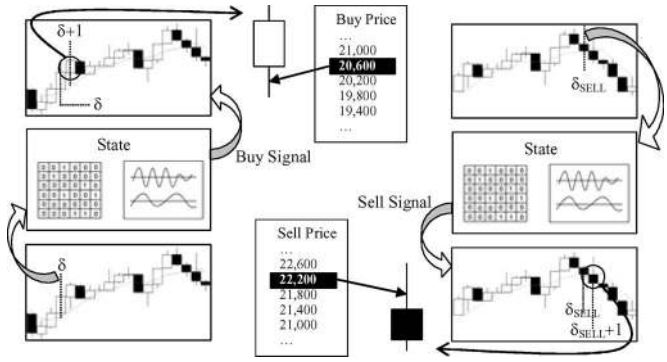

Fig. 2. Sample episode.

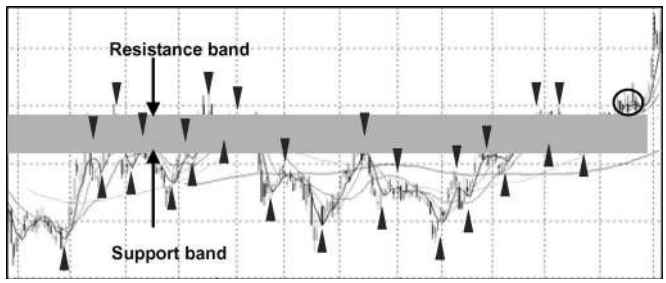

Fig. 3. Example of plots representing five-day MAs of closing prices.

certain day in the history, which is denoted by δ, from the environment. As shown in Fig. 1, the buy signal agent first makes price prediction by analyzing recent price movements of the considered stock and then makes a stock purchase decision based on whether the price is likely to rise in near future or not. When it decides not to buy the stock, the episode is ended, and a new episode is started at another randomly chosen day. The steps denoted as 1a and 2a in Fig. 1 correspond to this case.

On the other hand, if the decision is to buy the stock on day δ, the buy order agent, which first checks the feasibility of the purchase by consulting the maximum allowed BP, is informed. Subsequently, if the purchase is feasible, the buy order agent makes an actual purchase on $\delta + 1$ after it determines an appropriate BP. In some cases, BP might be set too low, resulting to an unsuccessful purchase. When this happens, the buy order agent tries different BPs until a successful purchase is made. The circle encompassing steps 3 and 4 in Fig. 1 represents this looping behavior. A reward is given by the environment to the buy order agent based on how much BP is close to the optimum, which is the lowest possible BP for successful purchase on day $\delta + 1$.

After the purchase is made on day $\delta + 1$, the sell signal agent examines the recent price history of the purchased stock as well as the current profit or loss accrued in order to decide either to hold the stock or to sell on each day starting from $\delta + 1$. When the sell signal agent decides to hold the stock on a day $\delta + k$, where $k \geq 1$, as indicated by step 6b in Fig. 1, the environment provides it with a reward and an updated state so that the same process can repeat on the next day. Otherwise, in case that the stock is sold at the SP valuated by the sell order agent, the sell order agent is provided with a reward in a way similar to the case of the buy order agent on day $\delta_{\mathrm{SELL}} + 1$, where δ_{SELL} indicates the day when the sell signal agent decides to sell the stock. Finally, a single episode ends after the environment notifies the buy signal agent of the resulting profit rate as a reward. A sample episode is illustrated in Fig. 2. We remark that an infinite loop in which a stock is indefinitely held is prevented by defining maximum possible days during which the sell signal agent may hold a stock. Detailed definitions of the state, action, and reward of each agent in MQ-Trader are given in the following sections.

B. State Representations for Signal Agents

One of the most important issues in defining a Q-learning framework is the representation of state for agent. Indeed, the development of an effective and efficient reinforcement learning system is an art of designing state and reward representations. In this section, we present the proposed state representations for the signal agents.

As discussed in the previous section, the signal agent is responsible for determining the day when a stock is bought or sold, and it maintains price history information of the stock in consideration as its state for being able to predict future trend. Furthermore, it was mentioned in the previous section that the sell signal agent defines additional state that summarizes profit or loss that occurred during an episode.

In order to efficiently represent the price change history of a stock over a long-term period, we utilize the notion of resistance and support instead of taking the whole raw price data as a state, which is computationally extensive for training agents. Specifically, we propose a state representation scheme, which is called the TP matrix, which can succinctly summarize the history of stock price changes over a long period.

A TP is a local extremal point in the plots generated by computing five-day MAs of closing prices. When it is a local minimum, it is called an upward TP, and similarly when it is

a local maximum, it is called a downward TP. The sequence of TPs shows the history of resistance to and support for stock price changes, and it has implications on the future price movements. For instance, the existence of a downward TP at the price of 100 for a stock in the past may be an indication that the future stock price is not likely to rise beyond 100. An example of plots representing the five-day MAs is shown in Fig. 3, where TPs are depicted as arrowheads.

A TP matrix $M = [A/B]$ is a partitioned matrix in which submatrices A and B are binary valued square matrices of size n. An element of M represents an existence of TP with specified properties that are defined for columns and rows. The columns of M represent time windows, and they are defined by the use of Fibonacci numbers $F_0, F_1, \ldots, F_n$, where $F_0 = 0$, $F_1 = 2$, and $F_2 = 3$, in such a way that the jth column corresponds to the time period $(\sum_{k=0}^{j-1} F_k + 1, \sum_{k=0}^{j} F_k)$ in the past. Given a time window (x, y) in the past, x represents the xth day when the days are counted backward starting from day D, which is a reference day on which the signal agent makes a decision. That is, the first column indicates the time window containing the yesterday and the day before yesterday.

On the other hand, the rows of M represent the ranges of price change ratio of a stock on day D with respect to the price at TP, which is defined as $C_{\mathrm{TP},D} = (P^C_{\mathrm{TP}} - P^C_D)/P^C_D$, where P^C_{TP} and P^C_D, respectively, indicate the closing prices of a stock on days TP and D. Similar to the case of time window definition, the whole range of the possible price change ratio is subdivided into the distinct intervals according to Fibonacci numbers. In particular, submatrix A represents the case in which price has not fallen on day D compared to that of TP, i.e., $C_{\mathrm{TP},D} \leq 0$, whereas submatrix B represents the opposite case. Therefore, it follows that the first row of A corresponds to the price increase within the range of 0% to 2%.

Each element $a_{ij} \in A$, $i = 1, \ldots, n$ and $j = 1, \ldots, n$, is formally defined as shown at the bottom of the page.

	2	3	5	8	13	21	34	55	89
-2	0	0	0	0	0	0	0	0	0
-3	0	0	0	0	0	0	0	0	0
-5	0	0	0	0	0	0	0	0	0
-8	0	0	0	0	1	0	0	0	1
-13	0	0	0	0	0	0	0	0	1
-21	0	0	0	0	0	1	0	1	1
-34	N	0	0	0	0	0	1	1	1
-55	N	N	0	0	0	0	0	0	0
-89	N	N	N	N	0	0	0	0	0
+2	0	0	0	0	0	0	0	0	0
+3	0	0	0	0	0	0	0	0	0
+5	0	0	0	0	0	0	0	0	0
+8	0	0	0	0	0	0	0	0	0
+13	0	0	0	0	0	0	0	0	0
+21	0	0	0	0	0	0	0	0	0
+34	0	0	0	0	0	0	0	0	0
+55	0	0	0	0	0	0	0	0	0
+89	N	N	N	N	0	0	0	0	0

Upward TP matrix

	2	3	5	8	13	21	34	55	89
-2	0	0	0	0	0	0	0	0	0
-3	0	0	0	0	0	0	0	0	0
-5	0	0	0	0	0	0	0	0	0
-8	0	0	0	0	0	0	0	0	0
-13	0	0	0	1	1	0	0	0	1
-21	0	0	0	0	0	1	0	0	1
-34	0	0	0	0	0	0	1	1	1
-55	N	0	0	0	0	0	1	1	1
-89	N	N	0	0	0	0	0	0	0
+2	0	0	0	0	0	0	0	0	0
+3	0	0	0	0	0	0	0	0	0
+5	0	0	0	0	0	0	0	0	0
+8	0	0	0	0	0	0	0	0	0
+13	0	0	0	0	0	0	0	0	0
+21	0	0	0	0	0	0	0	0	0
+34	N	0	0	0	0	0	0	0	0
+55	N	0	0	0	0	0	0	0	0
+89	N	N	N	N	0	0	0	0	0

Downward TP matrix

Fig. 4. Example of TP matrices.

