Журнал обчислювальної та прикладної математики **2025, № 1** Journal of Numerical & Applied Mathematics

УДК 519.1, 519.7
MSC 68Q17, 68Q25

ROBUST TIME TO BUY AND SELL STOCK

N. M. Skybytskyi

Taras Shevchenko National University of Kyiv, Kyiv, Ukraine,
E-mail: n.skybytskyi@knu.ua, ORCID: 0009-0002-9507-3589

НАЙКРАЩИЙ ЧАС ДЛЯ РОБАСТНОЇ ТОРГІВЛІ АКЦІЯМИ

Н. М. Скибицький

Київський національний університет імені Тараса Шевченка, Київ, Україна,
E-mail: n.skybytskyi@knu.ua, ORCID: 0009-0002-9507-3589

Abstract. This paper focuses on the robust version of the classical algorithmic problem of finding the best time to buy and sell stock. We consider its variants with different transaction limits, as well as other modifications like transaction fee or cooldown. We reduce each classical problem to its robust version, thereby obtaining lower bounds on the time complexity of all potential solutions. We extensively test all developed methods on random and adversarial data to ensure correctness and evaluate performance.

We propose efficient methods for the robust counterparts of almost all problems. We also discuss suboptimal polynomial method based on dynamic programming techniques for the limited number of transactions. Developed methods are valuable in the practical applications of stock trading to obtain a minimum regret solution and evaluate the regret of an existing solution. We expect these applications of dynamic programming techniques to robust optimization problems to be relatively easy to extend and generalize for similar issues in combinatorial optimization.
Keywords: stock trading, robust optimization, dynamic programming, regret minimization, combinatorial optimization.

Анотація. У цій статті розглядається робастна версія класичної алгоритмічної задачі визначення найкращого моменту для купівлі та продажу акцій. Ми аналізуємо її варіанти з різними обмеженнями на кількість транзакцій, такими як одна чи дві операції, скінченна кількість транзакцій, а також необмежена кількість транзакцій. Додатково розглядаються такі модифікації як комісія за операцію або період очікування. Кожну класичну задачу ми зводимо до її робастного варіанту, отримуючи тим самим

обмеження знизу на часову складність усіх можливих алгоритмів розв'язування відповідних задач. Ми всебічно тестуємо розроблені методи на випадкових та навмисно ускладнених даних, щоб перевірити їхню коректність і оцінити ефективність.

У статті запропоновано ефективні точні методи з поліноміальним часом роботи для робастних аналогів майже всіх розглянутих задач. Також обговорюється субоптимальний поліноміальний метод для випадку з обмеженою кількістю транзакцій, заснований на техніці динамічного програмування. Розроблені методи знайдуть застосування у практиці торгівлі акціями з метою отримання рішення з мінімальним жалем, а також для оцінки жалю вже знайдених розв'язків. Ми переконані, що аналогічне застосування техніки динамічного програмування буде легко поширити та узагальнити на подібні задачі робастної комбінаторної оптимізації.

Ключові слова: торгівля акціями, робастна оптимізація, динамічне програмування, мінімізація жалю, комбінаторна оптимізація.

1. Introduction

Practical applications of operations research share the following stages in common:

modeling, prediction, and control.

Researchers may further partition each phase or add new steps relevant to a specific problem. However, the simplified three-stage scheme above is sufficient for this article.

We mainly deal with stock prices and trading strategies. To recap:

1. The first problem researchers face is to choose a model that acts as a simple yet comprehensive approximation of reality. Stock prices are frequently modeled using stochastic processes, such as Brownian motion. One example of this approach is [1]; alternative models like [2] also exist. A specific model used in a particular application is out of the scope of this paper.
2. Once a model is fixed, researchers face the problem of predicting or forecasting future stock prices. In this stage, classical statistical models like [3, 4] compete with newcomers based on neural networks, such as [5, 6]. A specific forecasting method is out of the scope of this paper.
3. Once the prices are predicted, researchers face the final problem of choosing a trading strategy. The trading strategy may influence future prices when operating on a large scale. Hence this stage is sometimes referred to as controlling a solution, a term borrowed from the optimal control theory. Examples of trading strategies include [7–10]. We aim to develop optimal robust trading strategies for the problems outlined in the abstract.

