ISSN 2786-9482

Ukrainian Journal of Information Systems and Data Science

since 2023

ISSN 2786-9482

Ukrainian Journal of Information Systems and Data Science

2024 Volume 2 Issue 1

УДК 004
Міжнародний науковий журнал ***Ukrainian Journal of Information Systems and Data Science*** виходить двічі на рік. Засновник та видавець – Донецький національний університету імені Василя Стуса. Журнал засновано в травні 2023 р. як електронне наукове видання. Журнал публікує статті українською або англійською мовами.
ISSN 2786-9482
Тематика журналу:

- алгоритми та структури даних;
- інформаційні ресурси;
- моделі прийняття рішень;
- мета-евристична оптимізація;
- моделювання та оптимізація мережевих систем;
- безпечність та надійність складних систем;
- людино-комп'ютерна взаємодія;
- інформаційні системи на основі інтернету речей;
- машинне навчання;
- інженерія знань;
- технології обробки природної мови.

Ukrainian Journal of Information Systems and Data Science: міжнародний науковий журнал / гол. ред. С. Штовба. Вінниця: Донецький національний університету імені Василя Стуса, 2024. № 1. 77 с.

Ukrainian Journal of Information Systems and Data Science, №1 за 2024 р. видається за рішення Вченої ради Донецького національного університету імені Василя Стуса, протокол №1 від 29 серпня 2024 р.
Літературний редактор номер – **Ольга Солдатова**.
Номер зверстано редакційною колегією. Підписано до друку 04 жовтня 2024 р.

Адреса редакції:
Донецький національний університету
імені Василя Стуса, вул. 600-річчя, 21,
Вінниця, 21021, Україна
Тел.: +380-68-256-10-56.
https://jujisds.donnu.edu.ua

UDC 004
Ukrainian Journal of Information Systems and Data Science is an electronic international scientific journal with two issues per year. Vasyl' Stus Donetsk National University is the founder and publisher of the journal. The journal was founded in May 2023. Articles are published in Ukrainian and English.
ISSN 2786-9482
Topics of the journal include:

- Algorithms and Data Structures;
- Information Resources;
- Decision-Making Models;
- Metaheuristic Optimization;
- Modeling and Optimization of Networked Systems;
- Safety and Reliability of Complex Systems;
- Human-Computer Interaction;
- Internet of Things Systems;
- Machine Learning;
- Knowledge Engineering;
- Natural Language Processing Technologies.

Ukrainian Journal of Information Systems and Data Science: international scientific journal / editor-in-chief Serhiy Shtovba. Vinnytsia: Vasyl' Stus Donetsk National University, 2024, vol. 2, issue 1. 77 p.

Ukrainian Journal of Information Systems and Data Science, volume 2, issue 1 is published by the decision of the Academic Council of Vasyl' Stus Donetsk National University, minutes No 1 of August 29, 2024.

Language editor of the issue is **Olga Soldatova**.
The issue was designed by the editorial board. The issue was signed for publication on October 04, 2024.

Contacts:
Vasyl' Stus Donetsk National University,
600-richchia str., 21, Vinnytsia, 21021,
Ukraine
Phone: +380-68-256-10-56
https://jujisds.donnu.edu.ua

ЗМІСТ

CONTENTS

UDC 004.056.55+004.421.5

Knight Pseudorandom Walk Manifold for Scrambling Multidimensional Data

Vadim Romanuke
Professor, DrSc
ORCID: 0000-0001-9638-9572
v.romanyuk@vtei.edu.ua
romanukevadimv@gmail.com

Vinnytsia Institute of Trade and Economics
of State University of Trade and Economics

Keywords:
data scrambling;
knight pseudorandom walk;
break-in probability;
similarity rate.

The knight open tour problem is to build a sequence of knight moves covering a chessboard completely, without repetitions, where the starting position and ending position are always different. A solution of the knight open tour problem is similar to a sequence of pseudorandom numbers used to map data into non-readable yet usable information. The knight open tour problem has a manifold of solutions for a starting position of the knight depending on the chessboard size. Solutions of the knight open tour problem, which appear like a random series of knight positions, i. e. a pseudorandom walk, are used to further improve balance of the scrambling simplicity and productivity, where the main indicators are the break-in probability and similarity index. The break-in probability is dramatically decreased by taking into account a knight pseudorandom walk manifold for a starting position on a given chessboard. A pessimistic estimation of the break-in probability for an 8×8 chessboard is less than 10^{-16}. A similarly expected estimation for an $8\times 8\times 8$ chessboard is less than 10^{-26}. A distinct knight pseudorandom walk (out of a manifold of pseudorandom walks) is built online by a given seed integer for a pseudorandom number generator. The scrambled data vector is built online as well in linear runtime complexity. Meanwhile, the similarity index is acceptable, rapidly dropping as the chessboard size is increased (for bigger multidimensional data arrays). A knight pseudorandom walk is determined by the chessboard size, the starting position, the way to vectorize the knight pseudorandom walk, and the pseudorandom number generator seed allowing to specifically move the knight onward through situations with multiple possible moves of the knight. The knight-open-tour scrambler has 10^{16} to 10^{21} times lower break-in probability compared to an ordinary pseudorandom number generator, depending on the chessboard size and the starting position of the knight.

DOI: 10.31558/2786-9482.2024.1.1

Introduction

The main technical purpose of data privacy is to achieve sufficiently low likelihood of deliberately intruding, spoofing, hacking, delaying, distorting, etc. [1, 2]. In particular, data is scrambled by altering or shuffling its entries in accordance with an algorithm whose return is difficult to decipher while the scrambled data is still usable for legitimate demonstration [3, 4]. Within personal and limited access use, the algorithm must be unknown for attackers, or at least it must be unlikely to determine it within reasonable time [5, 6]. Another plausible requirement is to

maintain the algorithm as simple as possible due to its frequent application should not affect the operation speed [7, 8].

Basically, the requirement of scrambling algorithm simplicity often prevents from using shufflers based on pseudorandom number generators [9]. The latter utilize rather complex routines for generating a stream of pseudorandom numbers, but the main reason is the seed change [10, 11]. Although the range for the initial value of a pseudorandom sequence is sufficiently wide, the information about the seed change should be constantly sent to the recipient that increases the likelihood of break-in, if the type of the pseudorandom number generator is known [12]. On the other hand, cipher algorithms and cryptographic hashes are high-quality pseudorandom number generators, but generally they are considerably slower than non-cryptographic random number generators [13, 14]. Therefore, it is still desirable to develop a much wider list of simple and fast pseudorandom number generators for tasks of scrambling, regardless of the publicity of its use.

Goal and tasks

Issuing from the growing demands for security and reliability of data privacy and storage policy, the goal is to suggest a new scrambling technique which would provide improved cryptographic properties compared to a peer scrambling technique. The improved cryptographic properties imply decreasing the likelihood of break-in along with decreasing similarity between the initially given data and the scrambled data.

One of the simplest algorithms having been studied for possible scrambling is based on building solutions to the knight open tour problem [15, 16]. In fact, there are a few such algorithms, both exact and heuristic, which have slightly differing computational efficiency and coverage (some solutions may be omitted by a heuristic, whereas they are found by an exact approach, although not always within a reasonable time). A solution of the knight open tour problem is similar to a sequence of pseudorandom numbers used to map data into non-readable yet usable information. The first task is to describe the solution application. The second task is to estimate its main computable parameters for the worst-case scenario compared to an ordinary generator of pseudorandom numbers. The break-in probability and similarity rate will be estimated under the same case along with estimating operation speed and other limitations.

Pseudorandom walk

Denote by

$$\mathbf{U} = [u_I]_F \tag{1}$$

an N-dimensional array of data, where

$$F = \bigtimes_{n=1}^{N} M_n \tag{2}$$

is the format of the array,

$$I = \{i_n\}_{n=1}^{N} \tag{3}$$

is its indexation with indices

$$i_n \in \left\{\overline{1, M_n}\right\} \quad \forall n = \overline{1, N}, \tag{4}$$

M_n is the length of dimension n, $M_n \in \mathbb{N} \setminus \{1\}$, and u_I is a numerical or symbolical entry indexed by (3), (4). The total number of entries in array (1) is:

$$L = \prod_{n=1}^{N} M_n . \tag{5}$$

A scrambler s must take only the starting position:

$$I_1 = \left\{i_n^{(1)}\right\}_{n=1}^{N} \text{ by } i_n^{(1)} \in \left\{\overline{1, M_n}\right\} \quad \forall n = \overline{1, N} \tag{6}$$

for an algorithm A, whereupon the scrambled data is an array

$$\mathbf{Y} = s(\mathbf{U}, A, I_1). \tag{7}$$

Obviously, the number of entries of array (7) is (5), but its format may be changed from (2) into

$$G = \bigtimes_{b=1}^{B} K_b \tag{8}$$

by

$$L = \prod_{b=1}^{B} K_b . \tag{9}$$

So,

$$\mathbf{Y} = [y_J]_G \tag{10}$$

is the scrambled B-dimensional data array of format (8) with indexation

$$J = \{j_b\}_{b=1}^{B} \tag{11}$$

and indices

$$j_b \in \left\{\overline{1, K_b}\right\} \quad \forall b = \overline{1, B}, \tag{12}$$

by dimension b of length K_b, $K_b \in \mathbb{N} \setminus \{1\}$, where

$$\{y_J\}_{\forall J} = \{u_I\}_{\forall I} . \tag{13}$$

In particular, $B = 1$, i. e. array (1) is scrambled into a vector (7) of length (9), where

$$\mathbf{Y} = \left[y_j\right]_{1 \times L} \ (j = \overline{1, L}). \tag{14}$$

Another possibility, less applicable than (14), is dimensionality reduction. For instance, an $M_1 \times M_2 \times M_3$ matrix (that can represent some color image for $M_3 = 3$) is scrambled into an $M_1 \times M_2 M_3$, $M_1 M_2 \times M_3$, or $M_1 M_3 \times M_2$ matrix, regardless of possible transposition. In vectorizing-based applications, an $M_1 \times M_2 \times M_3$ matrix data is scrambled into a $1 \times M_1 M_2 M_3$

vector (obviously, a column vector is also possible).

The knight open tour problem whose solutions appear like a random series of knight positions [15, 16], i. e. a pseudorandom walk, is a promising approach to scrambling. The knight starts its open tour at a horizontal position x_1 and a vertical position y_1 on a chessboard of size $C \times C$, where $x_1 \in \{\overline{1, C}\}$ and $y_1 \in \{\overline{1, C}\}$. Then the knight moves onward according to an algorithm. In this way, the chessboard is completely covered by the knight, and a definite sequence of C^2 integers indicating the successive positions of the knight is formed. This sequence can be formed in two ways, each of which has two versions. If the chessboard squares are enumerated by the number of the knight move, then a matrix

$$\mathbf{B} = [c_{ik}]_{C \times C},$$

where c_{ik} is the number of the knight move at chessboard square $\{i, k\}$ ($c_{ik} = 1$ for $i = y_1$, $k = x_1$), is vectorized either by concatenating its columns or concatenating its rows. On the other hand, a vector

$$\mathbf{M} = [m_j]_{1 \times C^2}$$

is formed, where either

$$m_j = C \cdot (i - 1) + k \text{ for } c_{ik} = j \tag{15}$$

or

$$m_j = C \cdot (k - 1) + i \text{ for } c_{ik} = j. \tag{16}$$

So, the knight pseudorandom walk can be represented either by the vectorized matrix $\mathbf{B}$ or vector $\mathbf{M}$. It is easy to see that the two versions of the vectorized matrix $\mathbf{B}$ are directly connected. Knowing one version, the other one is obtained via a simple permutation pattern. The two versions of vector $\mathbf{M}$ by (15) or (16) are directly connected as well, where one version is obtained from the other one via another simple pattern which is the inverse to either (15) or (16).

Scrambler estimations and parameters

The break-in probability can be estimated as follows. The knight open tour problem has a great deal of solutions for a starting position of the knight depending on the chessboard size $C \times C$. Denote this number by $S_A(C; x_1, y_1)$. There are C^2 starting positions on the chessboard of size $C \times C$. For a definite chessboard size and a definite starting position, there are four versions of the pseudorandom walk for a distinct solution out of $S_A(C; x_1, y_1)$ solutions. Hence, an estimation of the break-in probability is

$$P_{\text{break-in}} = \frac{1}{4C^2 \cdot S_A(C; x_1, y_1)}. \tag{17}$$

On an 8×8 chessboard, for instance, there are more than 10^{14} knight open tours for a starting position, i. e. $S_A(8; x_1, y_1) > 10^{14}$ [17]. Therefore, the break-in probability (17) is

$$P_{\text{break-in}} = \frac{1}{4 \cdot 8^2 \cdot S_A(8; x_1, y_1)} = \frac{1}{256 \cdot S_A(8; x_1, y_1)} < 10^{-16} \tag{18}$$

for an 8×8 chessboard. The ending position of the knight cannot be $\{x_1, y_1\}$ but distinct open tours can end at the same position. However, number $S_A(C; x_1, y_1)$ depends on the starting position. If it is closer to a corner of the chessboard, this number is less than the number of knight open tours starting closer to the chessboard center. This is explained by the knight at a corner has fewer ways to move onward. Thus, many tours have a zigzag pattern on the chessboard borders (Figure 1). If the chessboard size is increased, the pattern may remain (Figure 2).

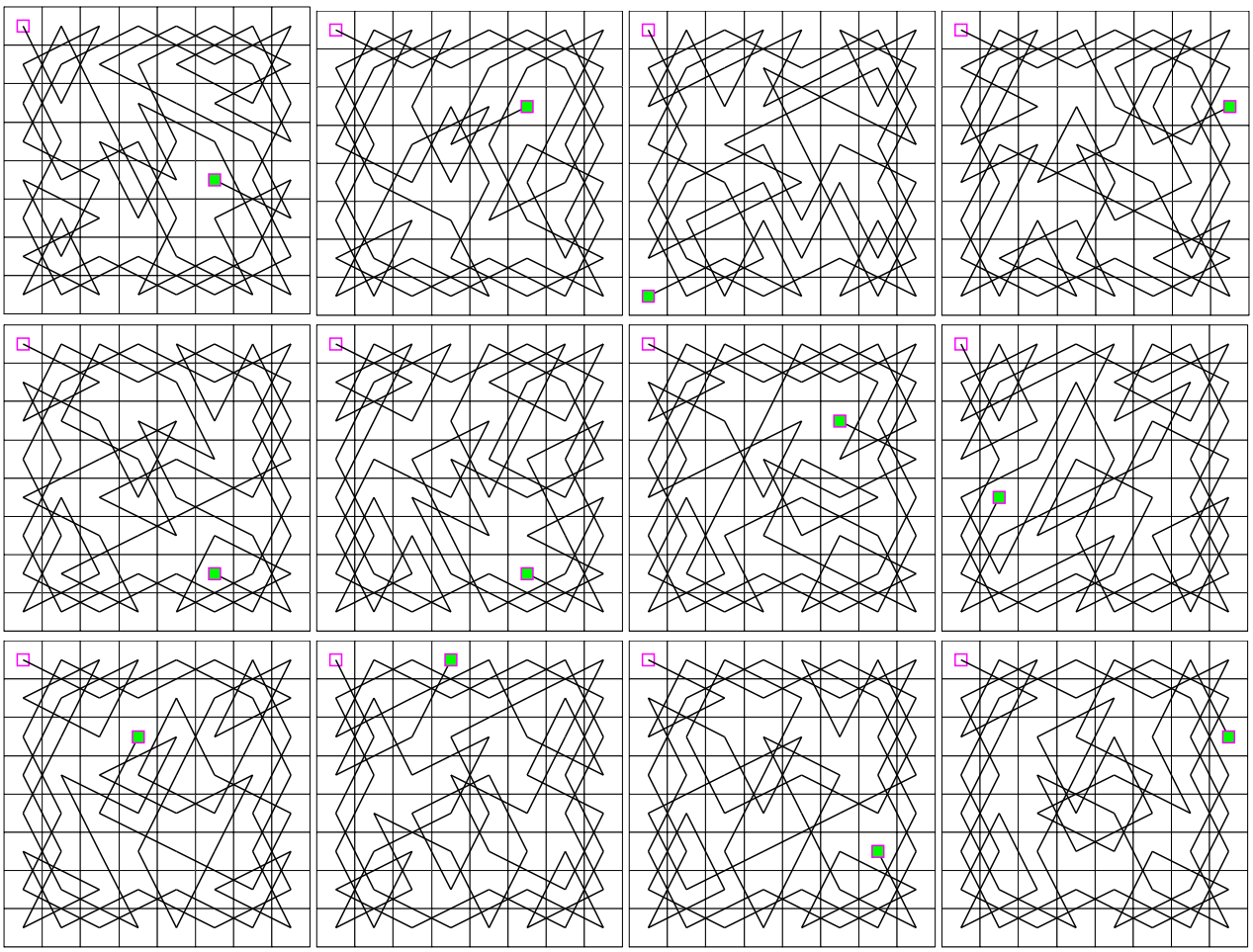

Figure 1. A set of 12 distinct pseudorandom walks on an 8×8 chessboard, each of which starts at the same square (from the top left corner), where the zigzag border pattern is seen

Nevertheless, the zigzag pattern becomes less influential on bigger chessboards. It is quite apparent by comparing pseudorandom walks in Figure 2 to those in Figure 1. This means it is more likely that number $S_A(2C; x_1, y_1)$ varies relatively less across all possible $4C^2$ starting positions on a $(2C) \times (2C)$ chessboard than, say, number $S_A(C; x_1, y_1)$ varying across all possible C^2 starting positions on a $C \times C$ chessboard. Therefore, bigger chessboards are more efficient to produce higher rates of randomness.

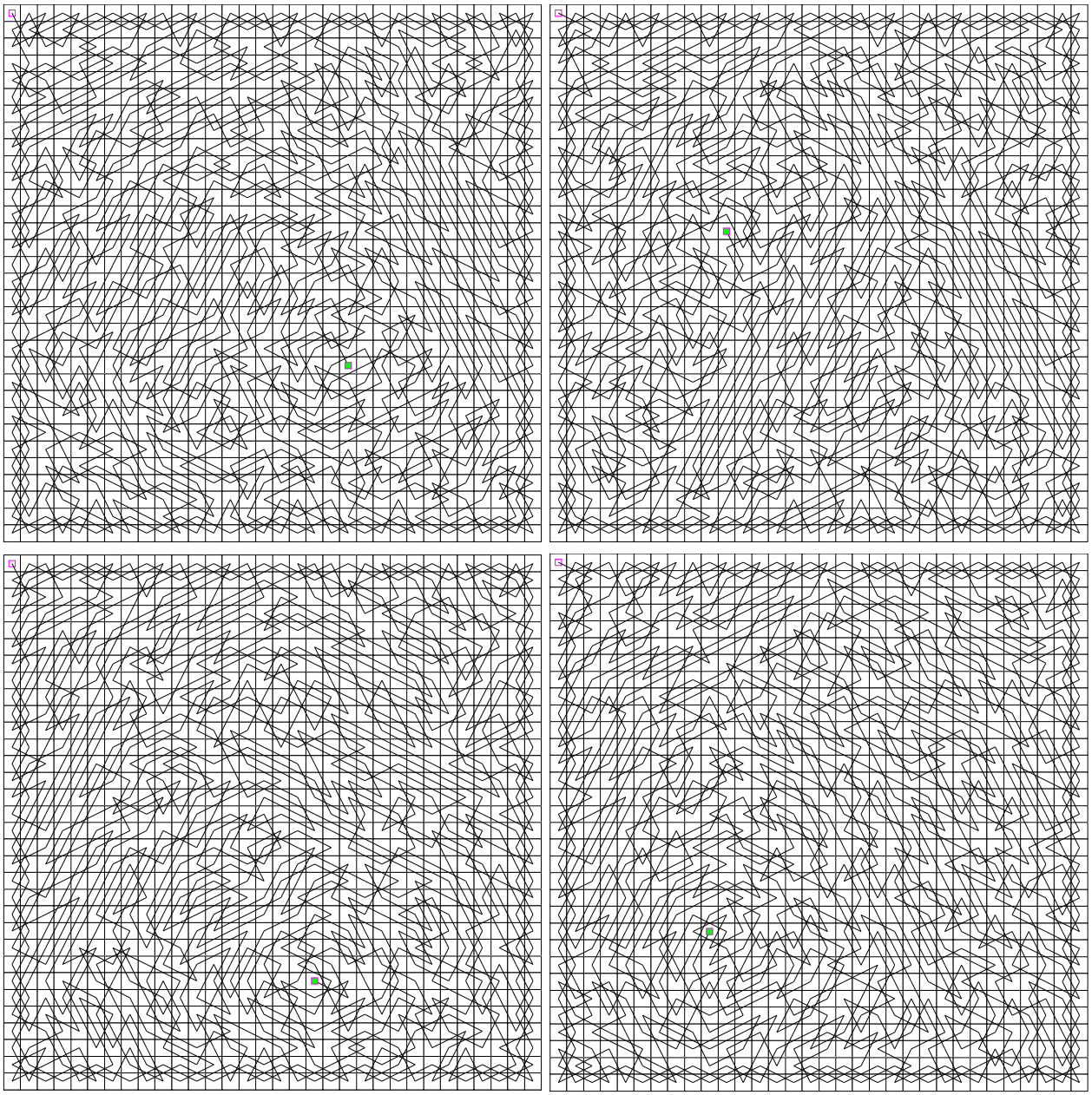

Figure 2. A set of four various pseudorandom walks on a 32×32 chessboard, each of which starts from the top left corner, where the zigzag border pattern is still seen reminiscent of that in Figure 1

So, starting an open tour off one of four corners, either close to it or peculiarly at chessboard squares $\{1,1\}$, $\{1,C\}$, $\{C,1\}$, $\{C,C\}$, is the worst-case scenario for break-in probability (17). In such a case, the break-in probability is higher than in any other case. Compared to shuffler-based scrambling, it is important to know a ratio of break-in probability (17) to the shuffler-based scrambling probability (which, obviously, is L^{-1} as the shuffler algorithm is presumed to be known and it can be defined by the position of an integer value from 1 to L). This ratio, called the break-in probability rate (BPR) for further consideration, must be estimated under the worst-case scenario with using (18) as reference value. Other important parameters are the similarity index and computation time ratio. The latter is calculated separately for scrambling and descrambling as a ratio of knight-open-tour time to shuffler time. The respective scrambling time rate (STR) and

descrambling time rate (DTR) must be close. The similarity index is calculated as the element-wise number of coincidences in multidimensional data array (1) and the scrambled data array (10) divided by L. This index is written as a knight-open-tour scrambler rate (KTSR) and a shuffler scrambler rate (SSR), whereupon a similarity rate (SR) is calculated as a ratio of KTSR to SSR.