The elements $b_{ij} \in B, i = 1, \ldots, n$ and $j = 1, \ldots, n$, are similarly defined as in the case of submatrix A except that the condition on $C_{\mathrm{TP},D}$ is replaced with $C_{\mathrm{TP},D} > 0$ for B.

The rationale behind the employment of Fibonacci numbers to subdivide the time windows as defined previously is to pay more interest in recent history. TPs in the recent past are considered by use of several time windows of small size, whereas TPs in the distant past are aggregated by use of a few windows of large size. Similarly, TPs with small price differences from P^C_D receives more attention than those with big price differences since they resemble more closely the situation of day D. Fig. 4 shows an example of the upward and downward TP matrices for the region circled in Fig. 3.

In Fig. 4, the past 230 days are considered, and accordingly, n is set to 9 to make the last time window include the 230th day in the past counted backward from D. From the definition of the TP matrix, it follows that the Fibonacci number associated with each column represents the size of a time window in terms of days, and it also follows that the starting day for the time

(Case 1) $1 \leq i < n$

$$a_{ij} = \begin{cases} 1, & \text{if there exists a TP such that } C_{\mathrm{TP},D} \leq 0 \text{ and } \sum_{k=0}^{i-1} F_k \leq |C_{\mathrm{TP},D}| \times 100 < \sum_{k=0}^{i} F_k \\ & \text{during the period } \left(\sum_{k=0}^{j-1} F_k + 1, \sum_{k=0}^{j} F_k\right) \\ 0, & \text{otherwise} \end{cases}$$

(Case 2) $i = n$

$$a_{ij} = \begin{cases} 1, & \text{if there exists a TP such that } C_{\mathrm{TP},D} \leq 0 \text{ and } \sum_{k=0}^{i-1} F_k \leq |C_{\mathrm{TP},D}| \times 100 < \infty \\ & \text{during the period } \left(\sum_{k=0}^{j-1} F_k + 1, \sum_{k=0}^{j} F_k\right) \\ 0, & \text{otherwise} \end{cases}$$

TABLE I
SAMPLE ENCODING SCHEME FOR PROFIT RATIO

Profit ratio	Encoding	Profit ratio	Encoding
+[0 ~ 5)	00000001	-(0 ~ 5]	00010000
+[5 ~ 12)	00000010	-(5 ~ 12]	00100000
+[12 ~ 20)	00000100	-(12 ~ 20]	01000000
+[20 ~)	00001000	-(20 ~)	10000000

window is equal to the sum of all preceding Fibonacci numbers plus one. For instance, the third column of the matrix in Fig. 4 corresponds to the time window of five days that starts six days ago and ends ten days ago. Finally, an element marked as "N" in Fig. 4 indicates that it is not allowed to make such a price change within the corresponding time period due to the regulation of a stock market considered.

In addition to the TP matrix, the sell signal agent has a few more bits as its state to represent profit rate obtained by holding a stock during an episode. The profit rate on day D, i.e., PR_D, is defined as follows:

$$PR_D = \frac{P_D^C - BP}{BP} \times 100.$$

Finally, in order to encode the value of PR_D as bits of fixed length, we divide the whole range of possible profit ratio into the intervals and map a bit to each interval to indicate whether or not a profit ratio belongs to the specific interval. Table I shows a sample case in which 8 bits are used for representing the profit ratio. Under the encoding scheme presented in Table I, a profit ratio of $+15\%$, for example, will be represented as 00000100.

C. State Representations for Order Agents

The objectives of buy order and sell order agents are, respectively, to figure out optimal bid and ask prices of orders for a specific trading day. In contrast to the signal agents that utilize the long-term price history information to predict future stock price movements, the order agents need to learn the characteristics of intraday stock price changes. For this purpose, the proposed framework bases its state representation for the order agents on the Granville's law [18] and Japanese candlesticks, which are popular methods for short-term technical analysis.

Granville's law is a widely used method that considers the correlations among the long-term and short-term MAs of closing prices in order to predict the short-term price movements. According to Granville's law, the short-term temporary behavior of stock price changes eventually resembles the long-term behavior, and therefore, a temporary deviation from the long-term behavior can be identified as an indicator that the behavior in the upcoming short period will soon follow the long-term behavior.

We apply this principle to the problem of estimating the trend of intraday stock price movements by introducing necessary indicators to the state representations of order agents as follows.

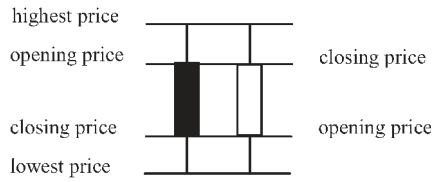

Fig. 5. Japanese candlestick representation.

An MA is an indicator that shows the average value of stock prices over a certain period of time. An arithmetic N-day MA on a trading day D, i.e., MA_D^N, is defined as

$$\mathrm{MA}_D^N = \frac{\sum_{i=D-N+1}^{D} P_i^C}{N}$$

where $D \geq N$ and P_i^C is the closing price of a considered stock on the ith trading day such that $i = D - N + 1, \ldots, D$. We define two indicators that can capture the characteristics of short-term price changes and incorporate them into the state representation for the order agents. First, a gradient of the N-day MA on day D, i.e., g_D^N, is defined as

$$g_D^N = \frac{\mathrm{MA}_D^N - \mathrm{MA}_{D-1}^N}{\mathrm{MA}_{D-1}^N}.$$

Second, the normalized distance between P_D and MA_D^N, i.e., d_D^N, is defined as follows:

$$d_D^N = \frac{P_D^C - \mathrm{MA}_D^N}{\mathrm{MA}_D^N}.$$

Following the Granville's law, g_D^N and d_D^N can be used to derive some sufficient conditions to make predictions on the price movements on day $D+1$. When $g_D^N > 0$ (i.e., a bull market), the stock price is likely to rise on day D, and the value of d_D^N will normally be positive. However, if d_D^N happens to have a negative value for a bull market, it is quite likely that it is an indication of price rise on day $D+1$. Furthermore, if the value of d_D^N is too high, the stock price is expected to fall on $D+1$. Similar arguments can be made for the case when $g_D^N < 0$ (i.e., a bear market).

Fig. 5 shows a standard representation of Japanese candlesticks. In this representation, a black bar indicates that the closing price of a stock is lower than the opening price on a trading day, whereas a white bar indicates the opposite case. Top line and bottom line of the candlestick, respectively, denote the highest price and the lowest price on a trading day.

The shape of a candlestick conveys important information for determining BP or SP. Accordingly, in MQ-Trader, the data contained in the Japanese candlestick are represented as a state for the order agents in terms of the following four indicators: 1) the body b_D; 2) upper shadow u_D; 3) lower shadow l_D; and 4) ratio of closing price difference q_D that are formally defined as follows. Let P_D^O, P_D^H, and P_D^L, respectively, denote

the opening, highest, and lowest price of a stock on a trading day D. Detailed definitions are given as follows:

$$b_D = \frac{P_D^C - P_D^O}{P_D^O}$$

$$u_D = \frac{P_D^H - \max\left(P_D^O, P_D^C\right)}{\max\left(P_D^O, P_D^C\right)}$$

$$l_D = \frac{\min\left(P_D^O, P_D^C\right) - P_D^L}{\min\left(P_D^O, P_D^C\right)}$$

$$q_D = \frac{P_D^C - P_{D-1}^C}{P_{D-1}^C}.$$

III. Learning Algorithms for MQ-Trader Agents

Q-learning is an incremental reinforcement learning method that does not require a model structure for its application. The objective of the Q-learning agent is to learn an optimal policy, i.e., a mapping from a state to an action that maximizes the expected discounted future reward, which is represented as a value function Q. One-step Q-learning is a simple algorithm in which the key formula to update the Q value to learn an optimal policy is defined as follows [12]:

$$Q(s_t, \alpha_t) \leftarrow Q(s_t, \alpha_t) + \lambda\left[r(s_t, \alpha_t) + \gamma \max_{\alpha} Q(s_{t+1}, \alpha) - Q(s_t, \alpha_t)\right]$$

where $Q(s_t, \alpha_t)$ is a value function defined for a state–action pair (s_t, α_t) at moment t, λ and γ are the learning rate and discount factor, respectively, and $r(s_t, \alpha_t)$ is a reward received as a result of taking action α_t in state s_t.