2. Classical problems

In this section, we give an overview of all classical problems. We highlight their similarities and differences. We also explain the practical meaning behind all input parameters.

All classical problems share the following input: $(p_i)_{i=1}^n$ — a sequence of positive integers, denoting prices of some stock on n consecutive days.

Definition 1. All classical problems share the following output: $I = (i_j)_{j=1}^{2k}$ — a strictly increasing sequence of indices from $[n]$. We refer to this sequence as a schedule.

Remark 1. Problem [15] imposes slightly stricter constraints on (i_j), namely $i_{2j+1} > i_{2j} + 1$ for all $j = 1, \ldots, k-1$. This corresponds to having a cooldown period of one day between consecutive sell and buy operations in this order. Note that there is no need to wait another day between consecutive buy and sell operations.

Definition 2. Problems [12, 15, 16] do not impose any limits on the number of transactions k. Problem [11] requires $k \leq 1$, [13] requires $k \leq 2$, and [14] requires $k \leq K$, where K is a new input parameter.

Remark 2. All problems with a limited number of transactions only introduce inequality constraints instead of equality constraints. This prevents optimal solutions from including loss-making transactions only to meet the equality constraint. In particular, enforcing even one operation does not make practical sense if sequence p is decreasing.

Definition 3. All problems except [16] define the profit of sequence $I = (i_1, \ldots, i_{2k})$ as the following alternating sum:

$$\begin{aligned} \text{profit}(I) &= -p_{i_1} + p_{i_2} - p_{i_3} + p_{i_4} - \ldots - p_{i_{2k-1}} + p_{i_{2k}} \\ &= (p_{i_2} - p_{i_1}) + (p_{i_4} - p_{i_3}) + \ldots + (p_{i_{2k}} - p_{i_{2k-1}}). \end{aligned} \tag{1}$$

Remark 3. Negative terms correspond to buying stock on days $i_1, i_3, \ldots, i_{2k-1}$ and paying money for it, whereas the positive terms correspond to selling stock on days $i_2, i_4, \ldots, i_{2k}$ and getting paid for it.

Definition 4. Problem [16] introduces another input parameter transaction fee f and extends the definition of profit as follows:

$$\begin{aligned} \text{profit}(I; f) &= (p_{i_2} - p_{i_1} - f) + (p_{i_4} - p_{i_3} - f) + \ldots + (p_{i_{2k}} - p_{i_{2k-1}} - f) \\ &= \text{profit}(i_1, \ldots, i_{2k}) - k \cdot f. \end{aligned} \tag{2}$$

Remark 4. This corresponds to having a fee f associated with every transaction. It forces an optimal solution to contain only high-profit transactions. In particular, it remotely resembles a technique of regularization.

The goal of every problem is to maximize profit. In other words, come up with a schedule having the maximum possible value of the corresponding definition of profit.

Remark 5. There may be multiple solutions with the same value of profit, in which case there is no need to enumerate all of them, and any single one is considered a valid return value.

3. Classical solutions

In this section, we briefly explore standard solutions to all of the classical problems. The observations we make — specifically those about dynamic programming techniques — will be used in the subsequent sections.

All problems can be solved using dynamic programming techniques. This is because all schedules consist of individual transactions. The transactions' value (profit or loss) does not depend on the rest of the schedule. Therefore, overlapping subproblems on subsegments $[m]$ of $[n]$ arise.

It is well known that dynamic programming is a go-to technique for many problems with optimal substructure. It consists of two primary parts: defining states – compact representations of subproblems, and transitions — relations between states' values. In the rest of this subsection, we will explore different kinds of parameters within a state.

As already hinted, states of all subproblems will contain a day m, denoting the prefix $[m]$ of days. The entire problem will have $m = n$.