Computational experiments are carried out on a single CPU Intel Core i5-7200U@2.50GHz in Matlab 2018a. The data are intended to emulate streaming images, so for this purpose integers between 0 and 255 are generated by a pseudorandom number generator with a known seed. Table 1 presents the mentioned parameters for signed 16-bit integers between 0 and 255 in multidimensional data arrays for $L = C^2$ (here and below the comparative rates are averaged over 100 repetitions), where it is clearly seen that the knight-open-tour scrambler and shuffler scrambler have roughly the same operation speed. SR varies badly, though, for smaller sizes of the chessboard. The estimation of BPR is made with respect to (18), where the tour starts at chessboard square $\{1,1\}$. Approximately the same parameters hold for numbers with single precision storage (Table 2) and numbers with double precision storage (Table 3) almost independently of the data dimensionality (in Matlab, vectorization or reshaping arrays is executed within a few microseconds), where the numbers are generated starting at the same seed of the pseudorandom number generator. Nevertheless, KTSR and SSR are a few times smaller than those in Table 1.

Overall, the scrambler is defined by four parameters: the chessboard size, the starting position, a specific integer σ_A determining one of $S_A(C; x_1, y_1)$ open tours, and one of four ways to obtain a $1 \times C^2$ vector representing the knight pseudorandom walk. Integer σ_A could be called the walk seed and it is a number between 1 and $S_A(C; x_1, y_1)$:

$$\sigma_A \in \overline{\{1, S_A(C; x_1, y_1)\}}.$$

This integer predetermines a distinct knight pseudorandom walk by setting the pseudorandom number generator seed to a definite integer. Algorithm A building the open tour online frequently stumbles over situations when there are a few possible onward moves of the knight. One of such moves is further selected by generating a random value and comparing a threshold to this value.

Table 1. Comparative rates for signed 16-bit integers between 0 and 255

C	10	20	30	40	50	60	70	80	90	100	110	120	130	140
KTSR	0.0339	0.0064	0.0046	0.0047	0.0051	0.0042	0.0043	0.0042	0.0042	0.0042	0.004	0.004	0.004	0.004
SSR	0.0134	0.0063	0.005	0.0045	0.0044	0.0039	0.004	0.0041	0.004	0.004	0.004	0.004	0.004	0.004
SR	2.5298	1.0278	0.922	1.044	1.1799	1.0676	1.0672	1.0194	1.0583	1.0474	1.0006	1.0059	1.0046	0.9925
STR	1.007	1.1648	0.9676	1	0.99	0.9961	0.9717	0.9799	0.9953	0.9637	0.9786	0.9966	0.9672	0.9845
DTR	1.0863	1.1381	1.062	1.0022	0.9934	0.957	1.0126	0.9986	0.9858	1.0097	1.008	1.0123	0.9979	1.0134
BPR	10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-17}	10^{-17}	10^{-17}	10^{-17}	10^{-17}	10^{-17}	10^{-17}

Table 2. Comparative rates for numbers with single precision storage

C	10	20	30	40	50	60	70	80	90	100	110	120	130	140
KTSR	0.03	0.0025	0.0011	0.0006	0.0012	0.0003	0.0004	0.0002	0.0002	0.0003	0.0001	0.0001	0.0001	0.0001
SSR	0.0102	0.0027	0.001	0.0007	0.0004	0.0002	0.0002	0.0002	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
SR	2.9411	0.9345	1.0638	0.9615	2.9411	1.1627	1.9607	1.0309	1.7857	3.2258	1.1111	0.9345	0.8849	1.8867
STR	1.0208	1.0489	0.9749	1.0002	0.9649	0.9911	0.988	0.9781	0.9782	0.9829	0.9705	0.9802	0.9667	0.9877
DTR	1.1964	1.0257	1.0046	0.9985	1.0009	0.9953	1.0032	1.0192	1.0039	1.004	1.0063	0.9963	1.0081	1.0128
BPR	10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-17}	10^{-17}	10^{-17}	10^{-17}	10^{-17}	10^{-17}	10^{-17}

Meanwhile, a scrambling technique can be applied multiple times. Thus, another quadruple of scrambler parameters should be assigned. Table 4 presents the comparative rates for numbers with double precision storage by double scrambling with the same chessboard size, where the tour at the second stage scrambling starts at chessboard square $\{4, 4\}$. In general, these rates do not much differ from those in Tables 1–3, but BPR now is much better. KTSR and SSR are comparable to those in Tables 2 and 3. However, if the second stage scrambling chessboard is of size $(C/2)\times(C/2)$, then KTSR is improved in about six times (Table 5). In addition, STR and DTR, varying between 0.5802 and 0.7547, are much favorable for the knight-open-tour scrambler. Its break-in probability increases, though, due to the smaller second stage scrambling chessboard. The decrement is hardly noticeable, anyway. This is an acceptable tradeoff for decreasing similarity along with speeding up the scrambling (descrambling) process.

Table 3. Comparative rates for numbers with double precision storage

C	10	20	30	40	50	60	70	80	90	100	110	120	130	140
KTSR	0.03	0.0025	0.0011	0.0006	0.0012	0.0003	0.0004	0.0002	0.0002	0.0003	0.0001	0.0001	0.0001	0.0001
SSR	0.0102	0.0027	0.001	0.0007	0.0004	0.0002	0.0002	0.0002	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
SR	2.9411	0.9345	1.0638	0.9615	2.9411	1.1627	1.9607	1.0309	1.7857	3.2258	1.1111	0.9345	0.8849	1.8867
STR	1.0555	1.0086	0.9855	1.0048	0.9898	0.9811	0.988	0.9822	0.9726	0.9973	0.974	0.9701	0.9859	0.9931
DTR	1.2022	1.0001	1.0542	0.9997	1.0339	0.9759	0.9969	1.0064	1.0168	0.9984	1.0122	1.0023	0.9902	1.0099
BPR	10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-17}	10^{-17}	10^{-17}	10^{-17}	10^{-17}	10^{-17}	10^{-17}

Table 4. Comparative rates for numbers with double precision storage by double scrambling

C	10	20	30	40	50	60	70	80	90	100	110	120	130	140
KTSR	0.02	0.0025	0	0.0019	0.0008	0.0017	0.0002	0.0005	0.0005	0.0005	0.0003	0.0002	0.0004	0.0001
SSR	0.0095	0.0026	0.0014	0.0006	0.0004	0.0003	0.0002	0.0002	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
SR	2.1052	0.9615	0	2.9126	1.8181	6.0606	1.0101	2.9126	4.4943	4.8076	3.8834	2.9702	6.0606	0.909
STR	1.1184	1.0018	0.9893	0.9946	0.9772	0.9718	0.9729	0.9886	0.9756	0.9822	0.9916	0.9753	0.9861	0.9838
DTR	1.1708	0.9912	1.0369	1.0016	0.9986	0.9978	1.0037	0.9987	1.0128	0.9912	1.0044	1.001	0.9956	1.0101
BPR	10^{-18}	10^{-18}	10^{-19}	10^{-19}	10^{-19}	10^{-19}	10^{-19}	10^{-20}	10^{-20}	10^{-20}	10^{-21}	10^{-21}	10^{-21}	10^{-21}

Table 5. Comparative rates for numbers with double precision storage by double scrambling with $C\times C$ and $(C/2)\times(C/2)$ chessboards

C	10	20	30	40	50	60	70	80	90	100	110	120	130	140
KTSR	0	0	0	0.0013	0.0008	0.0003	0.0002	0.0002	0.0005	0.0005	0.0003	0.0003	0.0002	0.0002
SSR	0.0098	0.0025	0.0011	0.0006	0.0004	0.0003	0.0002	0.0002	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
SR	0	0	0	2.2471	2.1052	1.0869	0.9523	0.9009	4.4943	4.8543	3.9603	3.9215	4.1666	3.0612
STR	0.7197	0.6731	0.6375	0.6593	0.6351	0.6246	0.6135	0.5992	0.5976	0.6063	0.5927	0.5931	0.5859	0.5802
DTR	0.7547	0.6669	0.6742	0.6499	0.6252	0.6331	0.6288	0.6164	0.6127	0.6047	0.5895	0.5983	0.5855	0.5811
BPR	10^{-17}	10^{-17}	10^{-18}	10^{-18}	10^{-18}	10^{-18}	10^{-18}	10^{-19}	10^{-19}	10^{-19}	10^{-20}	10^{-20}	10^{-20}	10^{-20}

It is noteworthy that the chessboard can be three-dimensional [18]. This leads to further decreasing the break-in probability. Indeed, the knight open tour problem has a far greater deal of solutions for a starting position of the knight on a chessboard of size $C\times C\times C$. For a depth position z_1 on a chessboard of size $C\times C\times C$, where $z_1 \in \{\overline{1, C}\}$, denote the number of solutions starting off position (cube) $\{x_1, y_1, z_1\}$ by $S_A(C; x_1, y_1, z_1)$. There are C^3 starting positions on the chessboard of size $C\times C\times C$. For a definite chessboard size and a definite starting position, there are four versions of the pseudorandom walk for every face of the chessboard. Having six faces, the number of versions of the pseudorandom walk is $6\cdot 4^C$, and a raw estimation of the break-in probability is

$$P_{\text{break-in}} = \frac{1}{6\cdot 4^C \cdot C^3 \cdot S_A(C; x_1, y_1, z_1)} \tag{19}$$

being much lower than (17) for the same number C. However, estimation (19) is made for a one sequence of C layers of the three-dimensional chessboard (successively from layer 1 to layer C, going through a face). Altogether there are $C!$ such sequences. So, a more precise estimation is

$$P_{\text{break-in}} = \frac{1}{6\cdot 4^C \cdot C!\cdot C^3 \cdot S_A(C; x_1, y_1, z_1)}. \tag{20}$$

It is expected that an $8\times 8\times 8$ chessboard has far more than 10^{14} knight open tours for a starting position, so $S_A(8; x_1, y_1, z_1) > 10^{14}$ at least. Therefore, the break-in probability (20) is

$$P_{\text{break-in}} = \frac{1}{6\cdot 4^8 \cdot 8!\cdot 8^3 \cdot S_A(C; x_1, y_1, z_1)} = \frac{1}{8117488189440\cdot S_A(8; x_1, y_1, z_1)} < 10^{-26} \tag{21}$$

for an $8\times 8\times 8$ chessboard. However, computing knight open tours on a three-dimensional chessboard may run into known computational issues [15, 16, 19], where searching for a specific knight pseudorandom walk may become intractable (when it cannot be completed within reasonable amount of time) due to a significantly deep “path” [20]. After all, the existence of solutions of the knight open tour problem on three-dimensional chessboards has not been proved yet for any size.

Discussion

Given a seed integer for a pseudorandom number generator, a distinct knight pseudorandom walk (out of a manifold of pseudorandom walks) is built online. This means that the scrambled data vector is built online as well, i. e. every next knight move is immediately followed by moving an entry of array (1) to a specific place. As any knight open tour algorithm has linear runtime complexity [15], the knight-open-tour scrambler does not make it any (significantly) longer than other scramblers.

Square chessboards are better than non-squarely-shaped chessboards as they contain richer manifolds of knight pseudorandom walks. Indeed, cornered start positions of the knight produce poorer manifolds. A square chessboard has the fewest corner-like start positions. Contrary to that, a rectangular chessboard has more corner-like start positions closer to the shorter side. Say, a horizontally stretched chessboard has corner-like start positions closer to the left and right sides because the knight is more “squeezed” there having fewer possible moves upwards and downwards than in the direction to the center.

The estimations of BPRs in Tables 1–5 may look too pessimistic, but they are considered as a “lowest” bound of the gain in security provided by the knight-open-tour scrambler. This bound might have been much bettered for bigger chessboards, but the number of the knight open tour problem solutions is itself an open question for such chessboards. However, if to step aside a little from the worst-case scenario, by assuming that another 10 squares along the chessboard dimension decreases the BPR by the factor of 2, a BPR estimate for $C = 140$ is 10^{-20} to 10^{-22}. It is also likely that the estimations by (18) and (21) are better in most non-corner-like cases. Owing to using the ratios, the reported comparative rates are expected to be independent of the hardware. However, it is worth noting that the real-time operation speed of the knight-open-tour scrambler has a limit being determined by the hardware performance. Thus, 19600 double-precision values are scrambled within 150 milliseconds on a single CPU Intel Core i5-7200U@2.50GHz, which is 7.975 Mb/s in terms of the speed. This means that the speed of streaming data intended for scrambling must not exceed 7.975 Mb/s for such a hardware. This limit, being the worst-case scenario, keeps roughly the same owing to the linear runtime complexity of the knight open tour algorithm. Moreover, the limit is quite comparable to the speed of shuffler-based scrambling and scrambling by pseudorandom binary numbers generated with using linear-feedback shift registers. Nevertheless, no limits exist for data privacy and storage purposes, where the data are stationary and the only intention is to store it securely protected.

Whichever the chessboard size is, the number of chessboard squares must be not fewer than the data length by (5). Obviously, the redundant chessboard squares are not used in the case when a multidimensional data array has fewer entries than the number of chessboard squares.

Conclusion

Multidimensional data array (1) is scrambled into array (7) by starting position (6) and algorithm A, whereupon the scrambled data array (10) has format (8) with indexation by (11)–(13). To further improve balance of the scrambling simplicity and productivity, solutions of the knight open tour problem are used. The break-in probability is dramatically decreased by taking into account a knight pseudorandom walk manifold for a starting position on a given chessboard. The

number of possible solutions is gigantic for an 8×8 chessboard, let alone bigger chessboards. Thus, a pessimistic estimation of the break-in probability for an 8×8 chessboard is less than 10^{-16}. A similarly expected estimation for an $8\times8\times8$ chessboard is less than 10^{-26}. Meanwhile, the similarity index is acceptable, rapidly dropping as the chessboard size is increased (for bigger multidimensional data arrays).

Compared to an ordinary generator of pseudorandom numbers, the knight-open-tour scrambler shuffles data also, having the same computational efficiency, but it has 10^{16} to 10^{21} times lower break-in probability depending on the chessboard size and the starting position of the knight. A knight open tour problem solution, also referred to as a knight pseudorandom walk, is determined by the chessboard size, the starting position, the way to vectorize the knight pseudorandom walk, and the pseudorandom number generator seed allowing to specifically move the knight onward through situations with multiple possible moves of the knight.

From the practical point of view, the knight-open-tour scrambler has a limited operation speed of 7.975 Mb/s while the data is streamed, whereas there is no such a limitation in securely storing stationary data. The data dimensionality has no impact on the scrambler performance, but the chessboard size should be consistent with the data length. Overall, the knight-open-tour scrambler is mainly intended for private use and corporate body security purposes including business and governmental agencies.

References

1. Pandya, P. (2013). Advanced data encryption. In *Cyber Security and IT Infrastructure Protection* (pp. 325–345). Elsevier. https://doi.org/10.1016/B978-0-12-416681-3.00015-X
2. Pandya, P. (2013). Advanced data encryption. In *Computer and Information Security Handbook* (pp. 1127–1138). Elsevier Inc. https://doi.org/10.1016/B978-0-12-394397-2.00070-2
3. Li, N., Zhang, N., Das, S. K., & Thuraisingham, B. (2009). Privacy preservation in wireless sensor networks: A state-of-the-art survey. *Ad Hoc Networks*, *7*(8), 1501–1514. https://doi.org/10.1016/j.adhoc.2009.04.009
4. De Capitani di Vimercati, S., Foresti, S., Livraga, G., & Samarati, P. (2022). Digital infrastructure policies for data security and privacy in smart cities. In *Smart Cities Policies and Financing: Approaches and Solutions* (pp. 249–261). Elsevier. https://doi.org/10.1016/B978-0-12-819130-9.00007-3
5. Waheed, A., Subhan, F., Suud, M. M., Alam, M. M., & Haider, S. (2023). Design and optimization of nonlinear component of block cipher: Applications to multimedia security. *Ain Shams Engineering Journal*, 102507. https://doi.org/10.1016/j.asej.2023.102507
6. Fair, T., Nordfelt, M., Ring, S., & Cole, E. (2005). Spying Basics. In *Cyber Spying* (pp. 45–86). Elsevier. https://doi.org/10.1016/b978-193183641-8/50005-8
7. Stapko, T. (2008). Choosing and optimizing cryptographic algorithms for resource-constrained systems. In *Practical Embedded Security* (p. 149–171). Elsevier. https://doi.org/10.1016/b978-075068215-2.50009-4

8. Sun, P. (2020). Security and privacy protection in cloud computing: Discussions and challenges. *Journal of Network and Computer Applications*, *160*, 102642. https://doi.org/10.1016/j.jnca.2020.102642
9. Bakiri, M., Guyeux, C., Couchot, J.-F., & Oudjida, A. K. (2018). Survey on hardware implementation of random number generators on FPGA: Theory and experimental analyses. *Computer Science Review*, *27*, 135–153. https://doi.org/10.1016/j.cosrev.2018.01.002
10. L'Ecuyer, P. (2006). Chapter 3. Uniform random number generation. In *Handbooks in Operations Research and Management Science. Vol. 13* (p. 55–81). Elsevier. https://doi.org/10.1016/S0927-0507(06)13003-0
11. Plesser, H. E., & Jahnsen, A. G. (2010). Re-seeding invalidates tests of random number generators. *Applied Mathematics and Computation*, *217*(1), 339–346. https://doi.org/10.1016/j.amc.2010.05.066
12. Dunn, W. L., & Shultis, J. K. (2012). Pseudorandom number generators. In *Exploring Monte Carlo Methods* (pp. 47–68). Elsevier. https://doi.org/10.1016/b978-0-444-51575-9.00003-8
13. Bowman, K. P. (2005). Statistics and pseudorandom numbers. In *An Introduction to Programming with IDL* (pp. 227–235). Elsevier. https://doi.org/10.1016/b978-012088559-6/50024-8
14. Ross, S. M. (2023). Random numbers. In *Simulation* (p. 39–45). Elsevier. https://doi.org/10.1016/b978-0-32-385738-3.00008-0
15. Parberry, I. (1997). An efficient algorithm for the Knight's tour problem. *Discrete Applied Mathematics*, *73*(3), 251–260. https://doi.org/10.1016/S0166-218X(96)00010-8
16. Lin, S.-S., & Wei, C.-L. (2005). Optimal algorithms for constructing knight's tours on arbitrary $n \times m$ chessboards. *Discrete Applied Mathematics*, *146*(3), 219–232. https://doi.org/10.1016/j.dam.2004.11.002
17. Weisstein, E. W. Knight Graph. URL: https://mathworld.wolfram.com/KnightGraph.html
18. Bai, S., Yang, X.-F., Zhu, G.-B., Jiang, D.-L., & Huang, J. (2010). Generalized knight's tour on 3D chessboards. *Discrete Applied Mathematics*, *158*, 1727–1731. https://doi.org/10.1016/j.dam.2010.07.009
19. Kyek, O., Parberry, I., Wegener, I. (1997). Bounds on the number of knight's tours. *Discrete Applied Mathematics*, *74*(2), 171–181. https://doi.org/10.1016/S0166-218X(96)00031-5
20. Romanuke, V. V. (2022). Finite uniform approximation of two-person games defined on a product of staircase-function infinite spaces. *International Journal of Approximate Reasoning*, *145*, 139–162. https://doi.org/10.1016/j.ijar.2022.03.005

The manuscript is received – 05/12/2023; accepted – 13/02/2024.