When the state space to be explored by an agent is large, it is necessary to approximate the Q value. One of the most commonly used approaches to the approximation is a gradient-descent method in which the approximated Q value at t, i.e., $\widehat{Q}_t$, is computed by use of a parameterized vector with a fixed number of real valued components, which is denoted as $\overrightarrow{\theta_t}$. Specifically, the function approximation in the proposed framework is carried out by use of a neural network in which link weights correspond to $\overrightarrow{\theta_t}$. In this framework, $\overrightarrow{\theta_t}$ is updated by the following expression, where the gradient $\nabla_{\overrightarrow{\theta_t}} \widehat{Q}_t(s_t, \alpha_t)$ can be computed by use of the backpropagation algorithm [19]:

$$\overrightarrow{\theta}_{t+1} \leftarrow \overrightarrow{\theta_t} + \lambda \nabla_{\overrightarrow{\theta_t}} \widehat{Q}_t(s_t, \alpha_t) \times \left[r(s_t, \alpha_t) + \gamma \max_{\alpha} \widehat{Q}_t(s_{t+1}, \alpha) - \widehat{Q}_t(s_t, \alpha_t)\right]. \quad (1)$$

Having discussed the employed Q-learning algorithm, we now proceed to formally define the learning algorithms for the agents of MQ-Trader. The algorithms are presented in Figs. 6–9. In the algorithm descriptions, s_δ denotes the state on day δ and α_{s_δ} denotes an action taken at state s_δ. Furthermore, BP_δ and SP_δ, respectively, represent the BP and the SP determined on δ. For the notational brevity, we omit the index indicating the agent type throughout the algorithm descriptions

REPEAT
  Choose a random day δ
  $\alpha_{s_\delta} \leftarrow \Gamma(s_\delta)$
  IF (α_{s_δ} = BUY)
    Invoke the buy order agent and wait for the invocation from the sell order agent
    $r(s_\delta, \alpha_{s_\delta}) \leftarrow \dfrac{100 \times ((1 - TC) \times SP_{s_{SELL}} - BP_\delta)}{BP_\delta}$
  ELSE
    $r(s_\delta, \alpha_{s_\delta}) \leftarrow 0$
    Update $\vec{\theta}$ with
      $\vec{\theta} + \lambda[r(s_\delta, \alpha_{s_\delta}) - \widehat{Q}(s_\delta, \alpha_{s_\delta})]\nabla_{\vec{\theta}} \widehat{Q}(s_\delta, \alpha_{s_\delta})$
UNTIL (Early stopping criterion is satisfied)

Fig. 6. Algorithm for the buy signal agent.

IF ($MA_\delta^N \times \beta_{MAX}^N < P_{\delta+1}^L$)
  $r(s_\delta, \alpha_{s_\delta}) \leftarrow 0$
ELSE
  REPEAT
    $\beta \leftarrow \Gamma(s_\delta)$
    $\Delta \leftarrow MA_\delta^N \times \beta - P_{\delta+1}^L$
    IF ($\Delta < 0$)
      $r(s_\delta, \beta) \leftarrow 0$
      Update $\vec{\theta}$ with
        $\vec{\theta} + \lambda[r(s_\delta, \beta) - \widehat{Q}(s_\delta, \beta)]\nabla_{\vec{\theta}} \widehat{Q}(s_\delta, \beta)$
  UNTIL ($\Delta \geq 0$)
  $r(s_\delta, \beta) \leftarrow e^{-100 \times \Delta / P_{\delta+1}^L}$
  Invoke the sell signal agent
Update $\vec{\theta}$ with $\vec{\theta} + \lambda[r(s_\delta, \alpha_{s_\delta}) - \widehat{Q}(s_\delta, \alpha_{s_\delta})]\nabla_{\vec{\theta}} \widehat{Q}(s_\delta, \alpha_{s_\delta})$

Fig. 7. Algorithm for the buy order agent.

although each agent has its own definitions of state, action, reward, and Q function.

Fig. 6 shows the Q-learning algorithm for the buy signal agent. The buy signal agent first examines the state of a stock on a randomly selected day δ, which includes the TP matrix described in the previous section. It then takes an action according to a well-known ε-greedy policy function $\Gamma(\cdot)$ that is defined as follows [19]:

$$\Gamma(s_\delta) = \begin{cases} \arg\max_{\alpha \in \Omega(s_\delta)} \widehat{Q}(s_\delta, \alpha), & \text{with probability } 1 - \varepsilon \\ \text{random } \alpha \in \Omega(s_\delta), & \text{with probability } \varepsilon \end{cases}$$

where ε is an exploration factor, and $\Omega(s_\delta)$ represents the set of actions that can be taken at state s_δ.

If the agent decides to buy the stock, it immediately invokes the buy order agent and waits until the sell order agent invokes it. The reward is given later in terms of the resulting profit

$k = 1$
$\alpha_{s_{\delta+1}} \leftarrow \Gamma(s_{\delta+1})$
IF ($\alpha_{s_{\delta+1}}$ = HOLD)
  REPEAT
    $k \leftarrow k+1$
    $r(s_{\delta+k}, \alpha_{s_{\delta+k}}) \leftarrow q_{\delta+k}$
    $\vec{\theta} + \lambda \nabla_{\vec{\theta}} \widehat{Q}(s_{\delta+k}, \alpha_{s_{\delta+k}}) \times$
    Update $\vec{\theta}$ with $[r(s_{\delta+k}, \alpha_{s_{\delta+k}}) +$
    $\gamma \max_{\alpha \in \Omega(s_{\delta+k+1})} \widehat{Q}(s_{\delta+k+1}, \alpha) - \widehat{Q}(s_{\delta+k}, \alpha_{s_{\delta+k}})]$
    $\alpha_{s_{\delta+k}} \leftarrow \Gamma(s_{\delta+k})$
  UNTIL ($\alpha_{s_{\delta+k}}$ = SELL)
$r(s_{\delta+k}, \alpha_{s_{\delta+k}}) \leftarrow 0$
Update $\vec{\theta}$ with
$\vec{\theta} + \lambda[r(s_{\delta+k}, \alpha_{s_{\delta+k}}) - \widehat{Q}(s_{\delta+k}, \alpha_{s_{\delta+k}})] \nabla_{\vec{\theta}} \widehat{Q}(s_{\delta+k}, \alpha_{s_{\delta+k}})$
Invoke the sell order agent

Fig. 8. Algorithm for the sell signal agent.

IF ($MA^N_{\delta_{SELL}} \times \sigma^N_{MIN} > P^H_{\delta_{SELL}+1}$)
  $r(s_{\delta_{SELL}}, \alpha_{s_{\delta_{SELL}}}) \leftarrow 0$
ELSE
  REPEAT
    $\sigma \leftarrow \Gamma(s_{\delta_{SELL}})$
    $\Delta \leftarrow P^H_{\delta_{SELL}+1} - MA^N_{\delta_{SELL}} \times \sigma$
    IF ($\Delta < 0$)
      $r(s_{\delta_{SELL}}, \alpha_{s_{\delta_{SELL}}}) \leftarrow 0$
      Update $\vec{\theta}$ with
      $\vec{\theta} + \lambda[r(s_{\delta_{SELL}}, \sigma) - \widehat{Q}(s_{\delta_{SELL}}, \sigma)] \nabla_{\vec{\theta}} \widehat{Q}(s_{\delta_{SELL}}, \sigma)$
  UNTIL ($\Delta \geq 0$)
  $r(s_{\delta_{SELL}}, \alpha_{s_{\delta_{SELL}}}) \leftarrow e^{-100 \times \Delta / P^H_{\delta_{SELL}+1}}$
Update $\vec{\theta}$ with
$\vec{\theta} + \lambda[r(s_{\delta_{SELL}}, \alpha_{s_{\delta_{SELL}}}) - \widehat{Q}(s_{\delta_{SELL}}, \alpha_{s_{\delta_{SELL}}})] \nabla_{\vec{\theta}} \widehat{Q}(s_{\delta_{SELL}}, \alpha_{s_{\delta_{SELL}}})$
Invoke the buy signal agent

Fig. 9. Algorithm for the sell order agent.

rate by considering the following: 1) transaction cost (TC), which is defined in terms of a fixed rate charged for a stock price whenever a stock is purchased, and 2) the price slippage caused by the difference between the estimated and actual stock prices. Otherwise, the agent receives a zero reward to nullify the episode. For the update of $\vec{\theta}$, the term $\gamma \max_\alpha \widehat{Q}_t(s_{t+1}, \alpha)$ in (1) is set to 0, since no further Q value update for the current episode is necessary for the buy signal agent. Finally, an early stopping method [20] is adopted for the buy signal agent to terminate training when a validation error rate starts to grow up.