First we will discuss the «hold, sold, and cold» states. Since problems [12, 15, 16] do not limit the number of transactions k. Therefore, we do not include information about the number of transactions in the state of a subproblem. Instead, the state depends on whether we hold stock at the end of the day m. The problem [15] is further concerned with us potentially being in a cooldown period if we just sold the stock on the previous day $m - 1$. The transitions in these problems are as follows, where mp stands for max profit.

– an unlimited number of transactions from [12]:

$$\begin{aligned} mp(m, \text{hold}) &= \max(mp(m-1, \text{hold}), mp(m-1, \text{sold}) - p_m); \\ mp(m, \text{sold}) &= \max(mp(m-1, \text{sold}), mp(m-1, \text{hold}) + p_m). \end{aligned} \tag{3}$$

– a transaction fee from [16]:

$$\begin{aligned} mp(m, \text{hold}) &= \max(mp(m-1, \text{hold}), mp(m-1, \text{sold}) - p_m); \\ mp(m, \text{sold}) &= \max(mp(m-1, \text{sold}), mp(m-1, \text{hold}) + p_m - f). \end{aligned} \tag{4}$$

Note that the following formulas are equivalent since the final answer is $mp(n, \text{sold})$:

$$\begin{aligned} mp(m, \text{hold}) &= \max(mp(m-1, \text{hold}), mp(m-1, \text{sold}) - p_m - f); \\ mp(m, \text{sold}) &= \max(mp(m-1, \text{sold}), mp(m-1, \text{hold}) + p_m). \end{aligned}$$

Feel free to use whichever you like most.

– a cooldown period from [15]:

$$\begin{aligned} mp(m, \text{hold}) &= \max(mp(m-1, \text{hold}), mp(m-1, \text{sold}) - p_m); \\ mp(m, \text{sold}) &= \max(mp(m-1, \text{sold}), mp(m-1, \text{cold})); \\ mp(m, \text{cold}) &= mp(m-1, \text{hold}) + p_m. \end{aligned} \tag{5}$$

Now we discuss the «number of transactions» states. Problems [11, 13, 14] all involve a limited number of transactions. Therefore, we include information about the number of transactions in the state of a subproblem. One may keep the explicit hold/sold parameter or replace it with implicit index j within the schedule. This is possible because the parity of j uniquely determines the operation type (buy or sell). The transitions in all of these problems are as follows:

$$mp(m, j) = \max(mp(m-1, j), mp(m-1, j-1) + (-1)^j p_m). \tag{6}$$

Remark 6. Note that the primary difference is not in the transitions. Instead, the location of the final answer $\max_{j=0..K} mp(n, 2j)$ differs. $K = 1$ in [11], and $K = 2$ in [13].

4. Robust problems

In this section, we introduce robust versions of all problems.

We define maximum worst-case profit and minimum regret solutions. We come up with observations that allow generalizing the solutions from the previous section to find the maximum worst-case profit. Finally, we show that a minimum regret solution can differ from a maximum worst-case profit solution.

Definition 5. All robust problems replace sequence $(p_i)_{i=1}^n$ with two sequences $(l_i)_{i=1}^n$ and $(r_i)_{i=1}^n$ of positive integers, denoting price ranges of some stock on n consecutive days. Namely, $l_i \le p_i \le r_i$ holds for every $i = 1, \ldots, n$.

Definition 6. Naturally, a definition of profit needs to be adjusted too. However, this adjustment is semantic rather than symbolic:

$$\begin{aligned} \text{profit}(I; p) &= -p_{i_1} + p_{i_2} - p_{i_3} + p_{i_4} - \ldots - p_{i_{2k-1}} + p_{i_{2k}} \\ &= (p_{i_2} - p_{i_1}) + (p_{i_4} - p_{i_3}) + \ldots + (p_{i_{2k}} - p_{i_{2k-1}}). \end{aligned} \tag{7}$$

In other words, the difference is that the value of profit now depends on a particular realization of prices p that satisfies given (l, r) constraints. Note that we fix the entire schedule before we learn any of the prices.