Різноманіття псевдовипадкових блукань шахового коня для скремблювання багатовимірних даних

Вадим Романюк

Анотація

Задача відкритого циклу шахового коня полягає у побудові послідовності ходів шахового коня, яка повністю покриває шахову дошку без повторів, де початкове та кінцеве положення є завжди різними. Розв'язок задачі відкритого циклу шахового коня подібний до послідовності псевдовипадкових чисел, за якими можна відобразити дані у корисну інформацію без можливості її читання. Задача відкритого циклу шахового коня для певного стартового положення має різноманіття розв'язків, кількість яких залежить від розміру шахової дошки. Розв'язки задачі відкритого циклу шахового коня виглядають як його псевдовипадкове блукання або як випадкова послідовність його положень. Ці розв'язки використовуються для подальшого покращення балансу простоти скремблювання та продуктивності, де головними показниками є ймовірність зламу та індекс подібності. Ймовірність зламу суттєво зменшується завдяки різноманіттю псевдовипадкових блукань шахового коня для певного стартового положення на даній шаховій дошці. Песимістична оцінка ймовірності зламу для шахової дошки розміром 8×8 є меншою за 10^{-16}. Аналогічна оцінка для шахової дошки розміром 8×8×8 є меншою за 10^{-26}. Конкретна реалізація псевдовипадкового блукання шахового коня будується в режимі онлайн за заданого початкового цілого для генератора псевдовипадкових чисел. Вектор даних після скремблювання також будується в режимі онлайн за лінійної часової складності. Індекс подібності є прийнятним. Він стрімко падає зі зростанням розміру шахової дошки (для більших масивів багатовимірних даних). Конкретне псевдовипадкове блукання шахового коня визначається розміром шахової дошки, початковим положенням, методом векторизації псевдовипадкового блукання шахового коня, а також початковим цілим для генератора псевдовипадкових чисел. Ці параметри визначають специфічний рух коня у ситуаціях, коли постають множинні варіанти подальшого руху. Скремблер на основі відкритого циклу шахового коня має від 10^{16} до 10^{21} разів меншу ймовірність зламу, порівняно зі звичайним псевдовипадковим генератором, залежно від розміру шахової дошки та початкового положення шахового коня.

Ключові слова: скремблювання даних; псевдовипадкове блукання шахового коня; ймовірність зламу; рівень подібності.

УДК 004.43 : 004.032.24

Дослідження ефективності розпаралелювання процесорозалежних задач мовою Python на основі механізму потоків

Роман Бабаков
доцент, д-р техн. наук
ORCID: 0000-0001-7196-0912
r.babakov@donnu.edu.ua

Донецький національний університет імені Василя Стуса

Олександр Баркалов
професор, д-р техн. наук
ORCID: 0000-0002-4791-2088
a.barkalov@imei.uz.zgora.pl

Університет Зеленогурський

Ключові слова:
процесорозалежні задачі;
швидкість обробки даних;
розпаралелювання;
механізм потоків;
мова Python.

Метою роботи є дослідження ефективності застосування паралельної обробки даних на основі потоків у мові Python для розв'язання задач, які потребують значних ресурсів центрального процесора. В якості такої задачі розглядалась задача обробки двовимірного масиву великих розмірів стандартними засобами Python без використання спеціалізованих бібліотек на кшталт Numpy. Виявлено непрогнозоване зменшення часу виконання багатопотокової програми на 30% у порівнянні з послідовною (непаралельною) реалізацією розв'язуваної задачі. Це дає змогу поставити під сумнів загальновідоме твердження про те, що багатопотокове розв'язання процесорозалежних задач саме в мові Python не є ефективним, оскільки не приводить до зменшення часу роботи програм. Раніше вважалося, що завдяки використанню так званого глобального замка (GIL) інтерпретатор мови Python у разі виконання багатопотокових програм із застосуванням модуля Threading спрямовує усі потоки програми на одне процесорне ядро, що виключає фізичне паралельне виконання потоків і за жодних умов не дає змоги зменшити час роботи програми. Відомою рекомендацією для отримання реального виграшу в часі є використання розпаралелювання на основі механізму процесів і модуля «multiprocessing», який допомагає задіяти кілька фізичних ядер процесора. Однак такий підхід потребує додаткових ресурсів процесора (ядер) та додаткових витрат часу на обмін даними між процесами, що може звести нанівець ефект від багатоядерної обробки. Проведені авторами експерименти довели, що застосування багатопотокового підходу також може бути доцільним у випадку процесорозалежних задач, оскільки виграш у часі, що досягається, не потребує додаткових ресурсів з боку центрального процесора комп'ютера та зайвих витрат часу на обмін даними між потоками.

DOI: 10.31558/2786-9482.2024.1.2

Вступ

Технологічний прогрес наблизився до межі можливостей збільшення швидкодії процесорних елементів комп'ютерної техніки. Частота роботи процесора на рівні 3–5 ГГц

була досягнута більше 10 років тому і зупинилась на цьому рівні. Подальше суттєве зростання частоти виявилось фізично неможливим, оскільки обмежується максимальною швидкістю проходження електричного струму в провідниках як у процесорному ядрі, так і між процесором та оперативною пам'яттю [1, 2].

Перспективним напрямом збільшення продуктивності обчислювальних систем виявилося застосування багатопроцесорних архітектур [3, 4]. У них використовується кілька процесорних елементів, які мають можливість працювати паралельно та зазвичай реалізуються у вигляді процесорних ядер на єдиному кристалі процесора. У сфері програмного забезпечення спостерігається переорієнтація на багатоядерні архітектури, що відображається на можливостях операційних систем і прикладних програм. У цьому аспекті очікуваним є зростання підтримки реалізації паралельних алгоритмів багатьма мовами програмування. Технології, як-от OpenMP, MPI та CUDA, добре відомі розробникам програмного забезпечення і дають змогу задіяти додаткове апаратне забезпечення для пришвидшення розв'язання обчислювальних задач [1, 3]. Альтернативою цим технологіям є використання функціоналу операційних систем, щоб задіяти ресурси сучасних багатоядерних процесорів.

Мова програмування Python містить 2 модулі підтримки паралельних алгоритмів [5–7]. Модуль *Multiprocessing* реалізує паралелізм на основі механізму процесів. Програма, написана з використанням цього модуля, дає змогу запустити декілька процесів, кожен із яких може виконуватись на окремому процесорному ядрі і має відокремлену область пам'яті. Перевага таких програм полягає у можливості одночасного виконання окремих ділянок програми, недоліком є значні витрати часу на обмін даними між процесами, які можуть нівелювати ефект від мультипроцесорної обробки даних.

Іншим модулем для розробки паралельних програм є модуль *Threading*, який реалізує паралелізм на основі механізму потоків. Усі потоки, які створено однією програмою, використовують спільну область пам'яті, що значно спрощує і пришвидшує обмін даними між потоками. Однак інтерпретатор мови Python запускає потоки лише на одному процесорному ядрі у режимі конкуренції [1, 6, 7]. Так само на одному ядрі виконуються звичайні послідовні Python-програми, що не використовують жодних засобів розпаралелювання. В тих Python-програмах, які значну частину часу витрачають на операції вводу-виводу, паралелізм на основі потоків дає змогу отримати значну економію часу. Наприклад, виконання одночасних запитів до великої кількості вебсайтів не потребує значних витрат з боку центрального процесора і може бути легко виконано на одному процесорному ядрі у псевдопаралельному режимі. Натомість складні обчислювальні алгоритми, які потребують безперервного використання процесора, у випадку багатопотокової реалізації будуть виконуватись не швидше, а довше, порівняно з їх послідовною реалізацією. Збільшення часу обумовлене додатковими діями інтерпретатора Python і операційної системи, спрямованими на створення, запуск і завершення множини потоків.

Отже, паралелізм на основі потоків є непридатним для розв'язання задач, що потребують значної потужності процесора. Втім це твердження є справедливим лише для мови Python, що зазначено у документації модуля Threading [5, 6]. Причина полягає у тому, що інтерпретатор Python використовує так званий глобальний замок інтерпретатора (GIL –

Global Interpreter Lock), який дає змогу виконувати байт-код тільки одному потоку в один момент часу. Застосування GIL забезпечує виконання тільки однієї атомарної інструкції за раз і гарантує безпеку та захист даних у багатопотокових програмах. Це стосується версії Python 3.12, що є актуальною сьогодні. Розробники Python мають плани на переробку GIL з метою одночасного доступу потоків до кількох ядер процесора, однак наразі використання механізму потоків для розпаралелювання складних обчислювальних алгоритмів вважається недоцільним.

Мета статті полягає в експериментальному дослідженні ефективності використання розпаралелювання програми на основі механізму потоків для розв'язання задачі, що потребує значних ресурсів центрального процесора, за критерієм витрат часу на виконання програми.

Постановка завдання дослідження

У роботі здійснюється експериментальна перевірка припущення про те, що використання паралелізму на основі потоків під час розв'язання процесорозалежних задач мовою Python не є доцільним і не приводить до зменшення часу роботи програми.

Для проведення експерименту обрано задачу з обробки двовимірного масиву, яка формулюється так.

Заданий двовимірний масив розміром M рядків на N стовпців, кожним елементом якого є псевдовипадкове ціле число в діапазоні від –1000 до 1000. Знайти серед усіх елементів масиву кількість таких елементів, для яких сума цифр у значенні елементу, незалежно від знака числа, дорівнює 10 (наприклад, 91, –253, 730, –28, 55). Потім замінити на це число усі елементи масиву, які є квадратами цілих чисел.

Запропоновано такий порядок проведення експерименту.

1. Розв'язати поставлену задачу за допомогою мови Python без використання засобів паралельного програмування. Результатом має бути послідовна програмна реалізація алгоритму задачі.
2. Розв'язати поставлену задачу за допомогою мови Python із використанням паралелізму на основі потоків на базі модуля Threading. Результатом має бути паралельна програмна реалізація алгоритму задачі.
3. Виміряти тривалість розв'язання задачі за послідовної і паралельної програмних реалізацій для різних розмірів масиву.
4. Порівняти результати виміру часу та зробити відповідні висновки.

Очікуваним результатом експерименту є те, що час роботи паралельної програми буде рівним або більшим за час роботи послідовної програми. Це пов'язано з тим, що в мові Python багатопотокові програми примусово виконуються на одному ядрі процесора. Отже, відсутня апаратна складова, що дає змогу прискорити виконання багатопотокової програми, порівняно з її послідовною версією.

Розв'язання задачі у вигляді послідовної реалізації

Розглянемо реалізацію алгоритму розв'язання поставленої задачі у послідовному вигляді, тобто без розпаралелювання.

Задамо розміри масиву у вигляді констант. Створимо двовимірний масив розміром $M \times N$, заповнений псевдовипадковими цілими числами в діапазоні [–1000, 1000]:

```
import random
M, N = 3, 4
mas = []                      # Підготовка порожнього масиву рядків
for i in range(0, M):         # Цикл за рядками
    mas_i = []                # Підготовка порожнього рядка
    for j in range(0, N):                # Цикл за стовпцями
        n = random.randint(-1000, 1000) # Генерація рандомного числа
        mas_i.append(n)                  # Додавання числа у рядок
    mas.append(mas_i)                    # Додавання рядка в масив
```

Знайдемо в масиві кількість елементів, сума цифр яких, незалежно від знака числа, дорівнює 10. Для зберігання кількості знайдених елементів будемо використовувати змінну `n10`:

```
n10 = 0
```

Організуємо вкладений цикл, всередині якого будемо брати з масиву черговий елемент та рахувати суму його цифр. Програмний код може виглядати так:

```
for i in range(0, M):            # Цикл за рядками
    for j in range(0, N):        # Цикл за стовпцями
        n = mas[i][j]            # Беремо елемент масиву
        s = str(n)               # Переводимо в символьний формат
        s = s.replace("-", "")   # Видаляємо знак "мінус", якщо він є
        m = 0                    # Початкове значення суми цифр
        for c in s:              # Перебирання символів рядка
            m += int(c)          # Додаємо цифру до суми
        if m == 10:              # Якщо сума цифр дорівнює 10
            n10 = n10 + 1        # Збільшуємо кількість
            print(n, end=" ")
print("\nЗнайдено чисел:", n10)
```

У наведеному фрагменті передостання команда `print` додана для перевірки працездатності програми і виводить на екран усі елементи масиву, сума цифр яких дорівнює 10. Ця команда потрібна лише на етапі тестування програми. Далі вона буде видалена, оскільки виведення інформації на екран може займати значний відсоток часу роботи програми.

Результат роботи програми для масиву розміром 10×10 виглядає так:

```
145 -64 712 640 -307 631 541 -181
Знайдено чисел: 8
```

Розв’яжемо другу частину задачі: замінимо всі елементи масиву, що є квадратами цілих чисел, на знайдене число `n10`. Для цього будемо проглядати в циклі всі елементи масиву та перевіряти, чи є черговий елемент додатним і чи є корінь із нього цілим числом. Для цього використаємо такий фрагмент програмного коду:

```
for i in range(0, M):              # Цикл за рядками
  for j in range(0, N):            # Цикл за стовпцями
    n = mas[i][j]                  # Беремо елемент масиву
```

```
if n > 0:                    # Якщо число додатне
   if n ** 0.5 == int(n ** 0.5):  # Якщо корінь є цілим числом
      mas[i][j] = n10        # Робимо заміну елементу на n10
      print("[", i, "][", j, "] = ",n, " -> ", n10, sep = "")
```

Як і в попередньому фрагменті, остання команда `print` потрібна лише на етапі налагодження програми.

Повний лістинг програми утворюється послідовним об'єднанням розглянутих вище фрагментів коду. Задавши розміри масиву $M = 10$, $N = 10$ і запустивши програму, отримаємо такий результат:

```
-451 505 -640 -343 -235 -82 64
Знайдено чисел: 7
[4][5] = 121 -> 7
[5][2] = 9 -> 7
[6][4] = 784 -> 7
[8][1] = 64 -> 7
```

У прикладі знайдено 7 чисел, сума цифр яких дорівнює 10. Потім було знайдено чотири числа, що є квадратами цілих чисел: 121, 9, 784, 64. Усі числа були замінені на число 7.

Масив розміром 10×10 не є достатньо великим, щоб у ньому зустрічалось багато псевдовипадкових чисел, що є квадратами цілих чисел. У наведеному вище прикладі таких чисел лише чотири. Якщо запустити програму ще кілька разів, результати можуть бути такими:

```
-370 -244 -280 190
Знайдено чисел: 4
[0][3] = 784 -> 4
[3][3] = 196 -> 4

-235 -190 -55 631 424 -730 910 -442 514
Знайдено чисел: 9
[3][2] = 484 -> 9

523 73 145 550 -811 -190 433 118 244 352
Знайдено чисел: 10
```

В останньому прикладі квадрати цілих чисел взагалі не були знайдені, і це цілком можлива ситуація для масиву невеликого розміру. Якщо задати розміри масиву як 20×20, знайдених квадратів зазвичай буде більше:

```
-55 -370 -370 226 -154 802 118 -271 -316 -253 -505 802 127 -19 352 253 -361 -190 433 433 -712 82 154 -64 -307
Знайдено чисел: 25
[1][14] = 676 -> 25
[3][9] = 529 -> 25
[4][5] = 625 -> 25
[4][14] = 121 -> 25
[7][0] = 900 -> 25
[12][2] = 25 -> 25
[18][0] = 49 -> 25
```

Додамо в програму команди для виміру часу виконання фрагментів коду. Для цього скористаємось можливостями стандартного модуля Time. Перший вимір поточного часу здійснюється відразу після псевдовипадкової генерації масиву, другий вимір – відразу після завершення програми. Різниця в часі буде дорівнювати кількості секунд, що витрачені на

виконання відповідного коду програми. Структурно розташування команд для виміру часу виглядатиме так:

```
import time
# Псевдовипадкова генерація масиву
t1 = time.time()         # Початковий час
# Обробка масиву
t2 = time.time()         # Кінцевий час
print("Час роботи програми:", t2-t1, "секунд.")
```

Розв'язання задачі у вигляді багатопотокової реалізації

Реалізуємо алгоритм розв'язання поставленої задачі обробки двомивірного масиву з використанням механізму потоків.

Задача складається з двох частин. У першій частині шукається кількість елементів масиву, сума цифр яких дорівнює 10. У другій частині шукаються і замінюються елементи, що є квадратами цілих чисел. У кожній частині іде робота з усіма елементами масиву, причому порядок перегляду елементів масиву для цієї задачі не має значення. Це дає змогу задіяти для обробки масиву декілька дочірніх потоків. Кожному потоку можна надавати якийсь фрагмент масиву, і потік буде виконувати завдання тільки для нього. Коли потік завершуватиме роботу, він передаватиме головному потоку результати своєї роботи і завершуватиметься. Головний потік буде швидко обробляти результати роботи дочірніх потоків.

Спроєктуємо програму так, щоб кожен дочірній потік виконував обробку тільки одного рядка двовимірного масиву. Результатом роботи є кількість цільових елементів, які містяться в цьому рядку масиву. Наприкінці роботи дочірнього потоку знайдена кількість додається до глобальної змінної `n10`, в якій ведеться накопичення результатів роботи кожного потоку. В даному випадку необхідна кількість потоків дорівнює кількості рядків масиву.

Команди створення і заповнення масиву залишимо такими, як у попередніх прикладах, за винятком того, що на початку програми під'єднаємо модуль Threading командою `import threading`.

Додамо в програму глобальний замок типу threading.Lock. Він буде застосовуватися для монопольного використання потоками команд `print` та доступу до глобальних змінних з метою упередження станів перегонів (race conditions):

```
L = threading.Lock()                # Замок
```

Як і для послідовної реалізації, створимо змінну `n10`, у якій буде накопичуватись кількість знайдених елементів масиву.

Напишемо функцію `f1`, яка приймає номер рядка двовимірного масиву, виконує в межах цього рядка пошук кількості елементів, сума цифр яких дорівнює 10, та додає цю кількість до глобальної змінної `n10`:

```
def f1(i):                    # Пошук у рядку з номером i
    global n10                # Будемо змінювати глобальну змінну
    k = 0                     # Кількість знайдених елементів у рядку
    for j in range(0, N):     # Цикл за стовпцями
```

```
        n = mas[i][j]         # Беремо елемент масиву
        s = str(n)            # Переводимо в символьний формат
        s = s.replace("-", "")  # Видаляємо знак "мінус", якщо він є
        m = 0                 # Початкове значення суми цифр
        for c in s:           # Перебирання символів рядка
            m += int(c)       # Додаємо цифру до суми
        if m == 10:           # Якщо сума цифр дорівнює 10
            k = k + 1         # Збільшуємо кількість
    L.acquire()
    n10 += k                  # Збільшуємо глобальну змінну
    L.release()
```

Функція працює безпосередньо з глобальним масивом `mas`. Це є можливим завдяки тому, що всі дочірні потоки в Python працюють з однією областю пам'яті і мають безпосередній доступ до глобальних змінних.

Створимо порожній список потоків, після чого в циклі створимо *M* дочірніх потоків. Кожен потік створимо на базі функції `f1` і передамо до неї номер рядка масиву:

```
t_list = []                   # Підготовка списку потоків
for i in range(0, M):         # Цикл зі створення потоків
    t = threading.Thread(target=f1, args=(i, ))
    t_list.append(t)
```

Виконаємо 2 цикли: цикл із запуску потоків та цикл з очікування потоків:

```
for t in t_list:              # Цикл із запуску потоків
    t.start()
for t in t_list:              # Цикл з очікування потоків
    t.join()
```

Після того, як робота всіх потоків завершиться, у змінній `n10` буде знаходитись кількість елементів масиву, сума цифр яких дорівнює 10.

Тепер розпаралелимо другу частину завдання, а саме процес заміни елементів масиву, що є квадратами цілих чисел, на значення кількості елементів, сума цифр яких дорівнює 10, тобто на значення змінної `n10`.

Як і для першої частини завдання, розпаралелимо обробку масиву на рівні окремих рядків – створимо на кожен рядок масиву один потік. Для цього напишемо функцію `f2`, яка приймає як аргумент номер рядка двовимірного масиву, проглядає цей рядок та замінює елементи, що є квадратами цілих чисел, на значення глобальної змінної `n10`:

```
def f2(i):                  # Заміна у рядку з номером i
  global mas                # Можливість змінювати глобальний масив
  for j in range(0, N):     # Цикл за стовпцями
    n = mas[i][j]           # Беремо елемент масиву
      if n > 0:             # Якщо число додатне
         if n ** 0.5 == int(n ** 0.5):  # Якщо корінь є цілим числом
            L.acquire()             # Захоплення замка
            mas[i][j] = n10         # Заміна елементу масиву на n10
            L.release()             # Вивільнення замка
```

Операція запису в глобальний масив виконується у «монопольному» режимі після захоплення замка. Хоча різні потоки працюють із різними рядками масиву і виникнення стану перегону даних не очікується, автори вважають використання замка правильним підходом, хоча це може трохи уповільнити роботи програми.

Розпаралелювання функції `f2` виконується так само, як і у випадку функції `f1`:

```
t_list = []                          # Підготовка списку потоків
for i in range(0, M):                # Цикл зі створення потоків
    t = threading.Thread(target=f2, args=(i, ))  # Функція f2
    t_list.append(t)
for t in t_list:                     # Цикл із запуску потоків
    t.start()
for t in t_list:                     # Цикл з очікування потоків
    t.join()
```

Об’єднаємо всі розглянуті вище фрагменти коду, пов’язані з багатопотоковою обробкою масиву, в єдину програму. Додатково додамо в програму команди виміру часу роботи подібно тому, як це було зроблено у випадку послідовної версії програми.

Результати експериментальних досліджень

Експерименти здійснено на комп’ютері з процесором i5-13500 та з оперативною пам’яттю типу DDR5 обсягом 64 Гб, який працює під управлінням ОС Windows 11. Використовувалась версія Python 3.12.