As described in Section II, the buy order agent has a state representation for the N-day MA, gradient, and normalized distance, as well as for several indicators for Japanese candlesticks. The action space for the buy order agent, i.e., $\Omega(s_\delta)$, is defined as a finite set of allowed BP ratio with respect to MA^N_δ, $\{\beta_1, \beta_2, \ldots, \beta_K\}$ such that $\beta_1 < \beta_2 < \cdots < \beta_K$ and $\beta_1 > 0$. We refer to β_K as β^N_{MAX} in what follows to represent the fact that it is dependent on N, which is the length of time window for the MA, and that it limits the maximum allowed BP. Given a BP ratio $\beta \in \Omega(s_\delta)$, the actual BP is determined by $\mathrm{BP}_\delta = \mathrm{MA}^N_\delta \times \beta$ on day δ for a trade on $\delta + 1$.

The learning algorithm for the buy order agent is presented in Fig. 7 in which β is used in place of α_{s_δ} whenever appropriate for clarity. It starts on day δ that is provided by the buy signal agent. If it turns out that a purchase cannot be made on day $\delta + 1$ with any BP ratio allowed in MQ-Trader, an episode ends after giving the minimum reward, which is 0. In case that a purchase is possible, the agent attempts to obtain a feasible BP for day $\delta + 1$ by repetitively trying different BP ratios by invoking the ε-greedy policy function. Since no state transition is made by the agent, the term $\gamma \max_\alpha \widehat{Q}_t(s_{t+1}, \alpha)$ in (1) is set to 0. The reward function for the buy order agent is defined in such a way that the computed reward is bounded by 0 and 1, and the reward becomes maximum when the BP determined is the same as the lowest possible BP of day $\delta + 1$.

The sell signal agent is informed about $\delta + 1$, which is the day when the stock is actually purchased, by the buy order agent. It then decides whether or not to sell the stock on $\delta + 1$ according to the ε-greedy function. Subsequently, if the decision is to sell the stock, the agent is provided with a zero reward as it will exit the market for the current episode. On the other hand, when the agent decides to hold the stock, the successive days are examined for selling the stock one by one by updating the Q value. The reward defined for this update is the ratio of closing price difference, i.e., $q_{\delta+k}$, which is defined in Section II, to inform whether the closing price has increased or not on the next day. We remark that unlike the buy order agent whose reward is bounded between 0 and 1, the reward for the sell signal agent may have a negative value. Furthermore, when the agent decides to sell, the term $\gamma \max_\alpha \widehat{Q}_t(s_{t+1}, \alpha)$ in (1) is set to 0, since no further state update is necessary for the episode. The algorithm for the sell signal agent is presented in Fig. 8.

Finally, δ_{SELL}, which is the day when the sell signal agent decided to sell the stock, is provided to the sell order agent that is responsible for determining an offer price. Similar to the case of the buy order agent, we define the action space for the sell order agent, i.e., $\Omega(s_{\delta_{\mathrm{SELL}}})$, to be a finite set of allowed SP ratio with respect to $\mathrm{MA}^N_{\delta_{\mathrm{SELL}}}$, $\{\sigma_1, \sigma_2, \ldots, \sigma_K\}$ such that $\sigma_1 < \sigma_2 < \cdots < \sigma_K$ and $\sigma_1 > 0$. We denote σ_1 as σ^N_{MIN}, since it determines the minimum allowed SP. Given an SP ratio $\sigma \in \Omega(s_\delta)$, the actual SP is computed in the same way as the case of the buy order agent.

As shown in Fig. 9, the agent first checks if it can sell the stock on day $\delta_{\mathrm{SELL}} + 1$ at the minimum allowed SP. If selling

of the stock even with the lowest possible price is not possible, the SP is set to $P^C_{\delta_{\rm SELL}+1}$, which is the closing price on day $\delta_{\rm SELL}+1$. The lowest reward, i.e., 0, is given in this case. Otherwise, the agent tries different prices until a feasible SP is obtained as in the case of the buy order agent. The reward function that considers the TC and price slippage for this case is defined similarly to that of the buy order agent and achieves the maximum value when the SP determined is equal to the highest possible price.

IV. Empirical Study

In this section, we first present the detail configuration of the MQ-Trader that is defined for empirical study and then discuss the predictability analysis results for the feedforward neural network employed for value function approximation. Finally, we present the results of an empirical study concerning the application of our multiagent approach to KOSPI 200, which is composed of 200 major stocks listed on the Korea stock exchange market, by comparing it with other alternative frameworks.

A. MQ-Trader Configuration

In addition to the major state definitions that were described in Section II, some additional state components are introduced for empirical study to further optimize the performance of MQ-Trader. Specifically, the signal agent is provided with ten additional binary technical indicators that include the relative strength index, MA convergence and divergence, price channel breakout, stochastics, on-balance volume, MA crossover, momentum oscillator, and commodity channel index. Detailed description of these indicators can be found in [21].

Furthermore, we consider the past 230 days for constructing the TP matrix and configure the sell signal agent to have the profit ratio representation scheme as shown in Table I where 8 bits are dedicated for the representation. Consequently, since the total number of bits required to represent the TP matrix is 324, it follows that the state of the buy signal agent consists of 334 bits and that the state of the sell signal agent consists of 342 bits.

As for the order agents, Table II shows the detailed state variables along with the number of bits configured for them for a trading day D. The value range of each state variable is divided into mutually exclusive intervals, and each interval is assigned with 1 bit to represent the fact that the current value belongs to the interval. Accordingly, both the buy order and the sell order agents require 88 bits.

We set $N=5$, which reflects the number of workdays in a week to train the order agents. In an attempt to minimize the required number of bits for representing the action space of the buy order agent while accommodating possible actions as many as possible, we analyzed the characteristics of KOSPI 200 by plotting the distribution of $P^L_{D+1}/{\rm MA}^5_D$, which is the ratio of the lowest stock price on $D+1$ to the five-day MA of stock prices on a trading day D. The result is presented in the left plot in Fig. 10, which suggests that the chance of producing an infeasible BP by the buy order agent with $\beta^5_{\rm MAX}=1.12$ is less than 2.5%. A similar conclusion can be drawn for the sell order agent from the right plot in Fig. 10, which shows the distribution of $P^H_{D+1}/{\rm MA}^5_D$. Based on this observation, the order agents are configured to have $\beta^5_{\rm MAX}=1.12$ and $\sigma^5_{\rm MIN}=0.88$, and the actual action space and its encoding used for the empirical study in this section are presented in Table III.

TABLE II
State Representation for the Order Agents

State variable	# of bits	State variable	# of bits
g^{20}_D	8	b_D	8
g^{10}_D	8	b_{D-1}	8
g^{5}_D	8	u_D	4
d^{20}_D	8	l_D	4
d^{10}_D	8	q_D	8
d^{5}_D	8	q_{D-1}	8

Finally, we remark that the algorithms presented in Figs. 7–9 may not terminate in some very rare cases. Therefore, we set the limit on the maximum number of iterations allowed during execution of the algorithms to prevent an infinite loop. When the loop is exited abruptly by this condition, the episode is discarded.

B. Predictability Analysis

The structure of a neural network for the Q value function approximation has a significant influence on the performance of MQ-Trader. In order to determine an appropriate structure, we considered several network structures by varying the number of hidden layers and the number of units for each layer.