Remark 7. Note that the classical version can be reduced to the robust one by setting $l_i = p_i = r_i$. Therefore, a solution of the robust version cannot be asymptotically faster than the optimal solution of its classical counterpart.

Definition 7. By minimizing this definition over realization p we can define worst-case profit of the schedule $I = (i_j)_{j=1}^{2k}$:

$$\begin{aligned} \text{worst-case profit}(I; p) &= \min_p \text{profit}(i_1, \ldots, i_{2k}; p) \\ &= (l_{i_2} - r_{i_1}) + (l_{i_4} - r_{i_3}) + \ldots + (l_{i_{2k}} - r_{i_{2k-1}}). \end{aligned} \tag{8}$$

The last transformation is possible because the constraints are independent of each other. The objective function is linear in each of its arguments p_m. Hence it attains its minimum value in one of the boundary points. The worst-case scenario when buying stock is r_m and the worst-case scenario when selling stock is l_m.

Dynamic programming solutions presented in the section on classical solutions can find the maximum worst-case profit solution of a robust problem. To adapt these solutions, one has to replace each occurrence of p_m in (3), (4), (5), (6) with either l_m or r_m, as explained above.

Definition 8. To define regret of a solution we first need to define a worst-case loss of the solution I with respect to another solution J:

$$\text{worst-case loss}(I; J) = \max_p(\text{profit}(J; p) - \text{profit}(I; p)). \tag{9}$$

Definition 9. A regret of solution I is the maximum worst-case loss over all possible solutions J, where $\arg\max_J$ is often called an optimal solution:

$$\text{regret}(I) = \max_J \text{worst-case loss}(I; J) \tag{10}$$

A minimum regret solution is the one having the smallest possible regret. It is important to understand that minimizing regret differs from maximizing worst-case profit.

Consider two days with price ranges of $(1, 3)$ and $(2, 4)$ respectively. There are two solutions: do nothing with $I = (\ )$, or buy and sell with $I = (1, 2)$. Their worst-case profits are 0 and $\min_p(p_2 - p_1) = 2 - 3 = -1$ respectively. Their regrets are $\max_p(0 - 0, (p_2 - p_1) - 0) = \max_p(0, p_2 - p_1) = 4 - 1 = 3$ and $\max_p(0 - (p_2 - p_1), (p_2 - p_1) - (p_2 - p_1)) = \max_p(p_1 - p_2, 0) = 3 - 2 = 1$ respectively. See Table for a side-by-side comparison.

Table 1. Comparison of worst-case profits and regrets of two schedules on price ranges $(1, 3)$ and $(2, 4)$. The optimal solution by each criterion (column) is underlined.

schedule	worst-case profit	regret
$I = (\ )$	$\underline{0}$	3
$I = (1, 2)$	-1	$\underline{1}$

Hence the maximum worst-case profit solution is to do nothing, and the minimum regret solution is to buy and sell. We interpret this result in favor of regret minimization in a bull market. The following example demonstrates that the solutions may coincide, particularly in the case of a bear market.

Consider two days with price ranges of $(2, 4)$ and $(1, 3)$ respectively. There are two solutions: do nothing with $I = (\ )$, or buy and sell with $I = (1, 2)$. Their worst-case profits are 0 and $\min_p(p_2 - p_1) = 1 - 4 = -3$ respectively. Their regrets are $\max_p(0 - 0, (p_2 - p_1) - 0) = \max_p(0, p_2 - p_1) = 3 - 2 = 1$ and $\max_p(0 - (p_2 - p_1), (p_2 - p_1) - (p_2 - p_1)) = \max_p(p_1 - p_2, 0) = 4 - 1 = 3$ respectively. Hence both maximum worst-case profit and minimum regret solutions are to do nothing. See Table for a side-by-side comparison.