Порівняємо час виконання послідовної і паралельної версій програми. Для цього зробимо таке.

1. Визначимо такі розміри масиву, щоб його послідовна обробка тривала 1, 2, 5, 10 та 30 секунд. Під час цього будемо дотримуватись рівності значень M і N. Результати вимірів наведені в табл. 1. За розмірів масиву, вказаних у другому стовпці табл. 1, час обробки масиву за допомогою послідовної версії програми був найбільш близьким до значення, вказаного в першому стовпці таблиці (за усередненими результатами кількох запусків програми).

Таблиця 1. Тривалість послідовного розв’язання задачі

Тривалість (t_1), с	Розмір масиву
1	1230×1230
2	1770×1770
5	2800×2800
10	3950×3950
30	6850×6850

2. Визначимо тривалість виконання паралельної версії програми для розмірів масиву, вказаних у другому стовпці табл. 1. Результати вимірів наведені в табл. 2.

Таблиця 2. Тривалість паралельного розв'язання задачі із використанням механізму потоків

Розмір масиву	Тривалість (t_2), с
1230×1230	0.80
1770×1770	0.96
2800×2800	3.52
3950×3950	6.84
6850×6850	20.5

Порівнюючи значення у першому стовпці табл. 1 зі значеннями у другому стовпці табл. 2, можна бачити, що для усіх розглянутих розмірів масиву тривалість виконання паралельної програми виявилась меншою, ніж тривалість виконання еквівалентної послідовної програми.

3. Визначимо виграш у часі у разі використанні паралельної версії програми замість послідовної. Значення виграшу будемо виражати у відсотках:

$$E = \frac{t_2 - t_1}{t_1} \cdot 100\,\% . \tag{1}$$

Результати оцінювання ефекту від паралелізму (табл. 3) засвідчують, що виграш у часі для масивів великих розмірів сягає приблизно 30%, порівняно з послідовною обробкою масиву.

Таблиця 3. Виграш в часі за рахунок багатопотоковості

Розмір масиву	t_1 , с	t_2 , с	E, %
1230×1230	1	0.80	20
1770×1770	2	0.96	52
2800×2800	5	3.52	29.6
3950×3950	10	6.84	31.6
6850×6850	30	20.5	31.6

Обговорення отриманих результатів

Розглянута в цій роботі задача обробки двовимірного масиву може вважатись процесорозалежною, оскільки передбачає значний обсяг математичних обчислень та не використовує значною мірою операції вводу-виводу даних. Хоча вважається, що багатопотокова реалізація процесорозалежних задач мовою Python не приводить до зменшення часу, порівняно з послідовною реалізацією, отримані результати демонструють зворотне. Отриманий тридцятивідсотковий виграш у часі досягнутий лише за рахунок багатопотокової реалізації без будь-якої оптимізації алгоритму обробки. Водночас додатковий час на створення, обробку та завершення кількох тисяч потоків не завадив загальному виграшу в часі роботи програми.

За проведеними дослідженнями однозначно встановити причину отриманого виграшу не вдалося. Авторами були проведені додаткові дослідження, які полягали у подібному розв'язанні інших задач, пов'язаних з обробкою двовимірних масивів. Дослідження показали, що у випадку деяких задач виграш також сягав приблизно 30%, тоді як для інших

задач виграш був відсутній, і час роботи паралельної версії програми був рівним або більшим за час роботи послідовної версії. Якихось узагальнень та ґрунтовних висновків зробити не вдалося.

Для розглянутої в цій роботі задачі виграш зберігався за різних конфігурацій комп'ютера та версій ОС Windows. Хоча фактичні значення часу роботи програм були більшими (через менш продуктивний процесор), значення виграшу у кілька десятків відсотків зберігалось. Це дає змогу попередньо говорити про незалежність отриманого ефекту від апаратно-програмного забезпечення.

На основі отриманих результатів автори вважають за доцільне провести додаткові дослідження, які полягатимуть у такому:

1) дослідження впливу на виграш співвідношення кількості рядків і стовпців двовимірного масиву;

2) дослідження впливу на виграш загальної завантаженості процесора комп'ютера;

3) дослідження впливу на виграш кількості використовуваних потоків;

4) дослідження ефективності багатопотокової реалізації інших процесорозалежних задач, зокрема операцій множення матриць;

5) проведення зазначених досліджень на комп'ютерах під управлінням різних операційних систем.

Висновки

Експериментально досліджено ефективність розв'язання процесорозалежних обчислювальних задач мовою Python із використанням розпаралелювання на основі механізму потоків. Результати експериментів показали, що у деяких випадках багатопотокова реалізація алгоритму скорочує на 30% тривалість роботи програми, порівняно з послідовною реалізацією. Виграш досягається лише завдяки застосуванню багатопотоковості, без використання додаткових апаратних ресурсів та програмних засобів, тобто «безкоштовно».

Отримані результати ставлять під сумнів загальноприйняте твердження про те, що застосування механізму потоків у мові Python під час виконання процесорозалежних алгоритмів є недоцільним. Якщо розпаралелювання алгоритму на основі процесів за якихось причин є неможливим або недоцільним (наприклад, у випадку великих витрат часу на обмін даними між процесами), програмістам на Python рекомендується завжди розглядати можливість багатопотокової реалізації алгоритмів та оцінювати відповідний ефект у кожному конкретному випадку. Це може привести до отримання виграшу в часі виконання алгоритму без додаткових витрат. Така рекомендація визначає практичну цінність отриманих результатів та перспективність подальших досліджень у цьому напрямі.

Література

1. Качко, О. Г. (2016). *Паралельне програмування*. Харків: Харківський національний університет радіоелектроніки.
2. Мельник, А. О., Яковлєва, І. Д. (2018). *Структурний аналіз і синтез паралельних алгоритмів*. Чернівці: Чернівецький національний університет.

3. Семеренко, В. П. (2018). *Технології паралельних обчислень*. Вінниця: Вінницький національний технічний університет.
4. Burns, B. (2018). *Designing Distributed Systems. Patterns and Paradigms for Scalable, Reliable Services*. Sebastopol: O'Reilly Media.
5. Lutz, M. (2013). *Learning Python*, 5th Edition. Sebastopol: O'Reilly Media.
6. A Guide to Python Multiprocessing and Parallel Programming (2022). https://www.sitepoint.com/python-multiprocessing-parallel-programming/
7. Parallel Processing in Python (2019). https://www.geeksforgeeks.org/parallel-processing-in-python/

Рукопис отримано – 25/06/2024; прийнято до публікації – 03/07/2024.

Research on the efficiency of parallelization of processor-dependent tasks in Python based on the thread mechanism

Roman Babakov, Olexander Barkalov

Abstract

The objective of the paper is to study the effectiveness of using parallel data processing based on threads in the Python language to solve problems that require significant CPU resources. The task of processing a two-dimensional array of large sizes using standard Python tools without using specialized libraries such as Numpy was considered as such a problem. As a result of the research, an unexpected decrease in the execution time of the multi-threaded program by 30% was found in comparison with the sequential (non-parallel) implementation of the problem being solved. This makes it possible to question the well-known statement that multi-threaded solving of processor-dependent tasks in the Python language is not effective, as it does not lead to a decrease in the running time of programs. Previously, it was believed that due to the use of the so-called global lock (GIL), the Python language interpreter, when executing multithreaded programs using Threading module, directs all program threads to one processor core, which excludes the physical parallel execution of threads and under no circumstances allows to reduce time of program execution. A well-known recommendation for obtaining a real gain in time is the use of parallelization based on the process mechanism and the "multiprocessing" module, which allows several physical processor cores to be used. However, this approach requires additional processor resources (cores) and additional time spent on data exchange between processes, which can nullify the effect of multi-core processing. The experiments conducted by the authors proved that the use of a multithreaded approach can also be appropriate in the case of processor-dependent tasks, since the time gain achieved does not require additional resources from the computer's central processor and unnecessary time spent on data exchange between threads.

Keywords: processor-dependent tasks; speed of data processing; parallelization; thread mechanism; the Python language.

References

1. Kachko, O. H. (2016). *Paralelne prohramuvannia*. Kharkiv: Kharkivskyi natsionalnyi universytet radioelektroniky.
2. Melnyk, A. O., Yakovlieva, I. D. (2018). *Strukturnyi analiz i syntez paralelnykh alhorytmiv*. Chernivtsi: Chernivetskyi natsionalnyi universytet.
3. Semerenko, V. P. (2018). *Tekhnolohii paralelnykh obchyslen*. Vinnytsia: Vinnytskyi natsionalnyi tekhnichnyi universytet.
4. Burns, B. (2018). *Designing Distributed Systems. Patterns and Paradigms for Scalable, Reliable Services*. Sebastopol: O'Reilly Media.
5. Lutz, M. (2013). *Learning Python*, 5th Edition. Sebastopol: O'Reilly Media.
6. A Guide to Python Multiprocessing and Parallel Programming (2022). https://www.sitepoint.com/python-multiprocessing-parallel-programming/
7. Parallel Processing in Python (2019). https://www.geeksforgeeks.org/parallel-processing-in-python/

УДК 004.4:004.051:004.31:004.9

Залежність швидкодії програм від інструментальних засобів розробки та синтаксичних конструкцій

Юрій Антонов
доцент, канд. фіз.-мат. наук
ORCID: 0000-0001-9285-2988
iu.antonov@donnu.edu.ua
y.s.antonov@gmail.com

Донецький національний університет імені Василя Стуса

Ключові слова:
компілятор;
інтерпретатор;
швидкодія;
оптимізація роботи програми.

Запропоновано реалізацію автоматизованої інформаційної системи, що дає змогу досліджувати та аналізувати вплив інструментальних засобів розробки та синтаксичних конструкцій на швидкодію роботи програм. У системі застосовано архітектуру трирівневих баз даних. Спеціалізований додаток LST Client використано як клієнт, LST Web Service як сервер додатків, а MySQL як реляційна СУБД. Взаємодія додатка LST Client з LST Web Service відбувається за допомогою Web API з використанням Application Key. Application Key являє собою унікальний ключ, який створюється для кожного пристрою, що бере участь у дослідженні. Для тих випадків, коли у досліджені має взяти участь пристрій, розташований поза межами intranet-мережі, підключення здійснюється аналогічним способом, але через спеціально налаштований SSH-тунель. Для кожного пристрою створюється відповідний користувач з авторизацією за особистим RSA-ключем максимальної довжини. Дослідження проведено для мов програмування C, C++, Fortran, Java, C#, JavaScript, PHP, Python з використанням компіляторів та інтерпретаторів, що належать до x64 архітектури під керуванням Windows 10 для освітніх установ. Результати експериментів засвідчили, що на швидкодію програми впливає версія компілятора чи інтерпретатора, та набір синтаксичних конструкцій, що використовується, наприклад, ++i та i++. Лише компілятори мов програмування C, C++ та Fortran змогли оптимізувати код та не виконувати зайві цикли. Розроблена система дає змогу максимально автоматизувати процес тестування швидкодії програмного коду. Запропонована система може бути використана для оцінювання нових версій мов програмування та програмного забезпечення. Вона також може бути задіяна під час викладання дисциплін, пов’язаних із програмуванням, для звернення уваги студентів на слабкі та сильні місця мов програмування та інструментальних засобів розробки.

DOI: 10.31558/2786-9482.2024.1.3

Вступ

ІТ-галузі притаманні стрімкі трансформації, внаслідок яких стек технологій та інструментів дуже швидко змінюється. Деякі технології або мови програмування з'являються та займають ринок, а потім зникають або продовжують своє існування багато років. Може здаватися, що питання швидкодії (оптимізації) програм вже не актуальне, оскільки можна скористатись хмарними технологіями і отримати більше потужностей для розв'язання тих чи інших задач, однак ці потужності надаються не безкоштовно, і за всі використані ресурси треба буде платити. Оптимізація програм за швидкодією може бути корисною у випадках спортивного програмування, розробки складних високонавантажених систем, що займаються розрахунками, або операційних систем реального часу.

Постановка задачі та аналіз публікацій

Під час створення програмного забезпечення розробникам необхідно не лише власноруч реалізовувати програмний код та Unit-тести, а й використовувати різноманітні компілятори / інтерпретатори, алгоритми, бібліотеки, фреймворки, патерни програмування. І тут одразу виникає питання, як на працездатність програми вплинуть нові версії бібліотеки (фреймворку) та версія компілятора чи інтерпретатора, використання тієї чи іншої синтаксичної конструкції; використання тих чи інших опцій компілятора чи інтерпретатора тощо. До того ж розробникам, архітекторам та DevOps-ам потрібно розуміння, чи варто переходити на нову версію програмного продукту, наприклад, з .NET 5 на .NET 6 чи на .NET8, та з яких саме причин – нові можливості розробки, заощадження деяких системних ресурсів тощо.

Зміни безпосередньо у синтаксисі мови програмування є доволі очевидними і дуже рідко залишаються без уваги, так само, як і поява нової мови програмування. Водночас більш глибокі зміни в інструментальних засобах розробки (компіляторах, інтерпретаторів, стандартних бібліотеках) або принципи їх роботи можуть залишатись без уваги.

Сьогодні проблемам програмування присвячено велику кількість робіт. Так, у роботі [1] проводиться порівняння швидкодії арифметичних операцій над великими цілими числами для мов програмування C++ із бібліотекою GMP та Python зі вбудованою підтримкою такої можливості. Під час експериментальних досліджень автори розглядали такі операції над великими цілими числами: множення двох чисел; піднесення до степеня за модулем; знаходження залишку за модулем; знаходження найбільшого спільного дільника. Авторами було встановлено, що реалізація арифметичних алгоритмів над цілими числами довільної точності на C++/GMP є в середньому у 4 рази швидшою за Python-реалізацію, але програмування на C++ є складнішим. Недоліками роботи можна вважати наявність виключно псевдокоду для кожного з алгоритмів, без детальної реалізації на мовах C++ та Python.

Деякі більш глибокі результати порівняння ефективності мов програмування C, C++, Java, Perl, Python, Rexx, Tcl виконано у роботі [2]. Досліджено лише одну алгоритмічну проблему, але над її розв'язанням працювало 74 різні програмісти. Кожен програміст реалізовував код самостійно. Частина суб'єктів була студентами магістратури комп'ютерних наук. Автори аналізували такі характеристики: час роботи програми; обсяг пам'яті, що

споживає програма; розмір початкового коду; щільність коментарів; структура програми; надійність; обсяг зусиль для написання програми. Результати дослідження показали, що для розв'язання проблеми за допомогою скриптових мов програмування (Perl, Python, Rexx, Tcl) потрібно вдвічі менше часу, ніж для мов, що компілюються (C, C++, Java). Щодо часу роботи та споживання пам'яті скриптові мови програмування виявились кращими за Java, та не набагато гіршими ніж, C або C++.

У роботі [3] автори порівнюють швидкодію роботи Java-програми на мікрокомп'ютері Raspberry Pi та класичному настільному комп'ютері. Для проведення дослідження автори створили 2 програми, а саме програму для швидкого алгоритму пошуку простих чисел до заданого цілого числа N (решето Аткіна) та програму для роботи з числами з комою, що плаває. Кількість запусків програми підібрано так, щоб стандартне відхилення не перевищувало 5% від середнього часу роботи програми. У роботі не наведено абсолютних значень швидкості, а лише відносні – швидкість роботи програми на стандартному персональному комп'ютері прийнято за одиницю. В результаті встановлено, що використання Oracle JDK дає змогу виконувати програми у 10–20 разів швидше, порівняно з OpenJDK.

У роботі [4] розглядається проблема обрання мови програмування для навчання або розробки програмного забезпечення за критерієм швидкодії програм. Численні дослідження проводились для таких мов програмування: C, C++, Fortran, Java, C#, PHP, Python, JavaScript. До недоліків роботи можна віднести наявність програмного коду лише для мови програмування C та відсутність опису платформи, на якій проводились дослідження.

У роботі [5] за допомогою мови програмування Java створено вебдодаток та протестована його швидкодія для різних типів серверів у хмарних сервісах AWS EC2, з подальшим визначенням оптимальних варіантів для розгортання цього вебдодатка. Оптимальним, на думу авторів, є таке рішення, яке дає максимальне співвідношення продуктивності та стабільності до ціни утримання такої системи. Для запуску тестів та візуалізації отриманих результатів використовувався інструмент Gatling.

Дослідження енергоефективності як нової метрики для оцінювання ефективності програмного забезпечення здійснено у роботах [6, 7]. Зокрема, в [6] створено рейтинг енергоефективності 27 мов програмування на основі аналізу впливу швидкості виконання програми та використання оперативної пам'яті на рівень спожитої енергії. Лідерами рейтингу є C, Pascal та Ada.

Отже, можна побачити, що дослідження впливу інструментальних засобів розробки та синтаксичних конструкцій на швидкодію програм проведено недостатньою мірою. Також не розкрита повністю проблема використання інформаційних систем для оцінки швидкодії різноманітних програм. ***Метою роботи*** є виявлення впливу інструментальних засобів розробки та синтаксичних конструкцій на швидкодію роботи програм для різних операційних систем, пристроїв, інтерпретаторів та компіляторів за допомогою автоматизованої інформаційної системи.

Інформаційна система для експериментальних досліджень швидкодії

Інформаційна система має відповідати таким функціональним вимогам:

- функціонування системи та запуск тестів має відбуватися для різних мов програмування, компіляторів та інтерпретаторів;
- запуск тестів має здійснюватися на різних пристроях зі збереженням інформації про операційну систему, процесор та обсяг оперативної пам'яті;
- система має давати можливість задавати різні параметри для компіляторів (інтерпретаторів) та середовища виконання;
- для чистоти експерименту буде вимірюватись час роботи виключно якогось алгоритму, а не програми загалом;
- запуск кожної програми відбуватиметься задану кількість разів поспіль з обов'язковим збереженням результатів у базі даних, і для подальшого аналізу враховуватиметься лише найменше значення;
- файли з програмним кодом зберігаються на рівні файлової системи, а не бази даних;
- звіти за тестом генеруються з урахуванням пристрою, на якому він виконувався.

Для реалізації системи візьмемо за основу архітектуру трирівневих баз даних. Клієнтом буде виступати спеціалізований додаток LST Client, на другому рівні буде працювати LST Web Service поверх вебсервера Apache, а на третьому рівні буде розташована реляційна СУБД MySQL (рис. 1).

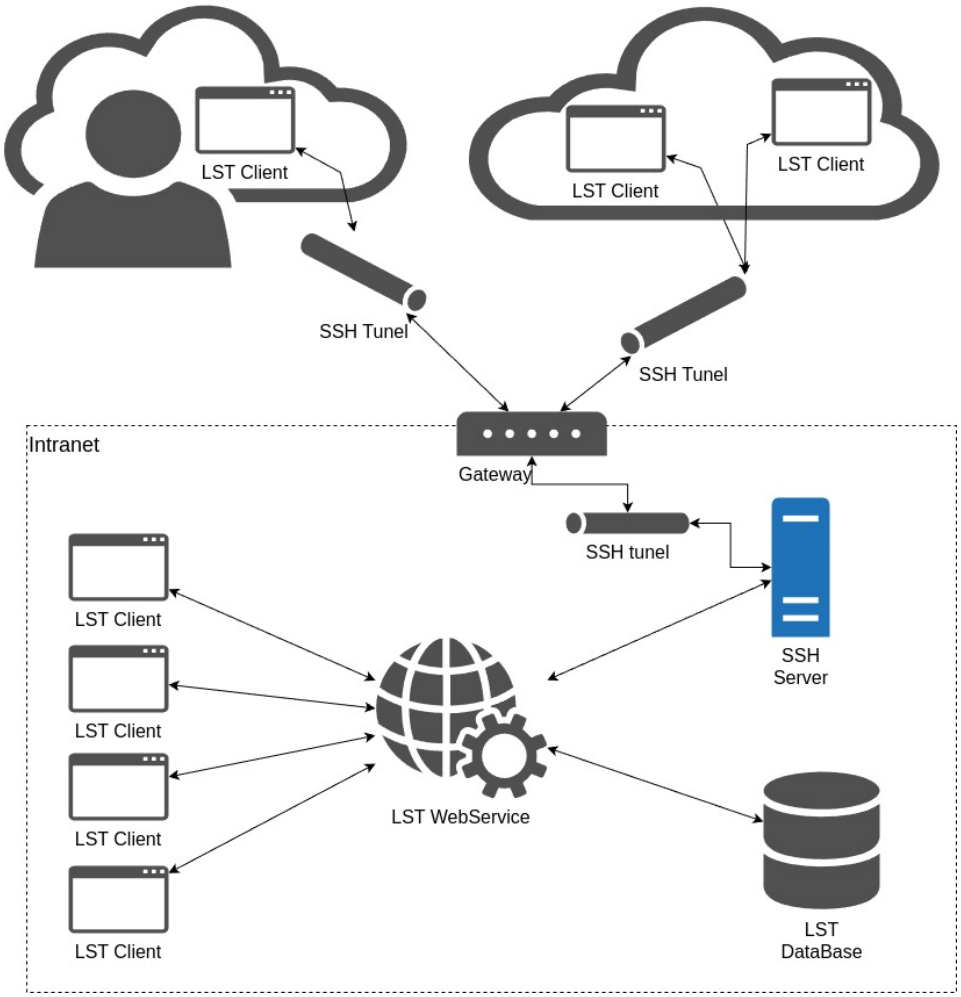

Рисунок 1. Концептуальна схема взаємодії компонентів системи LST

Взаємодія додатка LST Client з LST Web Service відбувається за допомогою Web API з використанням Application Key. Application Key являє собою унікальний ключ, який створюється для кожного пристрою, що бере участь у дослідженні. Для тих випадків, коли у досліджені має взяти участь пристрій, розташований поза межами intranet-мережі, підключення здійснюється аналогічним способом, але через спеціально налаштований SSH-тунель. Для кожного пристрою створюється відповідний користувач з авторизацією за особистим RSA-ключем максимальної довжини. Реляційна база даних інформаційної системи, за винятком компонентів авторизації та безпеки, наведена на рис. 2.