The data set for the experimentation is drawn from KOSPI 200. The whole data set is divided into four subsets as follows: 1) the training set with 32 019 data points, which covers the time period from January 1999 to December 2000; 2) the validation set with 6102 data points from January 2001 to May 2001; 3) the first test set with 33 127 data points from June 2001 to August 2003; and finally 4) the second test set with 34 716 data points from September 2003 to November 2005.

Training of the neural networks was carried out by applying the Q-learning algorithms presented in Section III. Specifically, we considered the network configurations with at most two hidden layers, and each of them was trained ten times with different initial weights. The same neural network structure was used for all the agents of MQ-Trader. Prediction performance of the agents was investigated by examining the correlation between the estimated Q values and the actual discounted cumulative rewards as well as the accuracy, which is defined as the ratio of the number of successful trades to the number of recommendations made by MQ-Trader, all under $\gamma=0.9$, $\lambda=0.3$, and $\varepsilon=0.1$.

We remark that the correlation was calculated for all the stock items, whereas the accuracy was measured only for the stock items that were recommended by MQ-Trader. That is,

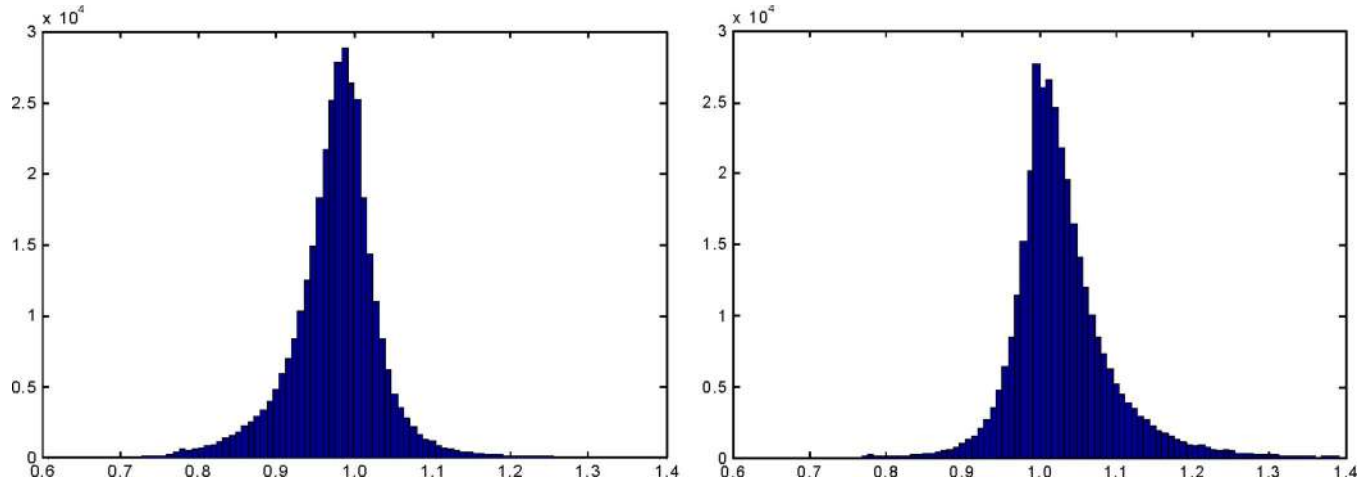

Fig. 10. Distribution of the ratio of the difference between the five-day MA and the lowest stock price to the five-day MA.

TABLE III
ACTION SPACE FOR THE ORDER AGENTS

Action	Encoding	Action	Encoding
0.93	00000001	1.12	10000000
0.93	00000010	1.07	01000000
0.95	00000100	1.03	00100000
0.99	00001000	1.01	00010000

the accuracy essentially represents how many stock trades were actually profitable out of those recommended for purchase. Table IV shows the prediction performance results for the considered neural network configurations. It suggests that the predictability for the recommended trades can be satisfactory even though the predictability for individual stocks is not. Based on these, we chose the network with 80 units in the first hidden layer and 20 units in the second hidden layer for implementing MQ-Trader.

The behavior of the trading agents with the selected network structure in MQ-Trader during the training process is depicted in Fig. 11, where the average number of trades made and the average profit rate incurred every 20 000 episodes are shown in the upper and lower graphs, respectively. In Fig. 11, the solid line represents the case of the validation set, whereas the dotted line represents the case of the training set. The vertical axes on the left- and right-hand sides in the upper plot in Fig. 11 represent the cases of the training set and the validation set, respectively.

It is worth mentioning that the number of trades made during the first 3 200 000 episodes is very small mainly due to the fact that MQ-Trader in this stage makes decisions only through the random exploration. In fact, the trading performance in terms of the profit rate in this stage is not satisfactory as shown in the bottom plot in Fig. 11. However, after this initial phase, the number of trades and the profit rate begin to increase in both data sets, indicating that MQ-Trader starts to trade stocks by use of the greedy policy. Finally, since it was observed that there was degradation of profit rate after 5 000 000 episodes, the training was stopped to prevent the overfitting.

C. Performance Evaluation Results

We implemented a simulation platform for trading systems to evaluate our approach. The platform consists of five independent subtraders for which initial assets are equally allocated. Each trader is allowed to hold only one stock item at a time. Motivation behind introducing multiple subtraders comes from the fact that the platform with a single trader may result in high variances of trading performance, making the performance comparison a sophisticated task. Indeed, in practice, there are very few investors who allocate their whole asset to a single stock.

At runtime, MQ-Trader implemented in the simulation platform constructs a set of recommended candidates out of 200 stock items based on the profit rate estimated by the trained neural network, and distributes the stocks with highest profitability randomly to the subtraders that do not hold a stock. Once a stock is to be purchased by a subtrader, BP is determined by comparing the estimated Q values for the set of possible actions. When the selected BP of the stock is unfortunately lower than the lowest stock price of the trading day, the stock is abandoned, and another randomly chosen profitable stock is provided to the subtrader. This process is repeated until there is no profitable stock left in the candidate set.

On the other hand, the decision of selling a stock proceeds as follows. On every trading day, two alternative actions, namely SELL or HOLD, are compared according to the Q values returned by the trained neural network. In case that the stock is to be sold, the SP is determined similarly to the case of the stock purchase. Whenever the SP determined is higher than the highest price of the trading day, the stock is sold at the closing price of the day.

In order to incorporate real-world trading constraints, we further introduced TCs, price slippage, and limitation on the stock purchase amount into the simulation model. First, three different rates for computing TCs based on the BP or SP, namely 0.5%, 1.0%, and 1.5%, were considered,[1] and the TC

[1]The actual rate for the transaction cost in KOSPI market is between 0.3% and 0.4%.

TABLE IV
PREDICTION PERFORMANCE OF THE CONSIDERED NEURAL NETWORK CONFIGURATIONS

One hidden layers			Two hidden layers		
Network Structure	Correlation	Accuracy	Network Structure	Correlation	Accuracy
80	0.274	0.577	40x20	0.291	0.617
100	0.283	0.593	60x20	0.304	0.639
120	0.291	0.611	80x20	0.336	0.663
160	0.285	0.598	100x20	0.315	0.651
200	0.278	0.584	120x40	0.296	0.625

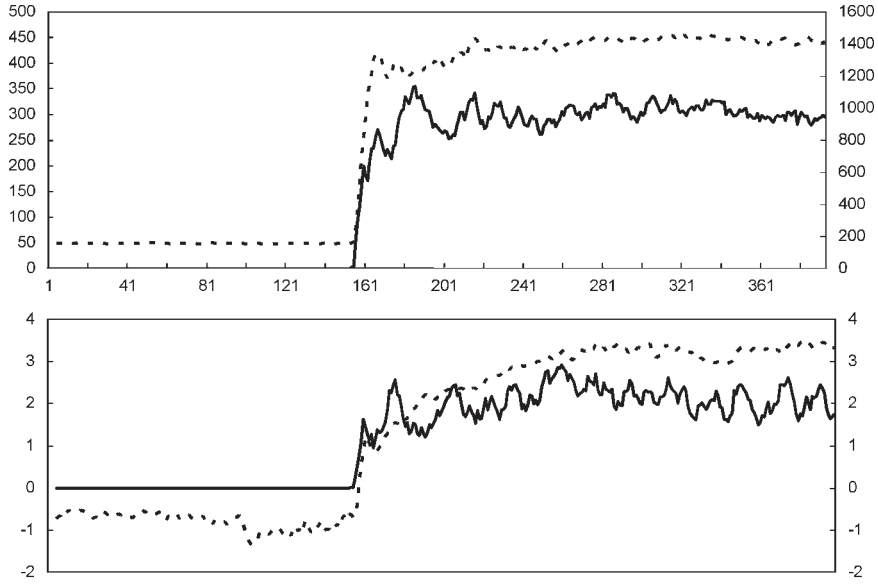

Fig. 11. Behavior of trading agents during the training process.

was charged whenever a stock is purchased or sold. Second, in order to account for the price slippages that may occur due to the difference between the estimated and actual stock prices, we introduced random perturbation of the actual stock prices by 0%, 0.5%, and 1%. Third, in an attempt to address the issue of minimizing the market influence caused by a stock trader, we limited the daily purchase amount of a single stock item by the trader to less than 1% of the daily trading volume of the stock in the market.