The first example can be further modified to show that the regrets of two solutions can differ arbitrarily. Specifically, in a case of two days with price ranges $(1, x + 1)$ and $(x, 2x)$, the maximum worst-case profit solution has a regret of $2x - 1$, whereas the minimum regret is 1. In particular, there does not exist a constant C s.t.:

$$\text{regret}(\text{max worst-case profit solution}) \leq C \cdot \text{regret}(\text{min regret solution}).$$

TABLE 2. Comparison of worst-case profits and regrets of two schedules on price ranges $(2, 4)$ and $(1, 3)$. The optimal solution by each criterion (column) is underlined.

schedule	worst-case profit	regret
$I = (\ )$	$\underline{0}$	$\underline{1}$
$I = (1, 2)$	-3	3

A new efficient method is therefore needed to find or even approximate (within a constant factor) the minimum regret solution.

5. Robust solutions

In this section, we develop efficient algorithms to find minimum regret solutions for all discussed problems.

We start by analyzing the regret definition. Regrets (10) and (9) together yield the following expression:

$$\text{regret}(I) = \max_J \text{worst-case loss}(I; J) = \max_{J,p}(\text{profit}(I; p) - \text{profit}(J; p)).$$

We are hence interested in

$$\min_I \max_{J,p}(\text{profit}(J; p) - \text{profit}(I; p)). \tag{11}$$

Consider an arbitrary day i. Schedules I and J can buy stock on that day, sell the stock, or do nothing. According to the product rule, there are 9 cases in total:

1. both schedules buy stock on the day i. The regret is independent of price p_i;
2. schedule I buys stock on day i and schedule J sells stock on day i. The $\max_p$ in the regret formula implies that we should set $p_i = r_i$. This means that schedule I will pay more and schedule J will earn more;
3. schedule I buys stock on day i and schedule J does nothing. We set $p_i = r_i$;
4. schedule I sells stock on day i and schedule J buys stock on day i. We set $p_i = l_i$ to make schedule I earn less and schedule J pay less;
5. both schedules sell stock on the day i. The regret is independent of price p_i;
6. schedule I sells stock on day i and schedule J does nothing. We set $p_i = l_i$;
7. schedule I does nothing on day i and schedule J buys stock on day i. We set $p_i = l_i$;
8. schedule I does nothing on day i and schedule J sells stock on day i. We set $p_i = r_i$;
9. both schedules do nothing on the day i. The regret is independent of price p_i.

The analysis is summarized in Table .

Hence, given a fixed pair of schedules I and J, we know the prices p resulting in the maximum regret. In particular, we can compute it in linear time.

TABLE 3. Each cell contains the value of p_i resulting in the maximum regret for a given pair of operations in schedules I and J. A dash indicates that the regret does not depend on the value of p_i.

$I \setminus J$	buy	pass	sell
buy	–	r_i	r_i
pass	l_i	–	r_i
sell	l_i	l_i	–

It remains to analyze the behavior of an expression of the form $\min_I \max_J$. This is reminiscent of a two-player game. As before, the min-max part implies the presence of some optimal substructure, and there are many overlapping subproblems. These are the primary characteristics of a dynamic programming problem. The difference is that this time we need to include the current state of both «players».

First we will discuss the «hold, sold, and cold» states. This subsubsection discusses the applications of a dynamic programming approach to problems [12, 15, 16].

As before, the state depends on whether schedule I holds stock at the end of the day m. Now the state also depends on whether schedule J holds stock at the end of the day m. The problem [15] is further concerned with I and J potentially being in a cooldown period if they just sold the stock on the previous day $m-1$. The transitions in these problems are as follows, where mr stands for «minimum regret».