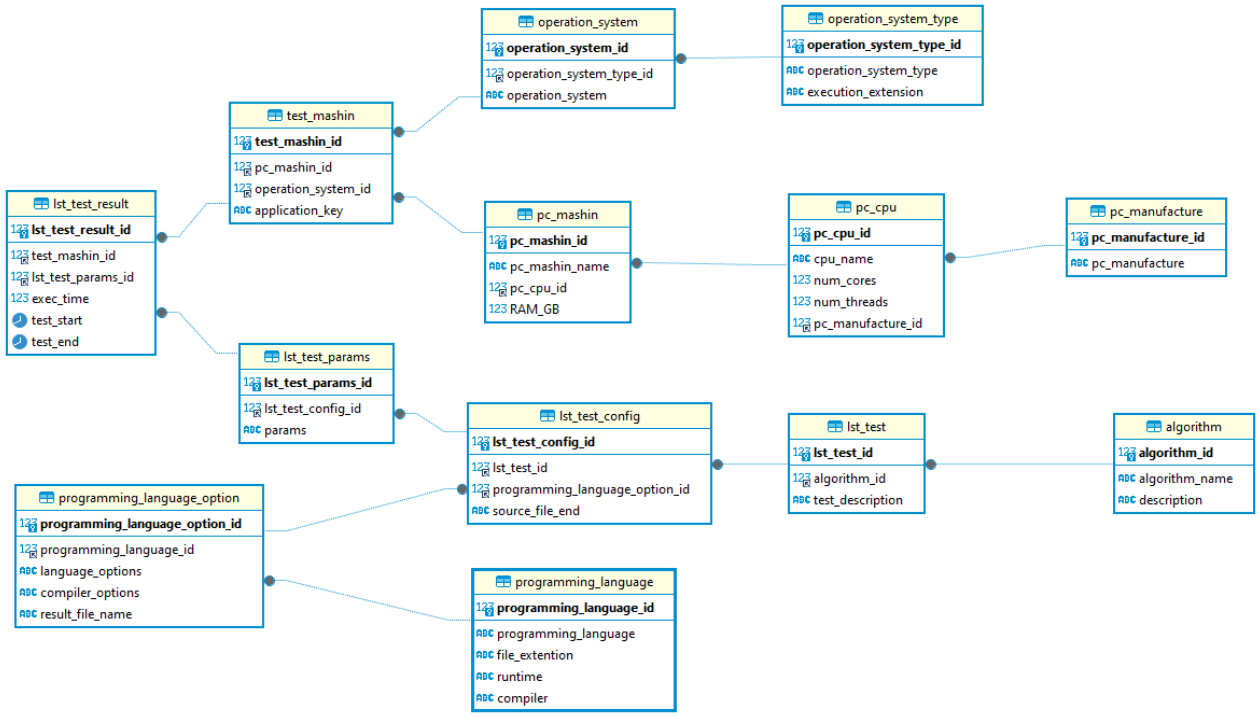

Рисунок 2. Фрагмент бази даних LST

Для експериментального дослідження впливу інструментальних засобів розробки та синтаксичних конструкцій на швидкодію роботи програм пропонується такий тестовий алгоритм:

```
for i = 1…M
   for j = 1…M
      for k = 1…M
         for t = 1…M
            s=i+j+k+t;
```

На перший погляд, мета такого коду – обчислення суми та її накопичення у циклі. Однак, якщо подивитись більш детально на останній рядок, ми побачимо, що накопичення не відбувається, і в результаті у змінній `s` буде зберігатись число, що дорівнює $4M$. Незважаючи на простоту запропонованого алгоритму, він дає змогу розкрити оптимізаційні можливості компіляторів та інтерпретаторів і зрозуміти, наскільки глибоко та якісно відбувається цей процес. За його допомогою можна і відслідковувати, чи відбуваються зміни з виходом нових версій компіляторів або інтерпретаторів.

Експерименти проведемо для компіляторів та інтерпретаторів мов програмування з табл. 1. Усі обчислення проведемо для хостів з табл. 2 для компіляторів / інтерпретаторів, що належать до x64 архітектури.

Таблиця 1. Специфікація інструментальних засобів

Мова	Інструмент	Псевдонім
C	gcc	C
C++	g++	C++
Fortran	gfortran	Fortran
Java	javac	Java
C#	csc	C# .NET
C#	dotnet core3	C# Core
JavaScript	Node JS	JavaScript
PHP	php5	PHP 5
PHP	php7	PHP 7
Python	pypy	Python pypy
Python	python3	Python 3

Таблиця 2. Параметри хостів

Machine ID	CPU	RAM	OS
4	Core i3 6100	8	Windows 10 для освітніх установ x64
6	Core i5 10500	32	Windows 10 для освітніх установ x64

Тестовий алгоритм реалізуємо мовою програмування C так:

```
for(int i = 1; i <= M; i++)
   for(int j = 1; j <= M; j++)
      for(int k = 1; k <= M; k++)
         for(int t = 1; t <= M; t++)
            s=i+j+k+t;
```

Для аналізу роботи компілятора створимо 4 різні програми – без оптимізації та з використанням трьох стандартних опцій -O, -O2 та -O3. Додатково створимо ще одну версію коду, в якій замість `i++` використовувався варіант `++i`. У табл. 3 наведено результати досліджень для різних значень M, з яких видно, що неоптимізований код у 5–6 разів повільніший, ніж код, створений за допомогою опції -O. Для опцій -O2 та -O3 час виконання програм дорівнює нулю, оскільки компілятор зміг визначити, що змінна `s` змінюється у циклі, але остаточне значення обчислюється на останній ітерації циклу. Компілятор обчислює це значення ($4M$) не вставляючи цикли в код програми. Для мови програмування C++ код алгоритму буде виглядати так само, як і для мови C, а результати та закономірності майже ідентичні результатам для мови C (табл. 4). Тривалість виміряно у секундах.

Таблиця 3. Тривалість виконання тестових програм на мові C

Тест	Machine ID	*M*					
		50	100	150	200	250	300
C(i++)	4	0	0.156	0.828	2.594	6.344	13.095
C(++i)	4	0	0.156	0.828	2.594	6.329	13.033
C(-O; i++)	4	0	0.031	0.156	0.484	1.156	2.36
C(-O; ++i)	4	0	0.031	0.156	0.484	1.141	2.36
C(-O2, -O3; i++)	4	0	0	0	0	0	0
C(i++)	6	0.009	0.142	0.702	2.199	5.335	10.998
C(++i)	6	0.01	0.145	0.708	2.199	5.324	11.017
C(-O; i++)	6	0.002	0.028	0.137	0.408	0.975	1.994
C(-O; ++i)	6	0.002	0.028	0.132	0.408	0.975	1.995
C(-O2, -O3; i++)	6	0	0	0	0	0	0

Таблиця 4. Тривалість виконання тестових програм на мові C++

Тест	Machine ID	*M*					
		50	100	150	200	250	300
C++(i++)	4	0	0.156	0.828	2.61	6.344	13.142
C++(++i)	4	0	0.156	0.828	2.594	6.329	13.095
C++(-O; i++)	4	0	0.031	0.156	0.484	1.172	2.391
C++(-O; ++i)	4	0	0.031	0.156	0.484	1.172	2.407
C++(-O2,-O3; i++)	4	0	0	0	0	0	0
C++(i++)	6	0.009	0.142	0.702	2.199	5.325	10.998
C++(++i)	6	0.009	0.142	0.703	2.199	5.316	11
C++(-O; i++)	6	0.002	0.028	0.136	0.416	0.991	2.023
C++(-O; ++i)	6	0.002	0.029	0.137	0.416	0.992	2.02
C++(-O2,-O3; i++)	6	0	0	0	0	0	0

Код тестової програми на мові Fortran виглядає так:

```
do i=1,M
   do j=1,M
      do k=1,M
         do l=1,M
            s=i+j+k+l
         end do
      end do
   end do
end do
```

Опції компіляції -O2 та -O3 для Fortran забезпечують нульовий час виконання програми. Неоптимізований код виявився повільніший у 6.5–7.2 разів, ніж код, створений за допомогою опції -O (табл. 5).

Таблиця 5. Тривалість виконання тестових програм на мові Fortran

Тест	Machine ID	*M*					
		50	100	150	200	250	300
Fortran	4	0	0.203	1.063	3.406	8.375	17.391
Fortran(-O)	4	0	0.031	0.153	0.484	1.172	2.406
Fortran(-O2, -O3)	4	0	0	0	0	0	0
Fortran	6	0	0.172	0.891	2.844	7.031	14.625
Fortran(-O)	6	0	0.017	0.125	0.406	0.984	2.031
Fortran(-O2, -O3)	6	0	0	0	0	0	0

Для мов Java та C# програмний код для реалізації алгоритму буде виглядати так само, як і для мови C. З даних, наведених у табл. 6 та 7, видно, що під час використання `i++` та `++i` різниця між часом виконання програми майже не помітна; вона знаходиться у межах похибки. Позначка Aggres означає, що програма Java запускалася з параметром `-XX: +AggressiveOpts`. Використання оптимізації для мови C# дає змогу збільшити швидкість роботи програми вдвічі за компілятора csc.exe та у 6 разів за інструмента dotnet run. За збільшення кількості ітерацій результати для Java, оптимізованого C# .NET та C# Core стають майже однаковими.

Таблиця 6. Тривалість виконання тестових програм на мові Java

Тест	Machine ID	*M*					
		50	100	150	200	250	300
Java(i++)	4	0.005	0.046	0.172	0.5	1.187	2.406
Java(++i)	4	0.005	0.031	0.172	0.5	1.172	2.391
Java(Aggres; i++)	4	0.006	0.047	0.171	0.5	1.187	2.406
Java(Aggres; ++i)	4	0.005	0.031	0.172	0.5	1.172	2.391
Java(i++)	6	0.004	0.033	0.144	0.428	1.009	2.042
Java(++i)	6	0.004	0.033	0.144	0.427	1.006	2.062
Java(Aggres; i++)	6	0.004	0.032	0.145	0.43	1.012	2.052
Java(Aggres; ++i)	6	0.005	0.032	0.144	0.428	1.01	2.069

Таблиця 7. Тривалість виконання тестових програм на мові C#

Тест	Machine ID	*M*					
		50	100	150	200	250	300
C#.NET(i++)	4	0.062	0.122	0.369	1.022	2.383	4.833
C#.NET(++i)	4	0.061	0.122	0.368	1.021	2.379	4.82
C#.NET(/o; i++)	4	0.06	0.092	0.22	0.555	1.24	2.469
C#.NET(/o; ++i)	4	0.059	0.091	0.221	0.553	1.236	2.464
C#Core(++i, i++)	4	0.004	0.038	0.168	0.515	1.22	2.49
C#.NET(i++)	6	0.006	0.057	0.267	0.819	1.962	4.022
C#.NET(++i)	6	0.006	0.057	0.267	0.819	1.957	4.022
C#.NET(/o; i++)	6	0.004	0.032	0.143	0.426	1.008	2.049
C#.NET(/o; ++i)	6	0.004	0.032	0.142	0.426	1.008	2.049
C#.NET8(i++)	6	0.012	0.16	0.774	2.407	5.83	12.02
C#.NET8(opt i++)	6	0.01	0.037	0.146	0.425	1.002	2.034
C#.NET6(i++)	6	0.012	0.158	0.77	2.401	5.833	12.011
C#.NET6(opt i++)	6	0.004	0.032	0.14	0.422	0.999	2.034
C#.NET3.1(i++)	6	0.013	0.159	0.773	2.405	5.837	12.029
C#.NET3.1(opt i++)	6	0.004	0.032	0.141	0.423	1	2.034

Для мови програмування JavaScript програмний код алгоритму буде виглядати так само, як і для мови C. З табл. 8 видно, що операція `++i` виконується швидше, ніж `i++`, на 1.2–4.8%, залежно від кількості ітерацій для хоста №4 зі старою версією Node.JS, тоді як для хоста №6 з Node.JS 22 ця різниця є непомітною.

Таблиця 8. Тривалість виконання тестових програм на мові JavaScript

Тест	Machine ID	*M*					
		50	100	150	200	250	300
JavaScript(i++)	4	0.027	0.398	1.991	6.258	15.141	31.047
JavaScript(++i)	4	0.026	0.389	1.925	6	14.589	29.987
JavaScript(i++)	6	0.008	0.059	0.264	0.802	1.936	3.924
JavaScript(++i)	6	0.007	0.0589	0.264	0.798	1.925	3.887

Для мови програмування PHP програмний код алгоритму, що розглядається, буде таким самим, як і для мови програмування C. Використання у мові програмування PHP конструкції `++i` робить програму на 6–10% швидшою, порівняно з варіантом, де використовується конструкція `i++` (табл. 9). Для хоста №4 інтерпретатор PHP 7 на 28% швидший за інтерпретатор PHP 5 як для `++i`, так і для `i++`. Для хоста №6 за конструкції `i++` маємо, що PHP 7 на 16.5–18% швидший за інтерпретатор PHP 8, та на 84–85% швидший

за PHP 5. Для синтаксичної конструкції `++i` PHP 7 на 31% швидший за інтерпретатор PHP 8 та на 91% швидший за PHP 5 (рис. 4).

Таблиця 9. Тривалість виконання тестових програм на мові PHP

Тест	Machine ID	M					
		50	100	150	200	250	300
PHP5(i++)	4	0.241	3.818	19.206	60.75	147.946	306.009
PHP5(++i)	4	0.219	3.494	17.487	55.205	134.789	279.996
PHP7(i++)	4	0.186	2.953	14.95	47.2113	115.179	238.54
PHP7(++i)	4	0.17	2.716	13.637	43.02	105.211	218.85
PHP5(i++)	6	0.227	3.555	17.979	56.485	137.732	284.984
PHP5(++i)	6	0.212	3.311	16.736	52.611	128.59	266.465
PHP7(i++)	6	0.122	1.92	9.689	30.593	74.543	154.614
PHP7(++i)	6	0.112	1.736	8.737	27.632	67.462	139.636
PHP8(i++)	6	0.146	2.268	11.287	36.09	88.067	182.346
PHP8(++i)	6	0.146	2.269	11.418	36.076	88.188	182.612

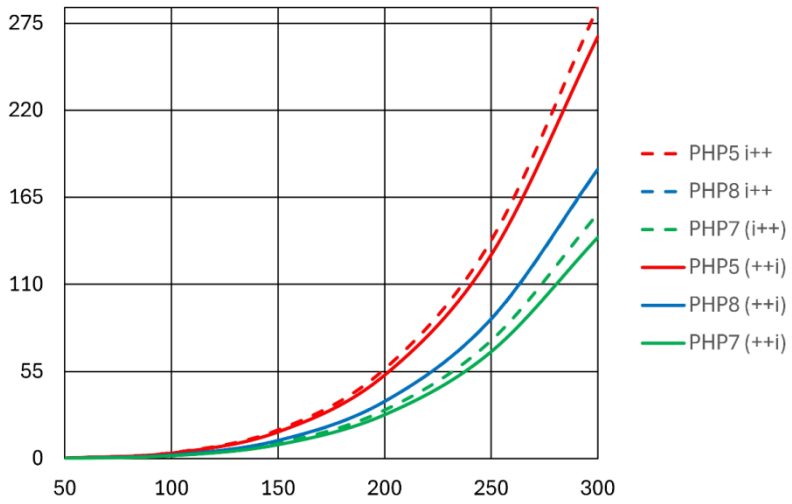

Рисунок 4. Тривалість виконання тестової програми на PHP для хоста №6

Базовий код на Python виглядає так:

```
startTime=time.time()
for i in range(1,M):
for j in range(1,M):
for k in range(1,M):
for t in range(1,M):
S=i+j+k+t
endTime=time.time()
```

Для перевірки гіпотези, що код на Python виконується швидше, якщо він розміщений в окремій функції, створимо ще таку програму:

```
def test(M):
S=0
myr=range(1,M)
for i in myr:
   for j in myr:
      for k in myr:
         for t in myr:
            S=i+j+k+t
return S
startTime=time.time()
S=test(M)
endTime=time.time()
```

Для хоста №4 використання коду у вигляді функції підвищує швидкодію на 56–62% за стандартного інтерпретатора Python, та на 53–73% для інтерпретатора pypy (табл. 10). Для хоста №6 використання функції підвищує швидкодію на 99–119% за стандартного інтерпретатора Python та на 87–93% за інтерпретатора pypy. Використання pypy пришвидшує програми у 42–63 рази.

Таблиця 10. Тривалість виконання тестових програм на мові Python

Тест	Machine ID	*M*					
		50	100	150	200	250	300
Python 3	4	0.688	10.767	57.756	197.302	504.366	1114.116
Python 3(proc)	4	0.464	7.186	35.55	121.888	319.627	713.882
Python pypy	4	0.015	0.265	1.266	3.938	9.485	19.517
Python pypy(proc)	4	0.016	0.141	0.719	2.432	6.198	11.266
Python 3	6	0.488	7.662	42.353	149.586	370.926	818.553
Python 3(proc)	6	0.19	3.131	19.31	70.315	185.049	410.683
Python pypy	6	0.018	0.177	0.85	2.635	6.4038	13.224
Python pypy(proc)	6	0.01	0.097	0.453	1.39	3.333	6.85

Висновки

Проведені дослідження показали, що:

- усі мови програмування та інструменти розробки мають свої тонкощі та особливості, які мають бути досліджені;
- у разі декількох компіляторів / інтерпретаторів їх поведінка, параметри та якість створених програм можуть суттєво відрізнятися;
- з випуском нових версій компіляторів / інтерпретаторів, ситуація може суттєво змінюватися;
- операційна система та апаратно-програмне забезпечення впливають на результати тестів;
- синтаксичні конструкції `++i` та `i++` можуть по-різному оброблюватися компіляторами / інтерпретаторами та впливати на роботу програми;
- лише компілятори мов програмування C, C++ та Fortran змогли оптимізувати код та не

створювати 4 зайві цикли. Для інших інструментальних засобів ситуація залишається незмінною вже протягом 7 років.

Розроблена інформаційна система дає змогу максимально автоматизувати процес тестування швидкодії програмного коду з його подальшим аналізом. Система може бути використана для таких завдань:

- виявлення кращих практик та слабких місць обраного стеку технологій під час розробки програмного забезпечення;
- оцінювання нових версій мов програмування та програмного забезпечення на предмет доцільності переходу на них;
- акцентування уваги студентів на слабких та сильних місцях мов програмування та інструментальних засобів розробки під час викладання дисциплін із програмуванням.

Проведені тести не є вичерпними та не відображають повної картини, але достатньою мірою ілюструють мінливість результатів залежно від способу написання коду, параметрів компілятора / інтерпретатора та середовища виконання. Розглянута у роботі система, після мінімальних змін також може бути задіяна у автоматизованих системах контролю знань з метою перевірки правильності написання програм.

Література

1. Новокшонов, А. К. (2016). Аналіз ефективності реалізації арифметичних алгоритмів на мовах програмування C++ та Python. *Проблеми програмування*, (2–3), 26–31. https://doi.org/10.15407/pp2016.02-03.026
2. Prechelt, L. (2000). An empirical comparison of C, C++, Java, Perl, Python, Rexx and Tcl. *IEEE Computer*, *33*(10), 23–29.
3. Дідух, О. І., Тищенко, В. В. (2015). Порівняння швидкодії Java на мікрокомп'ютері Raspberry Pi. *Вісник Національного технічного університету України «Київський політехнічний інститут». Серія: Радіотехніка. Радіоапаратобудування*, *60*, 107–113.
4. Антонов, Ю. С., Дзігора, К. Р. (2017). Проблема обрання мови програмування, як інструменту для навчання та розробки. *Матеріали наукової конференції професорсько-викладацького складу, наукових працівників і здобувачів наукового ступеня за підсумками науково-дослідної роботи за період 2015–2016 рр.* (с. 23–25). Донецький національний університет імені Василя Стуса.
5. Сігунов, О., Демків, Л. (2022). Дослідження швидкодії обробки паралельних запитів хмарними сервісами AWS. *Електроніка та інформаційні технології*, *20*, 30–41. http://dx.doi.org/10.30970/eli.20.4
6. Pereira, R., Couto, M., Ribeiro, F., Rua, R., Cunha, J., Fernandes, J. P., & Saraiva, J. (2021). Ranking programming languages by energy efficiency. *Science of Computer Programming*, *205*. https://doi.org/10.1016/j.scico.2021.102609
7. Gordillo, A., Calero, C., Moraga, M. Á. et al. (2024). Programming languages ranking based on energy measurements. *Software Quality Journal*. https://doi.org/10.1007/s11219-024-09690-4

Рукопис отримано – 06/08/2024; прийнято до публікації – 17/08/2024.