We now proceed to compare the performance of the proposed MQ-Trader with other trading systems with different architectures. The stock trading systems considered in this experimentation for performance comparisons are given as follows: (a) the Ideal 2Q (I2Q)-Trader that replaces the order agents of MQ-Trader with an ideal policy in which buy orders are traded at the lowest daily prices and sell orders are traded at the highest daily prices; (b) the MQ-Trader; (c) the 2Q-Trader in which only the signal agents are employed and the BP and SP are set to the closing price of a trading day; (d) the SMQ-Trader, which is the MQ-Trader without the TP matrix; (e) the 1Q-Trader where only the buy signal agent is employed and the selling signal is automatically generated after some predefined holding period; and finally (f) the SNN-Trader that has basically the same neural network structure as 1Q-Trader but employs a supervised learning algorithm.

TABLE V
STATISTICAL BENCHMARKING RESULTS FOR THE TRADERS

Trader	Average	Standard Dev.	Best Case
(b)	1072.18	53.91	1138.70
(c)	716.21	50.25	766.01
(d)	425.85	37.14	466.77
(e)	243.83	14.69	265.09
(f)	242.55	22.60	267.82

It should be noted that all traders except the SMQ-Trader implement the TP matrices for their state representations. We also remark that the 2Q-Trader and I2Q-Trader are the traders defined for the purpose of showing how the order agents play roles in enhancing the performance of the MQ-Trader by removing the order agents from the MQ-Trader and, respectively, replacing them with two extreme pricing policies. From the definitions of the 2Q-Trader and I2Q-Trader, it follows that the performance of the MQ-Trader should fall between those of the 2Q-Trader and I2Q-Trader.

In this experimentation, the neural network of each trading system (introduced as (a) to (f) previously) was trained with 20 different random initializations of the weights, and Table V summarizes the statistical benchmarking results in

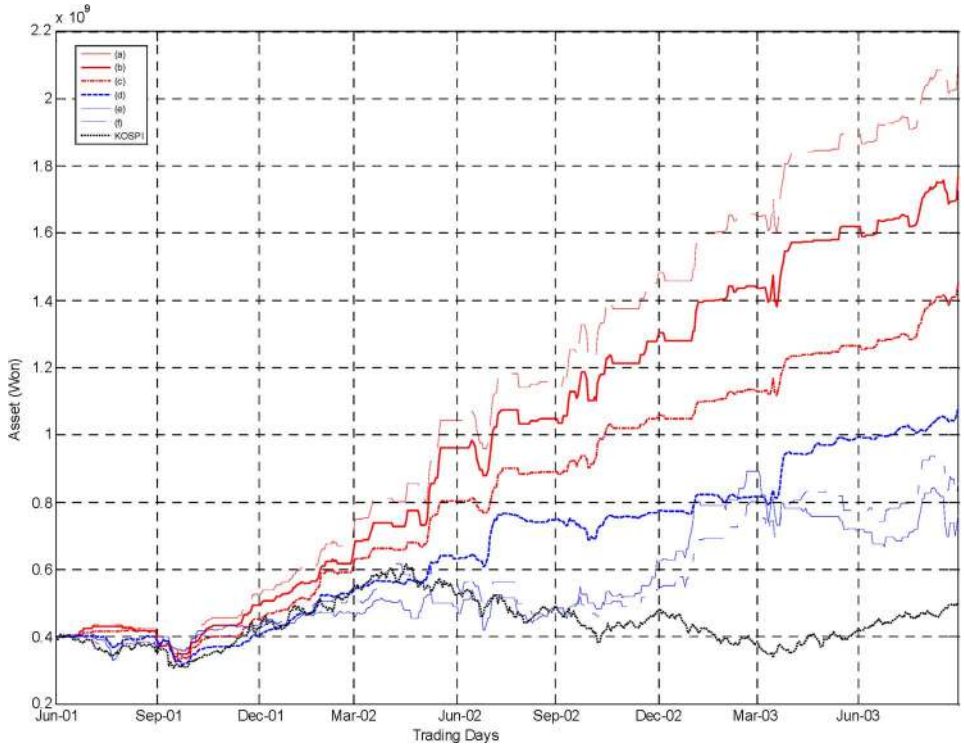

Fig. 12. Performance comparison result for the first test data set.

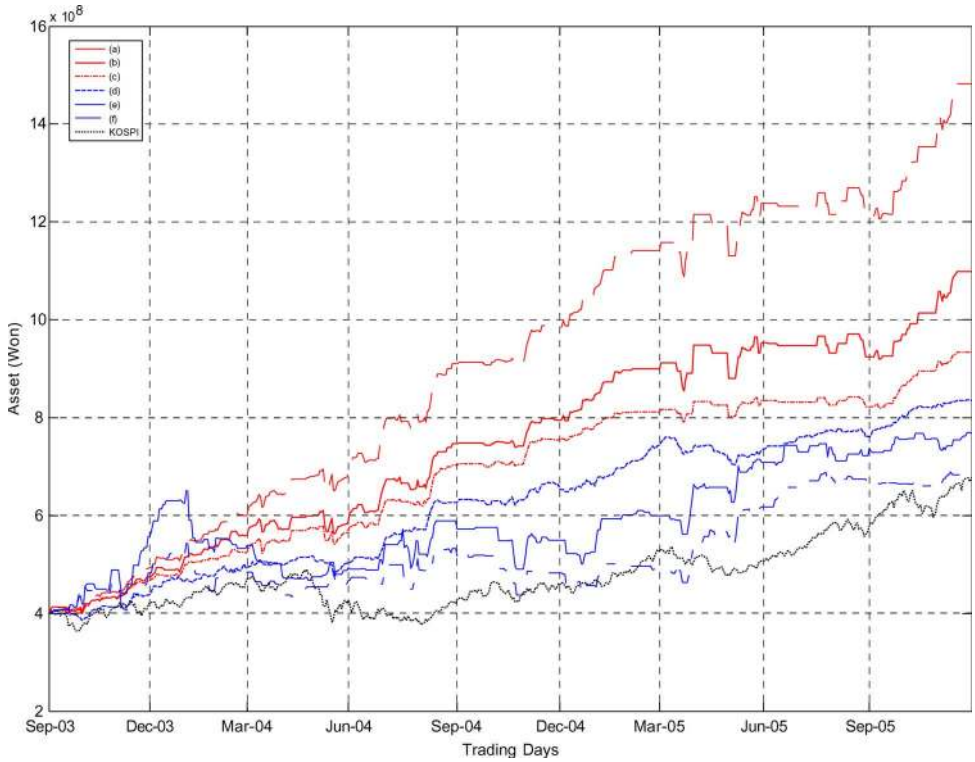

Fig. 13. Performance comparison result for the second test data set.

terms of the asset growth rates achieved by each trading system throughout the entire test period, from June 2001 to November 2005, when it was provided with 0.4 billion won initially (the basic unit of money in Korea). For example, the best case performance of MQ-Trader shows that its asset has grown to 4.55 $(= 0.4 \times 1138.7\ \%)$ billion won by the end of November 2005.