– an unlimited number of transactions from [12]:

$$\begin{aligned}
mr(m, \text{hold}, \text{hold}) = \min(&\max(mr(m-1, \text{hold}, \text{hold}),\\
&mr(m-1, \text{hold}, \text{sold}) - l_m),\\
&\max(mr(m-1, \text{sold}, \text{hold}) + r_m,\\
&mr(m-1, \text{sold}, \text{sold})));\\
mr(m, \text{hold}, \text{sold}) = \min(&\max(mr(m-1, \text{hold}, \text{hold}) + r_m,\\
&mr(m-1, \text{hold}, \text{sold})),\\
&\max(mr(m-1, \text{sold}, \text{hold}) + 2r_m,\\
&mr(m-1, \text{sold}, \text{sold}) + r_m));\\
mr(m, \text{sold}, \text{hold}) = \min(&\max(mr(m-1, \text{hold}, \text{hold}) - l_m,\\
&mr(m-1, \text{hold}, \text{sold}) - 2l_m),\\
&\max(mr(m-1, \text{sold}, \text{hold}),\\
&mr(m-1, \text{sold}, \text{sold}) - l_m));\\
mr(m, \text{sold}, \text{sold}) = \min(&\max(mr(m-1, \text{hold}, \text{hold}),\\
&mr(m-1, \text{hold}, \text{sold}) - l_m),\\
&\max(mr(m-1, \text{sold}, \text{hold}) + r_m,\\
&mr(m-1, \text{sold}, \text{sold}))).
\end{aligned} \tag{12}$$

As you can see, the equations are a bit verbose. They can be presented as a matrix instead:

$$\begin{pmatrix} 0 & -l_m & r_m & 0 \\ r_m & 0 & 2r_m & r_m \\ -l_m & -2l_m & 0 & -l_m \\ 0 & -l_m & r_m & 0 \end{pmatrix} \tag{13}$$

– a transaction fee from [16]:

$$\begin{pmatrix} 0 & -l_m - f & r_m + f & 0 \\ r_m & 0 & 2r_m + f & r_m + f \\ -l_m & -2l_m - f & 0 & -l_m - f \\ 0 & -l_m & r_m & 0 \end{pmatrix} \tag{14}$$

– a cooldown period from [15] has a 9×9 matrix that can be constructed similarly.

Now we discuss the «number of transactions» states. Problems [11, 13, 14] all involve a limited number of transactions. Therefore, we include information about the number of transactions in the state of a subproblem.

One may keep the explicit hold/sold parameter or replace it with implicit indices i and j within the corresponding schedule. This is possible because the parities of i and j uniquely determine the operation types (buy or sell). The transitions in all of these problems are as follows:

$$\begin{aligned} mr(m,i,j) = \min(\max(&mr(m-1,i,j), \\ &mr(m-1,i,j-1) + f_{0,(-1)^j}(l_m, r_m)), \\ \max(&mr(m-1,i-1,j) + f_{(-1)^i,0}(l_m, r_m), \\ &mr(m-1,i-1,j-1) + f_{(-1)^i,(-1)^j}(l_m, r_m))). \end{aligned} \tag{15}$$

Here f is a matrix of functions (with somewhat unconventional indices -1, 0, and 1) that calculate the regret change based on . The specific values are out of the scope of this article. Instead, the most important result is that some dynamic programming solution is possible.

Remark 8. Note that the primary difference is not in the transitions. Instead, the location of the final answer $\max_{i=0..K,j=0..K} mr(n, 2i, 2j)$ differs. $K = 1$ in [11], and $K = 2$ in [13].

Despite encountering 4×4 and 9×9 matrices, we do not include their sizes in the time complexity analysis because the sizes are constant with respect to the number of days. Hence, the obtained solutions for problems [12, 15, 16] have time complexity of $O(n)$. The time complexity of the solution developed for [14] is $O(nk^2)$ because the number of states grew by a factor of $O(k)$. In particular, time complexities of [11, 13] are still $O(n)$, since $K = O(1)$.

Overall, the time complexities of all solutions except [14] remained the same as those of their non-robust counterparts.

It is an open question whether more optimal solutions for the robust version of this problem exist.

We expect more efficient algorithms for this problem to be approximation schemes with a good constant factor.

A further study of the prices maximizing the regret may yield observations useful for certain approximation heuristics.

6. Conclusion

In this paper we introduced a robust version of the classical algorithmic problem of finding the best time to buy and sell stock [12]. We considered its variants with different transaction limits [11,13,14], as well as other modifications like [16] and [15]. We reduced each classical problem to its robust version, thereby obtaining lower bounds on the complexity of all potential solutions.