Dependence of program speed on development tools and syntactic constructions

Yuriy Antonov

Abstract

This article proposes the implementation of an automated information system that allows you to research and analyze the influence of development tools and syntactic structures on the speed of the programs. A fragment of the database scheme of this system and a conceptual scheme of the interaction of components are given. The requirements for such a system are formulated. During the implementation of the system, the architecture of three-level databases was used. The specialized application LST Client was used as the client, LST WebService and the MySQL relational DBMS were used as the application server. The LST Client application interacts with the LST Web Service using the Web API and the Application Key. The Application Key is a unique key that is created for each device participating in the study. For those cases when the device to participate in the research is located outside the intranet network, the connection is made in a similar way, but through a specially configured SSH tunnel. For each device, a corresponding user is created with authorization based on a personal RSA key of the maximum length. Numerical studies were performed for programming languages C, C++, Fortran, Java, C#, JavaScript, PHP, Python using compilers/interpreters belonging to the x64 architecture running Windows 10 for educational institutions. The conducted research made it possible to establish that the speed of the program is affected by the version of the compiler / interpreter and the set of syntax constructions used, for example ++i and i++. Only C, C++, and Fortran compilers were able to optimize the code and avoid unnecessary loops. The developed system makes it possible to automate the process of testing the speed of the software code as much as possible. This system can be used to evaluate new versions of programming languages and software. The obtained results make it possible to evaluate the expediency of switching to new versions of the software or to focus students' attention on the weak and strong points of a certain programming language and development tools when teaching disciplines related to programming.

Keywords: Compiler; interpreter; program operation speed; optimization of the program.

References

1. Novokshonov, A. K. (2016). Analiz efektyvnosti realizatsii aryfmetychnykh alhorytmiv na movakh prohramuvannia C++ ta Python. *Problemy prohramuvannia*, (2–3). 26–31. https://doi.org/10.15407/pp2016.02-03.026
2. Prechelt, L. (2000). An empirical comparison of C, C++, Java, Perl, Python, Rexx and Tcl. *IEEE Computer*, *33*(10), 23–29.
3. Didukh, O. I., Tyshchenko, V. V. (2015). Porivniannia shvydkodii Java na mikrokompiuteri Raspberry Pi. *Visnyk Natsionalnoho tekhnichnoho universytetu Ukrainy «Kyivskyi politekhnichnyi instytut». Seriia: Radiotekhnika. Radioaparatobuduvannia, 60*, 107–113.
4. Antonov, Yu. S., Dzihora K. R. (2017). Problema obrannia movy prohramuvannia, yak instrumentu dlia navchannia ta rozrobky. *Materialy naukovoi konferentsii profesorsko-vykladatskoho skladu, naukovykh pratsivnykiv i zdobuvachiv naukovoho stupenia za pidsumkamy naukovo-doslidnoi roboty za period 2015–2016 rr.* (s. 23–25). Donetskyi natsionalnyi universytet imeni Vasylia Stusa.
5. Sihunov, O., Demkiv, L. (2022). Doslidzhennia shvydkodii obrobky paralelnykh zapytiv khmarnymy servisamy AWS. *Elektronika ta informatsiini tekhnolohii*. *20*. 30–41. http://dx.doi.org/10.30970/eli.20.4
6. Pereira, R., Couto, M., Ribeiro, F., Rua, R., Cunha, J., Fernandes, J. P., & Saraiva, J. (2021). Ranking programming languages by energy efficiency. *Science of Computer Programming*, *205*. https://doi.org/10.1016/j.scico.2021.102609
7. Gordillo, A., Calero, C., Moraga, M. Á. et al. (2024). Programming languages ranking based on energy measurements. *Software Quality Journal.* https://doi.org/10.1007/s11219-024-09690-4

УДК 001.02.3

Експрес-підбір опонентів для разових рад із захисту PhD-дисертацій

Сергій Штовба
професор, д-р техн. наук
ORCID: 0000-0003-1302-4899
s.shtovba@donnu.edu.ua

Донецький національний університет імені Василя Стуса

Микола Петричко
ORCID: 0000-0001-6836-7843
mpetrychko@vntu.edu.ua

Вінницький національний технічний університет

Ключові слова:
задача про призначення рецензентів;
експрес-підбір;
обробка природної мови;
категоризація;
дискретна оптимізація;
аналіз даних;
Dimensions.

Сьогодні ради із захисту PhD-дисертацій формують у ручному режимі. Це обумовлює як корупційні ризики, так і значні витрати часу на пошук та аналіз кандидатів з великими шансами пропустити кваліфікованих опонентів. Тому виникає зацікавленість у автоматизації формування разових рад для усунення зазначених ризиків впливу людського фактора. Стаття фокусується на експрес-підборі рад, коли потрібно сильно звузити великий список кандидатів. Подальший короткий список можна аналізувати або вручну, або передавати на процедуру тонкого підбору, яка є ресурсно-витратною і вимагає значно більшого об'єму початкової інформації. Пропонується метод призначення команди рецензентів за їх відповідністю тематиці дисертації, який, на відміну від ізольованого підбору кандидатів, враховує здатність саме колективу рецензентів спільно оцінити роботу за всіма аспектами її тематики. Метод є збалансованим за критеріями якості підбору і витратами ресурсів на пошук членів ради. Метод включає три етапи. На першому етапі здійснюється категоризація дисертації та потенційних членів ради шляхом представлення їх тематик векторами у просторі наукових спеціальностей з ANZSRC-2020. На другому етапі розраховується рівень відповідності кандидатів тематиці дисертації з урахуванням спорідненості наукових спеціальностей ANZSRC-2020. На третьому етапі підбирається склад ради, яка відповідає тематиці дисертації з максимально можливим ступенем. Для реалізації третього етапу запропоновано кілька алгоритмів оптимізації. Тестування алгоритмів на сформованому датасеті із 67 PhD-дисертацій показало, що найкращий баланс за критеріями якості підбору й витрат ресурсів на пошук колективу забезпечують жадібний алгоритм без елітизму та повний перебір на прорідженій множині кандидатів. Внаслідок оптимізації вдалося покращити склад разових рад у середньому на 13–34% залежно від типу використаного алгоритму.

DOI: 10.31558/2786-9482.2024.1.4

Вступ

В Україні PhD-дисертації захищають у разових радах. Разова рада складається із 5 науковців, які мають бути фахівцями з тематики дисертації. Голова ради та 1 або 2 рецензенти представляють заклад, у якому утворюється разова рада, а 2 чи 3 опоненти

запрошують з інших установ. Членів разової ради добирають вручну і затверджують її склад рішенням вченої ради закладу.

Ручне формування разової ради має кілька недоліків. По-перше, це корупційні ризики, коли раду формують виключно з дружніх осіб, які апріорі надають лише схвальні відгуки незалежно від результатів дисертації. В успішному захисті дисертації зацікавлені як особи, які підбирають членів разової ради, так і заклад, який її затверджує. За даними з інформаційної системи НАЗЯВО на поточний момент відбулося 5 324 успішні захисти дисертацій; випадків відмов присудження наукового ступеня немає. Інколи успішно захищають не лише відверто слабкі дисертації, але навіть явно псевдонаукові. Бували навіть випадки, коли здобувачі ледь-ледь розмовляли українською, але це не вплинуло на ухвалення позитивних рішень. По-друге, витрачається багато часу на ручний пошук та аналіз кандидатів у члени ради. Щомісяця в Україні формується приблизно 300 разових рад. Якщо припустити, що на пошук членів однієї ради в середньому витрачається 10 людино-годин, тоді за місяць набігає 3 000 людино-годин, що еквівалентно 18 ставкам. По-третє, сформований склад ради може не повністю відповідати тематиці дисертації через те, що когось із хороших кандидатів упустили під час ручного пошуку. Тому виникає зацікавленість у автоматизації формування разових рад для усунення зазначених ризиків впливу людського фактора.

Формування разових рад із захисту PhD-дисертації є однією із задач призначення рецензентів наукових творів, яку в англомовній літературі називають *Reviewer Assignment Problem* або *Paper-Reviewer Assignment*. Для специфічних задач призначення рецензентів використовують також терміни *Committee Review Assignment* та *Conference Paper Assignment Problem* [1]. Тут під призначенням рецензентів розуміється підбір усіх осіб, які оцінюють наукові твори. За термінологією цих задач твір, який оцінюється, називається заявкою. В контексті експертизи PhD-дисертації рецензентами є усі члени разової ради – і голова ради, і опоненти, і номінальні рецензенти.

Задача призначення рецензентів складається з трьох етапів [2, 3]: 1) пошук рецензентів і вибір методу представлення даних про рецензентів та заявки; 2) підрахунок схожості між заявкою та рецензентами; 3) розподіл заявок за рецензентами для максимізації агрегованої схожості за всіма призначеннями за деяких обмежень. Типовими обмеженнями є збалансованість навантажень рецензентів, врахування їх вподобань та запобігання конфлікту інтересів. У цій роботі вважається, що список потенційних рецензентів наявний.

Автоматичний підбір рецензентів передбачає доступність деякої початкової інформації про рецензентів та заявки. Структуровану сукупність такої інформації називають профілем рецензента та профілем заявки. Доволі часто для побудови профіля рецензента використовується така інформація про його статті: назва, анотація, ключові слова, повний текст, список посилань та список цитувань [4]. Профіль також може містити і особисті дані рецензента. Для створення профіля заявки найчастіше використовують анотацію, повний текст, ключові слова, назву роботи та галузь дослідження [4].

Побудова профілів заявки та рецензентів здійснюється за допомогою різноманітних методів обробки природної мови на основі мішка слів [4, 5, 6], прихованого семантичного аналізу [7, 8], тематичного моделювання [9, 10, 11], статичних мовних моделей з глибоким навчанням [11, 12, 13, 14, 15, 16] та контекстуальних моделей з глибоким навчанням [17, 18,

19, 20, 21]. Підходи до вирішення задачі автоматичного призначення рецензентів у більшості випадків потребують доволі великого обсягу початкової інформації про публікації рецензентів, їх взаємодію з іншими науковцями та аналогічну інформацію про авторів заявок. Оброблення такої інформації є витратним і не буде доцільним, якщо під кожну разову раду детально аналізувати тисячі кандидатів.

Ми фокусуємося на задачі експрес-підбору рецензентів, коли потрібно сильно скоротити довгий початковий список кандидатів. Подальший короткий список можна аналізувати вручну, або активувати процедуру тонкого підбору, яка є ресурсно-витратною і вимагає значно більшого об'єму початкової інформації, ніж це потрібно для експрес-підбору. Під час експрес-підбору враховується лише семантична схожість між заявками та рецензентами – відбувається підрахунок індексів схожості і далі призначається необхідна кількість рецензентів так, щоб забезпечити максимальну відповідність колективу рецензентів заявці за деяким критерієм. Розробка ефективного алгоритму експрес-формування разової ради для PhD-дисертацій, який є збалансованим за критеріями якості підбору та витрат ресурсів на формування колективу, і є ***метою*** цього дослідження.

Представлення даних про заявку та рецензентів

На першому етапі призначення рецензентів необхідно обрати початкові дані для прийняття рішення, а також метод їх представлення у векторній формі. У випадку заявки використовується список її ключових слів, а у випадку рецензента – список ключових слів, який отримується з доступних даних. У загальному випадку цей список ключових слів може бути як зі свіжих публікацій кандидата, так із його CV чи із профілю з деякого реєстру науковців. За другого випадку ключові слова або дослідницькі інтереси кандидат формує на власний розсуд, тобто вони представлені у довільній формі без прив'язки до будь-якого рубрикатора чи класифікатора. За поточної практики захисту вітчизняних PhD-дисертацій тематика члена спецради описується ключовими словами кількох його свіжих статей.

Початкові дані зазвичай обробляють з використанням статистичних моделей, тематичних моделей та моделей ембедингу. Деякі з них аналізують частоту появи слів у тексті, інші формують вектори представлення на основі співпояви слів. Зазвичай результуючі векторні представлення складно інтерпретувати. До того ж для отримання таких представлень необхідна велика кількість даних. Ми пропонуємо використовувати підхід з [22], за яким сукупність ключових слів категоризується і відображається як вектор у просторі наукових спеціальностей з Австралійсько-Новозеландської системи класифікації наук ANZSRC-2020. ANZSRC-2020 включає в себе 171 спеціальність із 22 галузі. Отже, кінцеве представлення профілів заявки та рецензента виглядає як розподіл над 171 науковими спеціальностями з ANZSRC-2020.

Щоб здійснити категоризацію, необхідно мати корпус розмічених статей, які приписані до однієї чи кількох наукових спеціальностей, та модель машинного навчання, яка за ключовими словами віднесе аналізований профіль до тих чи інших спеціальностей. Це потребує великої кількості людських ресурсів для розмітки статей та постійного оновлення даних. Ми пропонуємо використовувати інформаційні ресурси системи Dimensions, в якій понад 100 мільйонів публікацій вже категоризовано за ANZSRC-2020. За пошуковим

запитом у формі ключового слова Dimensions формує видачу, в якій зазначається, скільки публікацій із цим ключовим словом віднесено до кожної зі спеціальностей. Схематично ця процедура зображена на рис. 1. З нього також видно, що в базі розмічених документів стаття може категоризуватися до кількох спеціальностей, наприклад, *Стаття 1* віднесена до *Науки 1* та *Науки 2*. На основі такої видачі можна побудувати розподіл появи ключового слова в контексті різних наук. Наприклад, для ключового слова з рис. 1 розподіл появи матиме такий вигляд: *Наука 1* – 1 поява, *Наука 2* – 1 поява, *Наука 3* – 2 появи. На основі такого розподілу далі виконується категоризація ключового слова "*neural network*" у межах системи класифікації наук. Для категоризації множини ключових слів застосуємо алгоритм із [22], який опирається на ресурси та сервіси інформаційної системи Dimensions. Цей алгоритм враховує як появу ізольованих ключових слів із профілю рецензента чи заявки, так і спільну появу пар ключових слів. Алгоритм дає змогу відфільтрувати інформаційні шуми, що спричиненні як стоп-словами, так і рідкими ключовими словами, достовірність висновків за якими низька.

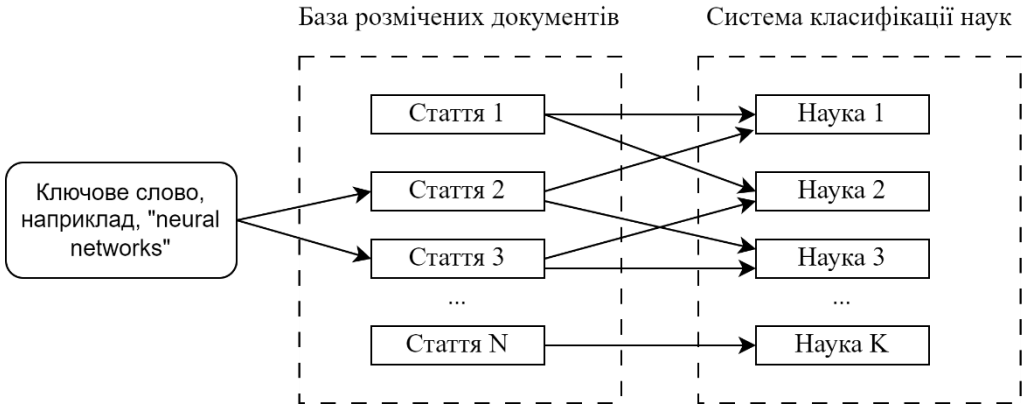

Рисунок 1. Схематичне зображення категоризації ключового слова

Внаслідок категоризації профіль заявки у формі множини її ключових слів $A_w = \{w_1, w_2, ..., w_n\}$ перетворюється у профіль заявки у формі категоріального розподілу за спеціальностям $A_t = \{\mu_{t_1}(A), \mu_{t_2}(A), ..., \mu_{t_m}(A)\}$, де $\mu_{t_i}(A) \in [0;1]$ – ступінь належності заявки A до спеціальності t_i, $i = \overline{1, m}$. Аналогічно, профіль рецензента у формі множини його ключових слів чи дослідницьких інтересів $R_w = \{w_1, w_2, ..., w_n\}$ перетворюється у профіль рецензента у формі категоріального розподілу за спеціальностям $R_t = \{\mu_{t_1}(R), \mu_{t_2}(R), ..., \mu_{t_m}(R)\}$.

Визначення схожості між профілями заявки та рецензента

Для зіставлення рецензентів та заявок потрібно знати, наскільки схожі 2 категоріальні розподіли – розподіл за спеціальностями ключових слів рецензента та розподіл за спеціальностями ключових слів заявки. У категоріальному просторі схожість двох об'єктів визначається зазвичай як суперпозиція схожості об'єктів за кожною категорією. Найчастіше – це сума схожості за окремими категоріями, коли кожна категорія розглядається

незалежно та ізольовано від інших. Усі метрики схожості з оглядових статей [23, 24] базуються на припущенні, що спорідненість між категоріями відсутня. Але деякі наукові спеціальності є спорідненими, зокрема для ANZSRC-2020 в [25] виявлено 20 пар сильно споріднених спеціальностей, 41 пару з середньою спорідненістю та 70 пар зі слабкою спорідненістю. Тому схожість доцільно розраховувати не лише напряму, як схожість між еквівалентними спеціальностями, але врахувати і перехресну схожість для споріднених спеціальностей. Для цього застосуємо метрику з [26]. Метрика розраховує схожість двох об'єктів X та Y із такими категоріальними розподілами $(\mu_1(X), \mu_2(X), ..., \mu_m(X))$ та $(\mu_1(Y), \mu_2(Y), ..., \mu_m(Y))$, де m – кількість категорій, якими в нашому випадку є наукові спеціальності, $\mu_i(X)$ – ступінь належності об'єкта X до i-ї категорії, $\mu_i(Y)$ – ступінь належності об'єкта Y до i-ї категорії, $i = \overline{1, m}$. Розподіли мають бути нормалізованими, тобто задовольняти такі умови:

$$\mu_i(X) \in [0;1], \qquad \mu_i(Y) \in [0;1], \qquad i = \overline{1, m};$$

$$\sum_{i=\overline{1, m}} \mu_i(X) = 1;$$

$$\sum_{i=\overline{1, m}} \mu_i(Y) = 1.$$

За метрикою [26], схожість об'єктів X та Y визначається так:

$$Fit(X, Y) = \sum_{i=\overline{1, m}} \min(\mu_i(X), \mu_i(Y)) + \Delta F(X, Y), \tag{1}$$

де $\sum_{i=\overline{1, m}} \min(\mu_i(X), \mu_i(Y))$ – доданок, що оцінює безпосередню (пряму) схожість об'єктів X та Y;

$\Delta F(X, Y)$ – доданок, що враховує схожість об'єктів X та Y через споріднені категорії.

Після розрахунку прямої схожості отримуємо такі залишки належності:

$r_i(X) = \max(0, \mu_i(X) - \mu_i(Y))$, $r_i(Y) = \max(0, \mu_i(Y) - \mu_i(X))$, $i = \overline{1, m}$.

Врахуємо внесок залишків у схожість двох об'єктів через спорідненість категорій. Вважатимемо, що інформація про попарну спорідненість категорій подана у формі такого бінарного відношення:

$$\mathbf{K} = \|k_{ij}\|,$$

де $k_{ij} \in [0;1]$– коефіцієнт спорідненості i-ї та j-ї категорій, $i = \overline{1,m}$, $j = \overline{1,m}$.

Чим більш подібні категорії, тим вищий коефіцієнт спорідненості. Відношення спорідненості є симетричним та рефлексивним, відповідно $k_{ij} = k_{ji}$ та $k_{ii} = 1$.

Композицію залишків представимо такою матрицею:

$$\mathbf{E} = \|e_{ij}\|,$$

де $e_{ij} = \min(r_i(X), r_j(Y))$, $i = \overline{1,m}$, $j = \overline{1,m}$.

Внесок залишків через перехресну спорідненість категорій розраховується так:

$$\Delta F(X,Y) = \sum_{i=1,\,m} \sum_{j=1,\,m} e_{ij} \cdot k_{ij} .$$

Підбір рецензентів як задача оптимізації

Розглядається задача підбору колективу рецензентів, які сукупно найкраще підходять для експертизи заявки. Для цієї задачі можливі 2 випадки: формування колективу з нуля та доповнення колективу новими членами.

Вважатимемо відомими профіль заявки $A_t = \{\mu_{t_1}(A), \mu_{t_2}(A), ..., \mu_{t_m}(A)\}$ та профілі k потенційних рецензентів $R_{tj} = \{\mu_{t_1}(R_j), \mu_{t_2}(R_j), ..., \mu_{t_m}(R_j)\}$, $j = \overline{1,k}$ у просторі з m спеціальностей. Усю множину рецензентів позначимо як $\mathbf{R} = \{R_1, R_2, .., R_k\}$.

Потрібно знайти підмножину рецензентів $S \subset \mathbf{R}$, яка має найбільшу сукупну відповідність заявці:

$$Fit(A, Agg(S)) \to \max .$$

де $Agg(S)$ - функція агрегації категоріальних розподілів множини відібраних рецензентів.