Only the best case results for the two aforementioned test sets in this section are separately given in Figs. 12 and 13, where the dotted line at the bottom indicates the KOSPI index

TABLE VI
ASSET GROWTH RATES OF MQ-TRADER FOR DIFFERENT TRANSACTION RATES AND PRICE SLIPPAGES

	Percentage of Price Slippage		
Transaction Cost Rate	0.0%	0.5%	1.0%
0.5%	1138.70	758.76	563.70
1.0%	616.81	391.84	254.99
1.5%	320.43	264.66	183.42

TABLE VII
TRADING FREQUENCIES OF MQ-TRADER FOR DIFFERENT TRANSACTION RATES AND PRICE SLIPPAGES

	Percentage of Price Slippage		
Transaction Cost Rate	0.0%	0.5%	1.0%
0.5%	893	882	843
1.0%	810	786	750
1.5%	717	682	636

that is similarly defined as the S&P 500 index of the U.S. stock market. KOSPI index is shown to compare the performances of the aforementioned trading systems to the baseline market performance during the test period, and it is equated to 0.4 billion at the beginning for visualization purposes. The series (a) through (f) in Fig. 12 show the accumulated assets for each trading system during the first test period (June 2001 to August 2003) when each system starts with an initial asset of 0.4 billion won. To make the performance comparison clear, the starting assets for the trading systems during the second test period (September 2003 to November 2005) are also equated to 0.4 billion in Fig. 13.

As can be seen from these results, the proposed MQ-Trader outperformed the other alternative trading frameworks (represented by the series (c) to (f) in Figs. 12 and 13) by achieving more than four times of asset growth for the first test period and more than 2.5 times for the second test period. The performance of MQ-Trader has always lied between those of the I2Q-Trader and the 2Q-Trader (respectively represented as the series (a) and (c) in Figs. 12 and 13) as expected. Accordingly, the performance difference between MQ-Trader and 2Q-Trader can be attributed to the contributions of the order agents. Furthermore, it can be deduced by comparing the series (b) and (d) that the proposed TP matrix can facilitate the performance improvement of a trading system.

It is interesting to note that MQ-Trader performs satisfactorily during the long period of bear market between April 2002 and April 2003. In addition, it endured quite well the short stock market shock during May 2004 with a relatively small loss. Note that, however, in the period of bull market (May 2005 to July 2005), the traders with multiple agents including MQ-Trader were not able to exploit the opportunity, while the other two single agent traders (indicated by the sharp rises of the series (e) and (f) during the period) were able to exploit the opportunity. Based on this observation, it appears that MQ-Trader can achieve a good performance particularly when the stock prices are sharply declining due to the market inefficiency incurred by some psychological reasons of investors.

The results of experimentation study to examine the effects of TCs and price slippages on the performance of MQ-Trader are presented in Tables VI and VII, where only the best case performances are shown among 20 multiple trials with different random initializations of neural networks. Three different rates for calculating TCs as well as three different probabilities of price slippages were considered, resulting to a total of nine configurations. Tables VI and VII, respectively, present the results of the asset growth rates and the trading frequencies achieved by MQ-Trader for different configurations during the entire period of June 2001 through November 2005. The initial asset given to MQ-Trader was 0.4 billion won.

From Table VI, it can be seen that the most profitable results (1138.7% asset growth) were obtained when both of the TC rate and the price slippage percentage were lowest, and that the profit decreases as the TC rate increases and the chance of price slippages becomes higher. Similar results were observed for the experimentation on the number of trades made during the same test period, as shown in Table VII. These together imply that MQ-Trader has learned the risks associated with stock trading, which were introduced through the TC and price slippage. When the TC is expensive, MQ-Trader buys and sells a stock carefully, leading to less frequent trades and smaller net profits. This is a natural consequence since a trade with small profit may end up with overall loss after paying the TCs for the stock purchase and disposition. In addition, with high chance of price slippage, it is advantageous for MQ-Trader to avoid aggressive trading.

Furthermore, Figs. 14 and 15 show how the profitability of MQ-Trader decreases as the risks increase throughout the entire test period. The TC rate and the percentage of price slippage used for each series in Figs. 14 and 15 are summarized in Table VIII. As expected, the profitability becomes highest when the risks represented by the TC and price slippage are lowest, and it becomes lowest when the risks are highest.

Finally, we found out that consideration of the current profit or loss by MQ-Trader did not necessarily lead to the disposition effect in contrast to the human investors who are subject to the disposition effect due to psychological reasons. The average number of days of holding a stock item by MQ-Trader for the profitable trades was 6.9, whereas it was 7.3 days for the unsuccessful trades. Therefore, this small difference of 0.4 day suggests that the MQ-Trader is not prone to the disposition effect.

V. CONCLUSION

There has long been a strong interest in applying machine learning techniques to financial problems. This paper has explored the issues of designing a multiagent system that aims to provide an effective decision support for daily stock trading problem. The proposed approach, which was named MQ-Trader, defines multiple Q-learning agents in order to effectively divide and conquer the stock trading problem in an integrated environment. We presented the learning framework along with the state representations for the cooperative agents

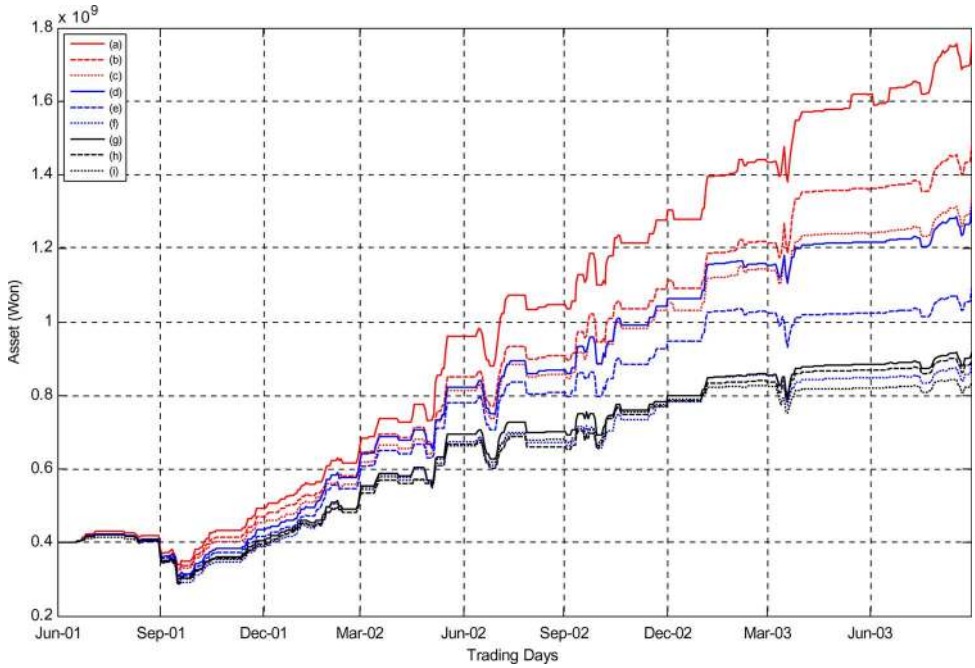

Fig. 14. Performance comparison result for different levels of risks during the first test period.

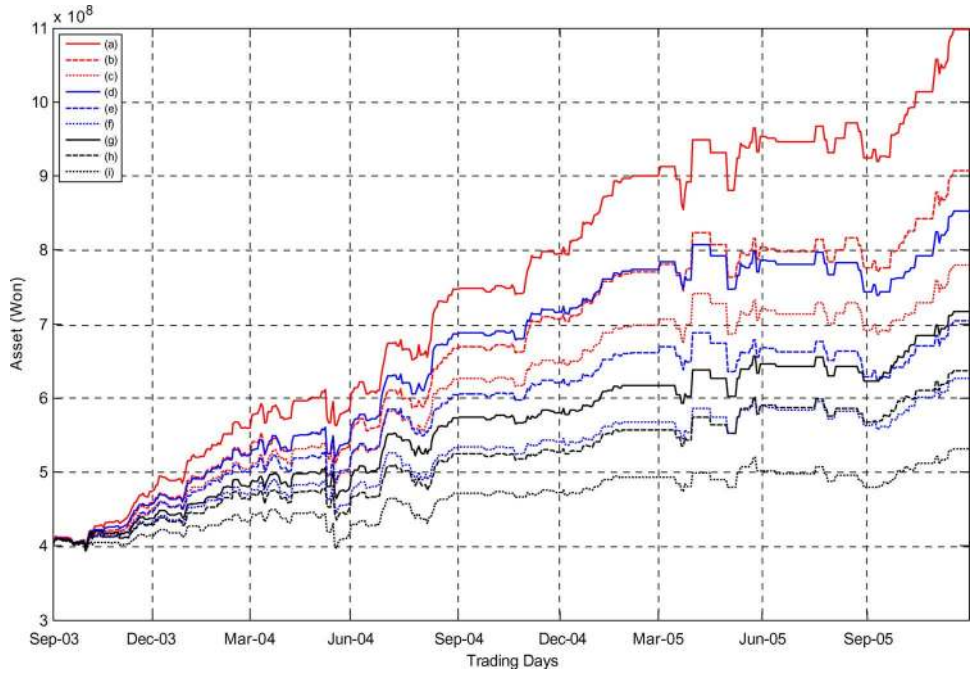

Fig. 15. Performance comparison result for different levels of risks during the second test period.

of MQ-Trader and described the detailed algorithms for training the agents.