We developed efficient methods for the robust counterparts of [11–13,15,16].

We also discussed suboptimal polynomial methods based on dynamic programming techniques for [14].

Finally, we outlined the open questions for future research.

The author declares that there is no conflict of interest concerning the publication of this article.

References

1. Bianchi S., Pantanella A., Pianese A. Modeling stock prices by multifractional Brownian motion: an improved estimation of the pointwise regularity. *Quantitative Finance*. 2013. Vol. 13, No. 8. P. 1317–1330.
2. Chernov M., Gallant A.R., Ghysels E., Tauchen G. Alternative models for stock price dynamics. *Journal of Econometrics*. 2003. Vol. 116, No. 1–2. P. 225–257.
3. Mondal P., Shit L., Goswami S. Study of effectiveness of time series modeling (ARIMA) in forecasting stock prices. *Int. J. Comput. Sci., Eng. and Applications*. 2014. Vol. 4, No. 2. P. 13.
4. Jarrett J.E., Kyper E. ARIMA modeling with intervention to forecast and analyze Chinese stock prices. *Int. J. Eng. Bus. Manag.*. 2011. Vol. 3, No. 3. P. 53–58.
5. Kim T., Kim H.Y. Forecasting stock prices with a feature fusion LSTM-CNN model using different representations of the same data. *PLoS ONE*. 2019. Vol. 14, No. 2. e0212320.
6. Tsantekidis A., Passalis N., Tefas A., Kanniainen J., Gabbouj M., Iosifidis A. Forecasting stock prices from the limit order book using convolutional neural networks. In: *Proc. IEEE 19th Conf. on Business Informatics (CBI)*. 2017. Vol. 1. P. 7–12.
7. Wu X., Chen H., Wang J., Troiano L., Loia V., Fujita H. Adaptive stock trading strategies with deep reinforcement learning methods. *Information Sciences*. 2020. Vol. 538. P. 142–158.
8. Nuij W., Milea V., Hogenboom F., Frasincar F., Kaymak U. An automated framework for incorporating news into stock trading strategies. *IEEE Trans. Knowl. Data Eng.*. 2013. Vol. 26, No. 4. P. 823–835.
9. Fifield S.G.M., Power D.M., Sinclair C. D. An analysis of trading strategies in eleven European stock markets. *Eur. J. Finance*. 2005. Vol. 11, No. 6. P. 531–548.
10. Vijh A.M. S&P 500 trading strategies and stock betas. *Rev. Financ. Stud.*. 1994. Vol. 7, No. 1. P. 215–251.

11. LeetCode. Best Time to Buy and Sell Stock with One Transaction. 2014. https://leetcode.com/problems/best-time-to-buy-and-sell-stock/. Accessed: 2023-03-21.
12. LeetCode. Best Time to Buy and Sell Stock with Unlimited Transactions. 2014. https://leetcode.com/problems/best-time-to-buy-and-sell-stock-ii/. Accessed: 2023-03-21.
13. LeetCode. Best Time to Buy and Sell Stock with Two Transactions. 2014. https://leetcode.com/problems/best-time-to-buy-and-sell-stock-iii/. Accessed: 2023-03-21.
14. LeetCode. Best Time to Buy and Sell Stock with Limited Transactions. 2015. https://leetcode.com/problems/best-time-to-buy-and-sell-stock-iv/. Accessed: 2023-03-21.
15. LeetCode. Best Time to Buy and Sell Stock with Cooldown. 2016. https://leetcode.com/problems/best-time-to-buy-and-sell-stock-with-cooldown/. Accessed: 2023-03-21.
16. LeetCode. Best Time to Buy and Sell Stock with Transaction Fee. 2017. https://leetcode.com/problems/best-time-to-buy-and-sell-stock-with-transaction-fee/. Accessed: 2023-03-21.

Received: 20.01.2025 / Accepted: 24.02.2025