Агрегацію категоріальних розподілів за профілями рецензентів R_{tj}, $j = \overline{1,k}$ у просторі спеціальностей пропонується реалізувати за третім етапом алгоритму категоризації з [22].

Кількість рецензентів для однієї заявки позначимо через $c = |S|$. Ця кількість є сталою; зазвичай це від 2 до 5 осіб. Рівень відповідності між заявкою та колективом рецензентів розрахуємо за формулою (1).

Алгоритми підбору членів спецради

Задача підбору опонентів з математичного погляду – це пошук підмножини фіксованої потужності з деякої множини. Для вирішення таких задач на практиці застосовуються переважно наближені алгоритми. Серед множини можливих алгоритмів необхідно обрати той, який забезпечує баланс за показниками якості підбору та витратами ресурсів на знаходження розв'язку. У цій роботі пропонується використати такі алгоритми.

Повний перебір. Оптимальний варіант можна знайти повним перебором. Для заявки *A* необхідно серед усіх можливих *c*-ок із елементів множини потенційних рецензентів **R** обрати таку, що забезпечує максимальне значення відповідності. Складність такого перебору зростає майже експоненційно, тому навіть для задач середньої розмірності перебрати всі можливі варіанти та вкластися в якісь часові обмеження нереально. Причому кількість варіантів дуже сильно залежить від *c*. Наприклад, якщо потрібно обрати двох рецензентів зі 100, то потрібно перебрати 4 950 варіантів. А якщо якщо обирати трьох рецензентів зі 100, тоді кількість варіантів зросте до 161 700.

Повний перебір на проріджений множині кандидатів. На практиці кандидати з низьким рівнем схожості навряд чи будуть призначені рецензентами заявки. Тому раціональним кроком буде нехтування потенційними рецензентами з дуже низькою схожістю. Відкинувши

кандидатів із низькою схожістю з дисертацією, наприклад, на рівні 0.1 чи 0.2, можна відчутно скоротити перебір. Чим сильніше проріджуватимемо початковий список кандидатів, тим меншою буде тривалість оптимізації, але водночас зростають ризики задалеко відхилитися від оптимуму.

Жадібний алгоритм. Суть полягає у тому, що рецензенти підбираються ітеративно з забезпечення на кожному кроці максимальної відповідності поточного фрагмента колективу заявці. Алгоритм виконується за c ітерацій. На кожній ітерації до колективу рецензентів додається один новий член, який на цій ітерації максимізує рівень відповідності поточного складу заявці. На першій ітерації знаходимо кандидата з найвищою схожістю з дисертацією. На другій ітерації обираємо кандидата, який сукупно зі вже підібраним членом ради має найвищу відповідність дисертації. Кількість операцій перебору за такого підходу значно зменшується, але розв'язок може вийти неоптимальним. Для задачі підбору опонентів, окрім вище описаного класичного варіанта жадібного алгоритму, можливий варіант із елітизмом. Елітизм полягає в тому, що спочатку додається кандидат із найбільшим рівнем відповідності дисертації. При цьому рівень відповідності оновленого фрагмента ради дисертації не враховується. Далі, інші опоненти підбираються за класичним жадібним алгоритмом, тобто призначаються кандидати, які на поточній ітерації максимізують рівень відповідності колективу експертів дисертації. Жадібний алгоритм, особливо його елітарна реалізація, суттєво скорочують тривалість оптимізації.

Ізольований підбір. Найпростішим способом призначення рецензентів є обрання тих, які найбільш схожі з профілем дисертації. При цьому відповідність колективу рецензентів дисертації не враховується. Вважається, що чим сильніше кожен із кандидатів відповідає тематиці дисертації, тим кращою буде і разова рада. Грубо кажучи, вважається, що рівень відповідності ради є сумою рівнів відповідності кожного члена. Алгоритмічно, ізольований підбір реалізується сортуванням кандидатів за спаданням рівня схожості до дисертації та відбором перших c кандидатів. Це дуже швидкий алгоритм, але з малими шансами потрапити в оптимум. За такого алгоритму можливі ситуації, коли всі члени разової ради відповідатимуть лише одному і тому ж фрагмента тематики дисертації, а інші фрагменти тематики достовірно оцінити не буде кому.

Датасет дисертацій

Для експериментів із підбору рецензентів сформуємо датасет дисертацій [27]. Для цього скористаємось інформаційною системою НАЗЯВО – NAQA.Svr. У цю систему подаються заявки на разові спецради для їх затвердження Міністерством освіти і науки України. Кожна дисертація містить перелік супровідних документів та дані про разову раду, що пропонується закладом. Ми зібрали інформацію за 67 дисертаціями: 17 із них відхилені міністерством через слабку відповідність тематики статей членів ради дисертації, а 50 дисертацій було захищено. У ті 50 дисертацій увійшли і 17 раніше відхилених, для яких сформували нові ради. Зібрані дисертації належать до різних спеціальностей (рис. 2) з домінуванням спеціальностей 12-ї галузі, а саме 121, 122, 123, 124, 125 та 126. Кожен запис у датасеті включає унікальний ідентифікатор дисертації, прізвище та ім'я здобувача, ключові

слова дисертації англійською мовою, список членів разової ради разом з ключовими словами їх статей та статус ради – відхилена чи схвалена міністерством.

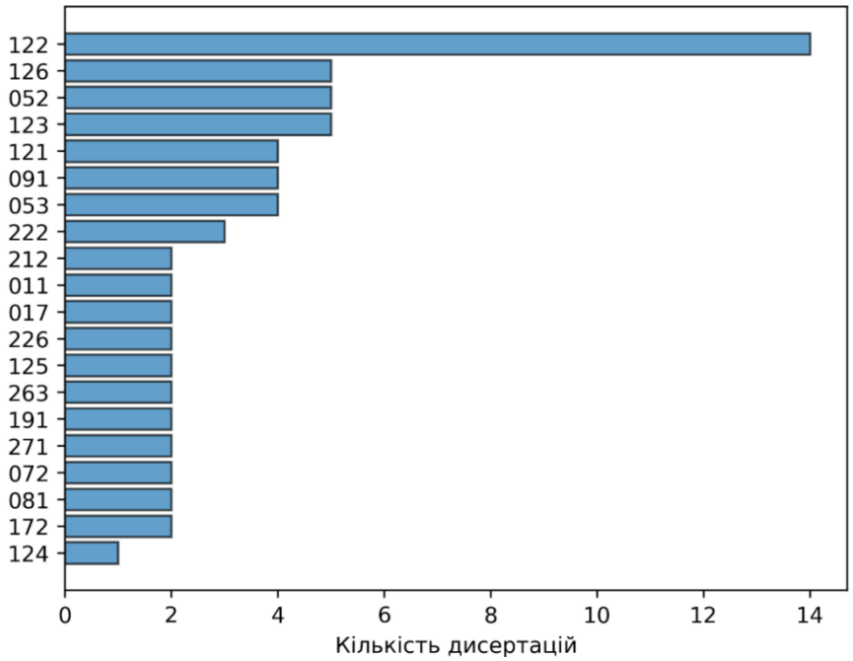

Рисунок 2. Розподіл дисертацій датасету за спеціальностями

На рис. 3 подано ранговий розподіл відповідності дисертації разовій раді, яка запропонована закладом. Більша частина разових рад мають відповідність вище 0.2 (62 ради), а решта мають незначущий рівень відповідності (5 рад). Міжквартильний інтервал приблизно дорівнює [0.4; 0.8].

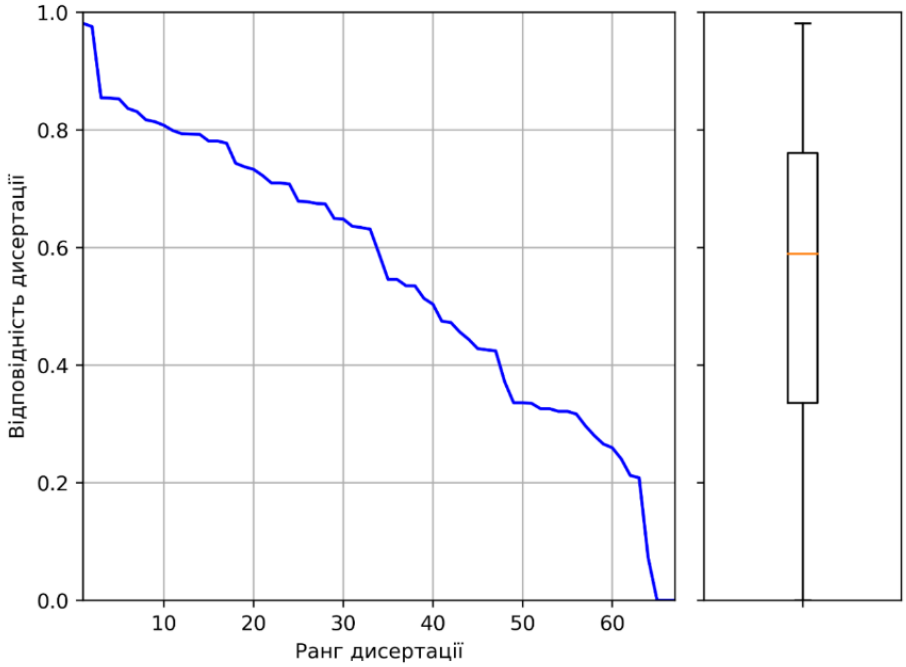

Рисунок 3. Ранговий розподіл відповідності дисертації разовій раді, яку сформував заклад

Експерименти з підбору опонентів

Проведемо експерименти з підбору опонентів на сформованому датасеті дисертацій. Для цього спочатку категоризуємо ключові слова дисертації за алгоритмом категоризації ключових слів у межах спеціальностей наук з ANZSRC-2020. Далі, аналогічним способом категоризуємо ключові слова статей членів разових рад. Пари ключових слів будемо поєднувати у додаткові запити лише в межах однієї статті. Для кожної ради вилучимо опонентів і спробуємо підібрати кращих із членів інших разових рад. Після вилучення опонентів отримуємо множину фрагментів разових рад, що містять голову та двох або одного рецензентів. Необхідно знайти опонентів, додавання яких до фрагментів разових рад забезпечить їх максимально можливу відповідність тематиці дисертацій.

Результати підбору опонентів порівняємо з варіантом разової ради, який сформовано закладом. Кількісно ефект оцінимо середнім рівнем зміни відповідності разових рад:

$$E = \frac{\sum\limits_{i=\overline{1,N}} F_i^{new} - F_i^{current}}{\sum\limits_{i=\overline{1,N}} F_i^{current}} \cdot 100\% ,$$

де N – кількість дисертацій;

F_i^{new} – рівень відповідності разової ради i-ї дисертації після оптимізації, $i = \overline{1,N}$;

$F_i^{current}$ – рівень відповідності разової ради i-ї дисертації до оптимізації, $i = \overline{1,N}$.

На рис. 4 подано результати оптимізації за різних алгоритмів підбору. Майже для всіх випадків разові ради від закладу мають нижчу відповідність тематиці дисертацій, ніж знайдені за будь-яким алгоритмом підбору. В умовах ручного формування рад закладом та обмежених можливостей щодо вибору членів рад отримуємо середній рівень відповідності тематиці. З іншого боку, за автоматичного підбору членів ради та достатнього великого пулу кандидатів отримуємо значне покращення рад лише за рахунок підбору опонентів.

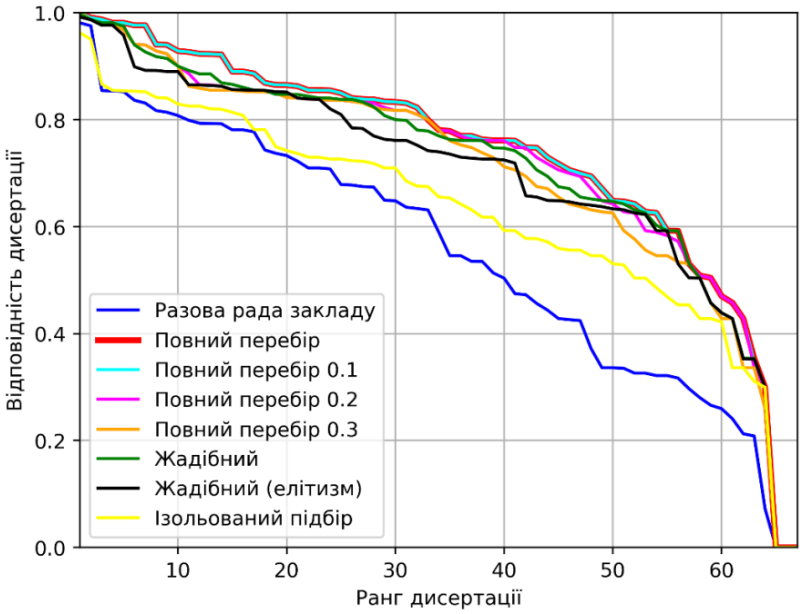

Рисунок 4. Результати підбору рад за різних алгоритмів оптимізації

На рис. 5 та 6 порівнюються рівні відповідності разових рад від закладу зі знайденими варіантами рад. За повного перебору наявне значне покращення відповідності більшості рад – як тих, які були відхилені, так і всіх інших. Деякі ради не покращено або рівень покращення низький. Це зумовлено насамперед тим, що в датасеті розподіл дисертацій за спеціальностями є нерівномірним (див. рис. 2) і датасет має малий обсяг. За найпростішого алгоритму на основі ізольованого підбору (див. рис. 6) більшість разових рад покращено, але відповідність кількох рад погіршилася. Погіршення пояснюється тим, що висока схожість кандидата з дисертацією не означає, що утворений за ізольованим підходом колектив буде покривати усю тематику дисертації.

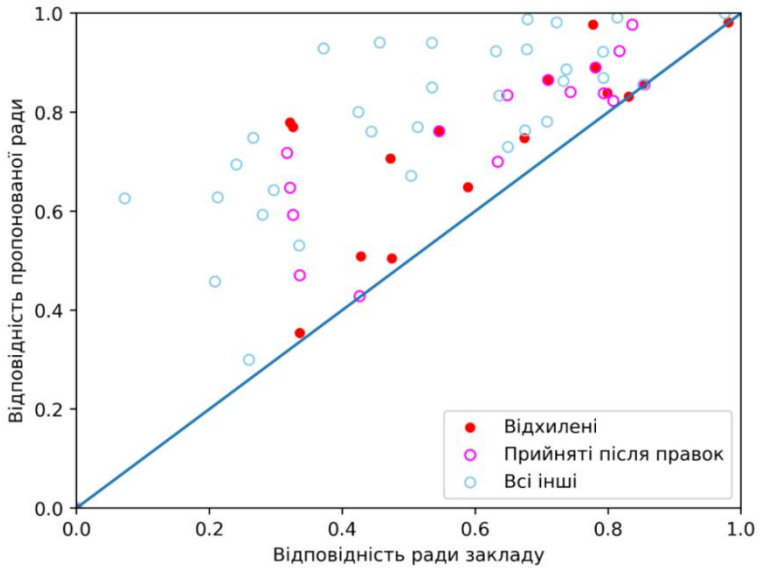

Рисунок 5. Порівняння рад від закладу з пропонованими радами, які сформовано за повним перебором

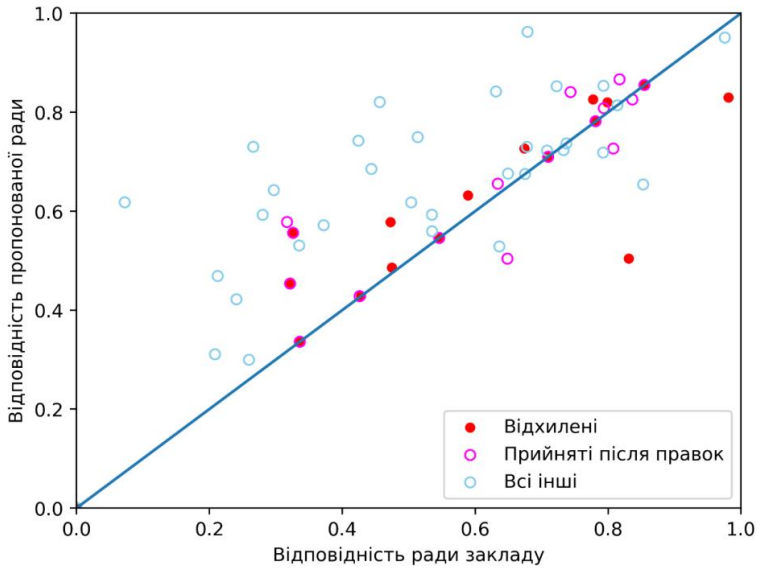

Рисунок 6. Порівняння рад від закладу з пропонованими радами, які сформовано за ізольованим підбором

На рис. 7 порівнюються результати підбору рад за різними алгоритмами оптимізації. Перебір на проріджений множині кандидатів за порогу у 0.3 однозначно невдалий. Усі інші утворюють множину Парето. Тому під час вибору алгоритму необхідно врахувати пріоритети – потрібен швидкий результат чи якісний. З рис. 7 видно, що рівень покращення внаслідок переходу з жадібного пошуку на алгоритми повного перебору зростає повільно. Але тривалість оптимізації зростає суттєво. Тому найбільш збалансованим можна вважати жадібний алгоритм підбору без елітизму. Альтернативою йому може бути алгоритм повного перебору з проріджуванням множини кандидатів за рівнем відповідності в околі 0.25. Але треба мати на увазі, що ці висновки ґрунтуються на експериментах на невеликому датасеті. За реальних баз даних великого обсягу тривалість оптимізації за алгоритмами повного перебору може зрости занадто сильно.

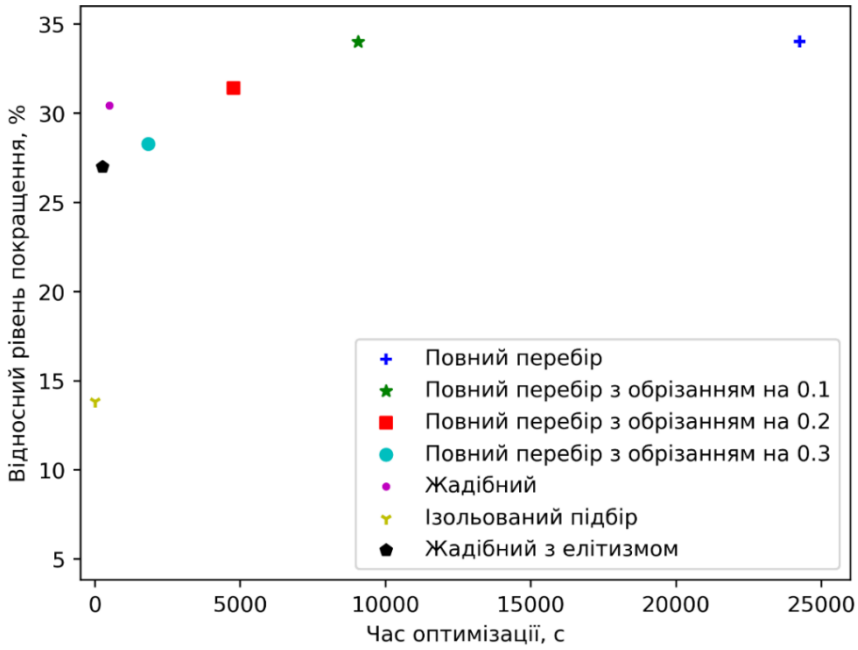

Рисунок 7. Оцінка алгоритмів підбору опонентів за критеріями «тривалість – якість»

Приклад покращення разової ради від закладу

Розглянемо, як внаслідок оптимізації вдалося покращити склад разової ради, порівняно зі складом від закладу, на прикладі дисертації *Моделі та методи обробки даних системи віддаленого моніторингу стану пацієнтів з цукровим діабетом* авторства *Віталія Левківського*; ідентифікатор дисертації в НАЗЯВО – *4756*.

Ключові слова дисертації:

edge devices; *IoT;*
diagnostics; *diseases;*
intelligent data analysis; *information technologies;*
medical information systems; *modeling;*
monitoring; *data processing;*
patient; *forecasting;*
software component model; *system design;*
diabetes.

З переліку ключових слів зрозуміло, що тематикою дисертації є медичні інформаційні системи і технології отримання та обробки медичних даних. Для категоризації початкову множину ключових слів розширимо попарним поєднанням ключових слів. Це дасть змогу під час категоризації надати перевагу тим науковим спеціальностям, у яких одночасно зустрічаються пари ключових слів [22].

Результат категоризації множини ключових слів дисертації є таким:

4605 Data Management and Data Science – 0.382;

4606 Distributed Computing and Systems Software – 0.255;

4609 Information Systems – 0.205;

4203 Health Services and Systems – 0.158.

Назві наукової спеціальності передує її чотирицифровий ідентифікатор в ANZSRC-2020. Саме ці ідентифікатори використовуватимемо для подальшого викладу матеріалу. Аналізована дисертація представляється таким вектором: $A_t = \left(\frac{0.382}{4605}, \frac{0.255}{4606}, \frac{0.205}{4609}, \frac{0.158}{4203} \right)$.

У базі НАЗЯВО тематику досліджень кожного члена ради представлено ключовими словами 3 або 4 його статей. Для їх категоризації застосуємо принцип мішка ключових слів. Категоризація відбувається так: 1) для кожної множини ключових слів однієї статті додаємо їх попарні поєднання; 2) об’єднуємо отримані множини ключових слів різних статей в одну множину – в один мішок; 3) категоризуємо отриману множину ключових слів за алгоритмом [22].