Furthermore, in an attempt to address the complexity problem that arises when considering a large amount of data to compute long-term dependence among the stock prices, we had proposed a new state representation scheme, which was named TP matrix, that can succinctly represent the history of price changes.

Through an extensive empirical study using real financial data from a Korean stock market, we found that our approach produces better trading performances than the systems based on other alternative frameworks. Based on these observations, the profits that can be obtained from the proposed framework appear to be promising.

From the future research point of view, there are some clear extensions to be investigated. These include addressing the issues of distributing the asset to multiple portfolios and of adapting to the trend of a stock market. While the reinforcement learning is promising, introduction of these considerations will make the problem more complex. Therefore, one of the future

TABLE VIII
LEGEND FOR THE SERIES IN THE PLOTS OF FIGS. 14 AND 15

	Transaction cost rate	Percentage of price slippage
(a)	0.5%	0%
(b)	0.5%	0.5%
(c)	0.5%	1%
(d)	1%	0%
(e)	1%	0.5%
(f)	1%	1%
(g)	1.5%	0%
(h)	1.5%	0.5%
(i)	1.5%	1%

research problems will be to make the reinforcement learning formulation with these considerations tractable.

REFERENCES

[1] K. H. Lee and G. S. Jo, "Expert systems for predicting stock market timing using a candlestick chart," *Expert Syst. Appl.*, vol. 16, no. 4, pp. 357–364, May 1999.

[2] B. G. Malkiel, *A Random Walk Down Wall Street*. New York: Norton, 1996.

[3] K. Pantazopoulos, L. H. Tsoukalas, N. G. Bourbakis, M. J. Brun, and E. N. Houstis, "Financial prediction and trading strategies using neuro-fuzzy approaches," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 28, no. 4, pp. 520–531, Aug. 1998.

[4] T. Chenoweth, Z. Obradovic, and S. S. Lee, "Embedding technical analysis into neural network based trading systems," *Appl. Artif. Intell.*, vol. 10, no. 6, pp. 523–541, Dec. 1996.

[5] E. F. Fama and K. R. French, "Dividend yields and expected stock returns," *J. Financ. Econ.*, vol. 22, no. 1, pp. 3–26, Oct. 1988.

[6] M. Kearns and L. Ortiz, "The Penn–Lehman automated trading project," *IEEE Intell. Syst.*, vol. 18, no. 6, pp. 22–31, Nov./Dec. 2003.

[7] R. S. T. Lee, "iJADE stock advisor: An intelligent agent based stock prediction system using hybrid RBF recurrent network," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 34, no. 3, pp. 421–428, May 2004.

[8] R. Neuneier, "Enhancing *Q*-learning for optimal asset allocation," in *Proc. Adv. Neural Inf. Process. Syst.*, 1998, vol. 10, pp. 936–942.

[9] S. T. Chou, H.-J. Hsu, C.-C. Yang, and F. Lai, "A stock selection DSS combining AI and technical analysis," *Ann. Oper. Res.*, vol. 75, no. 1, pp. 335–353, Jan. 1997.

[10] Y. Luo, K. Liu, and D. N. Davis, "A multi-agent decision support system for stock trading," *IEEE Netw.*, vol. 16, no. 1, pp. 20–27, Jan./Feb. 2002.

[11] E. W. Saad, D. V. Prokhorov, and D. C. Wunsch, II, "Comparative study of stock trend prediction using time delay, recurrent and probabilistic neural networks," *IEEE Trans. Neural Netw.*, vol. 9, no. 6, pp. 1456–1470, Nov. 1998.

[12] T. M. Mitchell, *Machine Learning*. New York: McGraw-Hill, 1997.

[13] R. Neuneier and O. Mihatsch, "Risk sensitive reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 1999, vol. 1, pp. 1031–1037.

[14] X. Gao and L. Chan, "Algorithm for trading and portfolio management using *Q*-learning and Sharpe ratio maximization," in *Proc. ICONIP*, Taejon, Korea, 2000, pp. 832–837.

[15] J. Moody, Y. Wu, Y. Liao, and M. Saffell, "Performance functions and reinforcement learning for trading systems and portfolios," *J. Forecast.*, vol. 17, no. 5, pp. 441–470, 1998.

[16] J. Moody and M. Saffell, "Learning to trade via direct reinforcement," *IEEE Trans. Neural Netw.*, vol. 12, no. 4, pp. 875–889, Jul. 2001.

[17] S. Nison, *Japanese Candlestick Charting Techniques*. New York: New York Inst. Finance, 1991.

[18] J. E. Granville, *A Strategy of Daily Stock Market Timing for Maximum Profit*. Englewood Cliffs, NJ: Prentice-Hall, 1960.

[19] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

[20] R. Caruana, S. Lawrence, and L. Giles, "Overfitting in neural nets: Backpropagation conjugate gradient, and early stopping," in *Proc. Adv. Neural Inf. Process. Syst.*, 2001, vol. 13, pp. 402–408.

[21] M. A. H. Dempster and C. M. Jones, The Profitability of Intra-Day FX Trading Using Technical Indicators, 2000, Cambridge, U.K.: Centre Financ. Res., Univ. Cambridge. Working Paper.

Jae Won Lee received the Ph.D. degree from Seoul National University, Seoul, Korea, in 1998.

In 1999, he joined the faculty of the School of Computer Science and Engineering, Sungshin Women's University, Seoul, where he is currently an Assistant Professor. His current research interests include computational finance, machine learning, and natural language processing.

Jonghun Park (S'99–M'01) received the Ph.D. degree in industrial and systems engineering with a minor in computer science from the Georgia Institute of Technology, Atlanta, in 2000.

He is currently an Associate Professor with the Department of Industrial Engineering, Seoul National University (SNU), Seoul, Korea. Before joining SNU, he was an Assistant Professor with the School of Information Sciences and Technology, Pennsylvania State University, University Park, and with the Department of Industrial Engineering, Korea Advanced Institute of Science and Technology, Daejeon. His research interests include Internet services, entertainment computing, and mobile services.

Jangmin O received the Ph.D. degree from Seoul National University, Seoul, Korea, in 2006.

He is a Research Scientist with NHN Corporation, Seongnam, Korea. His research interests include intelligent agent systems, reinforcement learning, data mining, and probabilistic graphical models.

Jongwoo Lee received the Ph.D. degree from Seoul National University, Seoul, Korea, in 1996.

From 1996 to 1999, he was with Hyundai Electronics Industries, Co. He was an Assistant Professor with the Division of Information and Telecommunication Engineering, Hallym University, Chuncheon, Korea, from 1999 to 2002. Then, he moved to Kwangwoon University, Seoul, where he was an Assistant Professor of computer engineering from 2002 to 2003. Since 2004, he has been an Assistant Professor with the Department of Multimedia Science, Sookmyung Women's University, Seoul. His research interests include storage systems, computational finance, cluster computing, parallel and distributed systems, and system software.

Euyseok Hong received the Ph.D. degree from Seoul National University, Seoul, Korea, in 1999.

He was an Assistant Professor with the Department of Digital Media, Anyang University, Anyang, Korea, from 1999 to 2002. In 2002, he joined the faculty of the School of Computer Science and Engineering, Sungshin Women's University, Seoul, where he is currently an Associate Professor. His research interests include software engineering and web information systems.