Ключові слова статей голови разової ради є такими:

1) *information technology, cardiovascular system, diagnostics, SCORE, systematic coronary risk evaluation, blood pressure, diseases, medical information systems;*

2) *information technology, data processing, data, data visualization, COVID-19, mobile application, disease, medical information systems;*

3) *differential equations, acceleration, cloud computing, task analysis, computational complexity, computer simulation, computerized integration procedure, integration procedure, numerical method.*

Результат категоризації голови ради вийшов таким:

4609 Information Systems – 0.381;

4203 Health Services and Systems – 0.225;

4606 Distributed Computing and Systems Software – 0.214;

4601 Applied Computing – 0.180.

Ключові слова статей першого рецензента є такими:

1) *IoT, monitoring system, microclimate parameters, educational institutions;*

2) *biotechnical system, edge device, photoplethysmograph, photoplethysmography, pulse wave, saturation sensor, cardiovascular system, data handling, edge computing, optoelectronic devices, quantum optics, circulatory disorders, functional state, mode of operations, non-contact sensors, photoplethysmographic, physical conditions, pulse wave signal, technical realization, digital storage;*

3) *pandemic, health-saving educational environment, model, information and digital environment;*

4) *planning technologies, use case, project, systems design, project complexity.*

Результат категоризації першого рецензента вийшов таким:

4606 Distributed Computing and Systems Software – 0.337;

4605 Data Management and Data Science – 0.256;

4003 Biomedical Engineering – 0.244;

3208 Medical Physiology – 0.162.

Ключові слова статей другого рецензента є такими:

1) *mathematical model, biotechnical system, edge devices, methods of studying, human body health, hemodynamic characteristics, diagnostic systems, digital signal processing, information technology;*

2) *biotechnical system, edge device, photoplethysmograph, photoplethysmography, pulse wavesaturationsensor, cardiovascular system, data handling, edge computing, optoelectronic devices, quantum optics, circulatory disorders, functional state, mode of operations, non-contact sensors, photoplethysmographic, physical conditions, pulse wave signal, technical realization, digital storage;*

3) *IoT, monitoring system, microclimate parameters, educational institutions, edge devices.*

Результат категоризації другого рецензента вийшов таким:

4606 Distributed Computing and Systems Software – 0.426;

4605 Data Management and Data Science – 0.299;

4003 Biomedical Engineering – 0.138;

4604 Cybersecurity and Privacy – 0.135.

Ключові слова статей першого опонента є такими:

1) *information expert system, control, method of fuzzy sets, medical diagnostics, diabetes, dentistry;*

2) *medical information technologies, medical information systems, coronary channels, coronary artery disease;*

3) *intelligent technologies, computer planning, modeling, medical diagnostics, nasal breathing testing, treatment, rehabilitation, medicine, tissue trophic complex, magnetic data analysis, electrocardiographic data analysis, audiological data analysis;*

4) *medical expert systems, fuzzy logic, coronary artery disease, coronary arteries, problems of cardiology, cardiovascular diseases, myocardial infarction, patient safety.*

Результат категоризації першого опонента вийшов таким:

3201 Cardiovascular Medicine and Haematology – 0.387;

3203 Dentistry – 0.215;

4605 Data Management and Data Science – 0.205;

4602 Artificial Intelligence – 0.192.

Ключові слова статей другого опонента є такими:

1) *Data analysis, Computational modeling, Neural networks, Predictive models, Safety, Personnel, Task analysis, hypertension, medical robotic platforms, TIMA, health and safety digital competencies;*

2) *Medical Diagnosis, Forecasting, Neuroevolution, Synthesis, Adaptive Mechanism, Genetic Algorithm, Parallel Genetic Algorithm, Crossover, Hospital data processing, Intelligent systems, Neural networks, Crossover operator, Medical diagnostics, Modern computer systems, Network synthesis, Neuro evolutions, Parallelizations, Resource consumption, Synthesis process, Diagnosis;*

3) *dermatoscopy, medical diagnosis, convolutional neural network, skin disease, ResNet50 model, software component model.*

Результат категоризації другого опонента вийшов таким:

4602 Artificial Intelligence – 0.435;

4611 Machine Learning – 0.357;

4605 Data Management and Data Science – 0.208.

Зведемо отримані результати в табл. 1. З неї видно, що всі члени разової ради мають значний рівень схожості з дисертацією. Агрегуємо результати категоризації усіх членів разової ради, щоб знайти рівень відповідності разової ради дисертації. Агрегування реалізуємо третім етапом алгоритму категоризації ключових слів з [22]. Внаслідок агрегування отримуємо:

$$Agg\left(\begin{array}{l}\left(\frac{0.381}{4609},\frac{0.225}{4203},\frac{0.214}{4606},\frac{0.180}{4601}\right)\\ \left(\frac{0.337}{4606},\frac{0.256}{4605},\frac{0.244}{4003},\frac{0.162}{3208}\right)\\ \left(\frac{0.426}{4606},\frac{0.299}{4605},\frac{0.138}{4003},\frac{0.135}{4604}\right)\\ \left(\frac{0.387}{3201},\frac{0.215}{3203},\frac{0.205}{4605},\frac{0.192}{4602}\right)\\ \left(\frac{0.435}{4602},\frac{0.357}{4611},\frac{0.208}{4605}\right)\end{array}\right)=\left(\frac{0.389}{4606},\frac{0.374}{4605},\frac{0.236}{4602}\right).$$

Таблиця 1. Результати категоризації профілів членів разової ради від закладу

Член разової ради	Профіль члена ради R_{jt}	Схожість між дисертацією та членом ради $Fit(A, R_j)$
Голова	$\left(\frac{0.381}{4609},\frac{0.225}{4203},\frac{0.214}{4606},\frac{0.180}{4601}\right)$	0.577
Рецензент 1	$\left(\frac{0.337}{4606},\frac{0.256}{4605},\frac{0.244}{4003},\frac{0.162}{3208}\right)$	0.564
Рецензент 2	$\left(\frac{0.426}{4606},\frac{0.299}{4605},\frac{0.138}{4003},\frac{0.135}{4604}\right)$	0.521
Опонент 1	$\left(\frac{0.387}{3201},\frac{0.215}{3203},\frac{0.205}{4605},\frac{0.192}{4602}\right)$	0.239
Опонент 2	$\left(\frac{0.435}{4602},\frac{0.357}{4611},\frac{0.208}{4605}\right)$	0.227

Рівень відповідності разової ради до дисертації становить: $Fit\left(\left(\frac{0.382}{4605},\frac{0.255}{4606},\frac{0.205}{4609},\frac{0.158}{4203}\right),\left(\frac{0.389}{4606},\frac{0.374}{4605},\frac{0.236}{4602}\right)\right)=0.631$. Це доволі непоганий рівень відповідності, який в основному обумовлено сильним співпадінням за двома спеціальностями із чотирьох, а саме за *4606 Distributed Computing and Systems Software* та *4605 Data Management and Data Science*. Під час розрахунку рівня відповідності також враховано спорідненості спеціальностей *4602* та *4605* та спеціальностей *4605* та *4606*, які дорівнюють 0.122 та 0.124, відповідно [25].

Спробуємо підібрати кращих опонентів, щоб підвищити сукупну відповідність разової ради дисертації. Як потенційних кандидатів візьмемо членів усіх інших разових рад сформованого датасету. Внаслідок оптимізації опонентами призначено *Оксану Мелеховець* із відповідністю тематиці дисертації на рівні 0.158 та *Камілу Сторчак* із відповідністю тематиці дисертації на рівні 0.542.

Ключові слова статей першого запропонованого опонента є такими:

1) *trophic ulcers, diabetes mellitus, chronic venous insufficiency, photodynamic therapy, plasma therapy;*

2) *diabetes mellitus, diabetic foot, photodynamic therapy, autologous plasma;*

3) *type 2 diabetes, middle age, obesity, quality of life, physical therapy program;*

4) *type 2 diabetes, middle age, obesity, physical therapy program.*

Результат категоризації першого запропонованого опонента вийшов таким:

3210 Nutrition and Dietetics – 0.274;

4203 Health Services and Systems – 0.261;

4202 Epidemiology – 0.251;

3205 Medical Biochemistry and Metabolomics – 0.214.

Ключові слова статей другого запропонованого опонента є такими:

1) *intelligent data analysis, data mining system, neural network;*

2) *web system, machine learning, web development, Heroku, Streamlit;*

3) *machine learning methods, neural network, smart home, failures, prediction, data analysis, decision theory, information technology.*

Результат категоризації другого запропонованого опонента вийшов таким:

4605 Data Management and Data Science – 0.555;

4611 Machine Learning – 0.306;

4609 Information Systems – 0.139.

Виконаємо агрегування профілів членів ради за нових опонентів:

$$Agg\left(\begin{array}{l}\left(\frac{0.281}{4611},\frac{0.269}{4605},\frac{0.228}{4602},\frac{0.222}{4608}\right)\\ \left(\frac{0.556}{4612},\frac{0.302}{4602},\frac{0.142}{4007}\right)\\ \left(\frac{0.457}{4611},\frac{0.196}{4603},\frac{0.183}{4605},\frac{0.163}{4609}\right)\\ \left(\frac{0.274}{3210},\frac{0.261}{4203},\frac{0.251}{4202},\frac{0.214}{3205}\right)\\ \left(\frac{0.555}{4605},\frac{0.306}{4611},\frac{0.139}{4609}\right)\end{array}\right)=\left(\frac{0.361}{4605},\frac{0.335}{4606},\frac{0.156}{4609},\frac{0.148}{4203}\right).$$

Рівень відповідності дисертації разовій раді з новими опонентами становить: $Fit\left(\left(\frac{0.382}{4605},\frac{0.255}{4606},\frac{0.205}{4609},\frac{0.158}{4203}\right),\left(\frac{0.361}{4605},\frac{0.335}{4606},\frac{0.156}{4609},\frac{0.148}{4203}\right)\right)=0.923$. Порівнюючи з разовою радою від закладу, бачимо суттєве покращення рівня відповідності – профіль нової ради описується всіма тими самими спеціальностями, що і дисертація. Покращення становить приблизно 46%.

З наведеного прикладу видно, що хоча індивідуальна схожість окремого члена ради дисертації може бути і посередньою, проте сукупно рівень відповідності ради може вийти високим. Це відбувається завдяки тому, що новий опонент покриває так звану мінорну частину тематики дисертації, яка знаходиться поза полем експертизи інших членів ради. В аналізованому прикладі це частина тематики, яка відповідає спеціальностям *4609 Information Systems* та *4203 Health Services and Systems* (рис. 8).

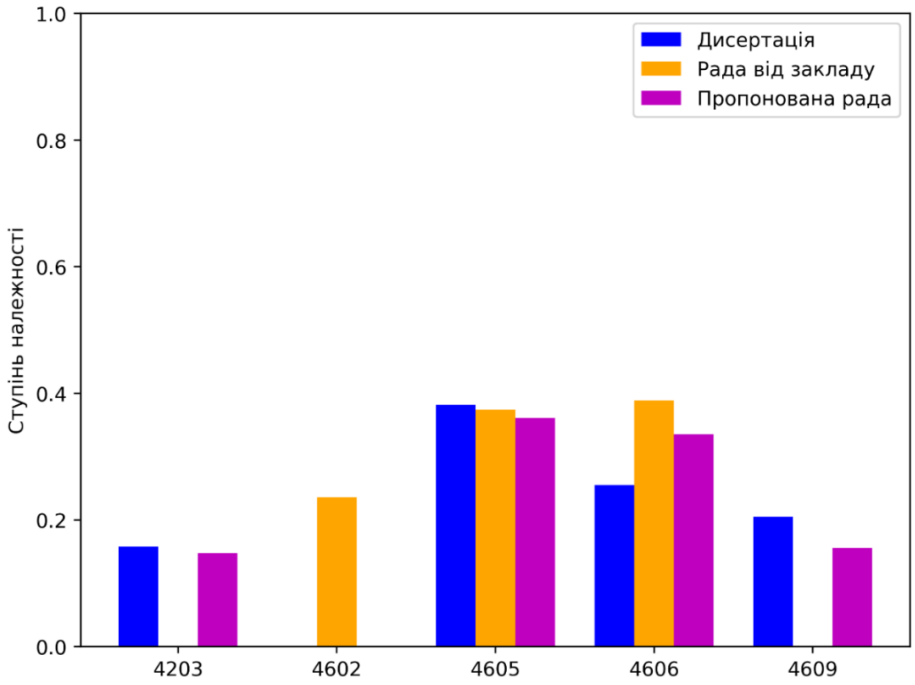

Рисунок 8. Порівняння категоризації дисертації та двох разових рад

Висновки

У роботі запропоновано швидкий метод підбору разової ради для захисту PhD-дисертацій, який, на відміну від ізольованого підбору рецензентів, враховує здатність саме ради як колективу спільно оцінити дисертацію за усіма аспектами її тематики. На першому етапі підбору здійснюється категоризація дисертації та потенційних членів разової ради шляхом представлення їх профілів як векторів у просторі наукових спеціальностей з ANZSRC-2020. На другому етапі розраховуються рівні відповідності потенційних членів ради тематиці дисертації з урахуванням спорідненості наукових спеціальностей. На третьому етапі підбирається склад ради, яка відповідає тематиці дисертації з максимально можливим ступенем. Для реалізації третього етапу запропоновано алгоритми на основі повного перебору на усій множині кандидатів, повного перебору на прорідженій множині кандидатів, на основі жадібного підходу без елітизму та з елітизмом, та за простим ізольованим пошуком. Тестування алгоритмів на сформованому датасеті із 67 PhD-дисертацій показало, що найкращий баланс за критеріями якості підбору та витрат ресурсів на пошук колективу забезпечує жадібний алгоритм без елітизму та повний перебір на прорідженій множині кандидатів. Внаслідок оптимізації вдалося покращити склад разових рад в середньому на 13–34% залежно від типу використаного алгоритму. Запропонований метод може

застосовуватись для покращення ефективності управління процесами призначення рецензентів для експертизи доповідей на конференціях, рецензування рукописів журнальних статей, експертизи наукових проєктів та грантових заявок, експертизи дисертацій тощо. Метод може використовуватися і в аудиторських цілях для швидкої перевірки коректності сформованих разових рад з подальшим відбором підозрілих справ для ґрунтовної ресурсовитратної експертизи. Подальші дослідження можуть стосуватися застосування запропонованого методу експрес-підбору рецензентів у більш трудомістких та ітеративних процедурах формування колективу рецензентів, коли потрібно враховувати не лише відповідність тематиці роботи, але й відсутність конфлікту інтересів, баланс навантаження на рецензентів та інші можливі обмеження. Подальші дослідження також варто зосередити на створенні більшого та репрезентативного датасету для ефективної оцінки якості різноманітних підходів до підбору рецензентів. Доцільно також під час формування ради враховувати не лише відповідність тематики рецензентів та дисертації, а також і кваліфікаційний рівень експертів.

Подяка

Автори висловлюють подяку Digital Science & Research Solutions Inc. за надання доступу до ресурсів Dimensions за проєктом DIM-371.

Література

1. Zhao, X., & Zhang, Y. (2022). Reviewer assignment algorithms for peer review automation: A survey. *Information Processing and Management*, *59*(5). https://doi.org/10.1016/j.ipm.2022.103028
2. Петричко, М. В., & Штовба, С. Д. (2024). Автоматизація підбору наукових рецензентів: огляд задач і методів. *Вісник Вінницького політехнічного інституту*, (1), 56-64. https://doi.org/10.31649/1997-9266-2024-172-1-56-64
3. Wang, F., Shi, N., & Chen, B. (2010). A comprehensive survey of the reviewer assignment problem. *International Journal of Information Technology and Decision Making*, *9*(4), 645-668. https://doi.org/10.1142/S0219622010003993
4. Aksoy, M., Yanik, S., & Amasyali, M. F. (2023). Reviewer assignment problem: A systematic review of the literature. *Journal of Artificial Intelligence Research*. AI Access Foundation. https://doi.org/10.1613/JAIR.1.14318
5. Tan, S., Duan, Z., Zhao, S., Chen, J., & Zhang, Y. (2021). Improved reviewer assignment based on both word and semantic features. *Information Retrieval Journal*, *24*(3), 175-204. https://doi.org/10.1007/s10791-021-09390-8
6. Yarowsky, D., & Florian, R. (1999). Taking the load off the conference chairs: Towards a digital paper-routing assistant. In *Proceedings of the 1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora, EMNLP 1999* (pp. 220–230). Association for Computational Linguistics (ACL).
7. Karimzadehgan, M., Zhai, C. X., & Belford, G. (2008). Multi-aspect expertise matching for review assignment. In *Proceedings of International Conference on Information and Knowledge Management* (pp. 1113–1122). https://doi.org/10.1145/1458082.1458230

8. Mirzaei, M., Sander, J., & Stroulia, E. (2019). Multi-aspect review-team assignment using latent research areas. *Information Processing and Management*, *56*(3), 858–878. https://doi.org/10.1016/j.ipm.2019.01.007
9. Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent Dirichlet allocation. *Journal of Machine Learning Research*, *3*(4–5), 993–1022. https://doi.org/10.7551/mitpress/1120.003.0082
10. Ekinci, E., & Omurca, S. I. (2020). NET-LDA: A novel topic modeling method based on semantic document similarity. *Turkish Journal of Electrical Engineering and Computer Sciences*, *28*(4), 2244–2260. https://doi.org/10.3906/ELK-1912-62
11. Anjum, O., Gong, H., Bhat, S., Xiong, J., & Hwu, W. M. (2019). Pare: A paper-reviewer matching approach using a common topic space. In *EMNLP-IJCNLP 2019 – 2019 Conference on Empirical Methods in Natural Language Processing and 9th International Joint Conference on Natural Language Processing, Proceedings of the Conference* (pp. 518–528). Association for Computational Linguistics. https://doi.org/10.18653/v1/d19-1049
12. Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013). Distributed representations ofwords and phrases and their compositionality. In *Advances in Neural Information Processing Systems*. Neural information processing systems foundation.
13. Pennington, J., Socher, R., & Manning, C. D. (2014). GloVe: Global vectors for word representation. In *EMNLP 2014 – 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference* (pp. 1532–1543). Association for Computational Linguistics (ACL). https://doi.org/10.3115/v1/d14-1162.
14. Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, *5*, 135–146. https://doi.org/10.1162/tacl_a_00051
15. Sun, C., Ng, K. T. J., Henville, P., & Marchant, R. (2019). Hierarchical word mover distance for collaboration recommender system. In *Communications in Computer and Information Science* (Vol. 996, pp. 289–302). Springer Verlag. https://doi.org/10.1007/978-981-13-6661-1_23
16. Kong, X., Jiang, H., Yang, Z., Xu, Z., Xia, F., & Tolba, A. (2016). Exploiting publication contents and collaboration networks for collaborator recommendation. *PLoS ONE*, *11*(2): e0148492. https://doi.org/10.1371/journal.pone.0148492
17. Howard, J., & Ruder, S. (2018). Universal language model fine-tuning for text classification. In *ACL 2018 – 56th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference* (Long Papers) (Vol. 1, pp. 328–339). Association for Computational Linguistics (ACL). https://doi.org/10.18653/v1/p18-1031
18. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL HLT 2019 – 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies – Proceedings of the Conference* (Vol. 1, pp. 4171–4186). Association for Computational Linguistics (ACL).
19. Alec, R., Jeffrey, W., Rewon, C., David, L., Dario, A., & Ilya, S. (2019). Language models are unsupervised multitask learners | Enhanced Reader. *OpenAI Blog*, *1*(8), 9. Retrieved from https://github.com/codelucas/newspaper

20. Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter. ArXiv 2019. *arXiv preprint arXiv:1910.01108.*
21. Zhao, Y., Tang, J., & Du, Z. (2019). EFCNN: A restricted convolutional neural network for expert finding. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 11440 LNAI, pp. 96–107). Springer Verlag. https://doi.org/10.1007/978-3-030-16145-3_8
22. Shtovba, S., & Petrychko, M. (2021). An algorithm for topic modeling of researchers taking into account their interests in Google Scholar profiles. In *CEUR Workshop Proceedings* (Vol. 2864 "The Fourth International Workshop on Computer Modeling and Intelligent Systems", pp. 299–311). CEUR-WS. https://doi.org/10.32782/cmis/2864-26
23. Jie, Y., Amores, J., Sebe, N., & Qi, T. (2006). A new study on distance metrics as similarity measurement. In *2006 IEEE International Conference on Multimedia and Expo, ICME 2006 – Proceedings* (Vol. 2006, pp. 533–536). https://doi.org/10.1109/ICME.2006.262443
24. Cha, S.-H. (2007). Comprehensive survey on distance/similarity measures between probability density functions. *City*, *1*(2), 1.
25. Штовба, С. Д., & Петричко, М. В. (2024). Ідентифікація рівня спорідненості наукових спеціальностей на основі даних системи Dimensions. *Проблеми програмування*, (1), 77–85. https://doi.org/10.15407/pp2024.01.077
26. Shtovba, S., Petrychko, M., & Shtovba, O. (2023). Similarity metric of categorical distributions for topic modeling problems with akin categories. In *CEUR Workshop Proceedings* (Vol. 3392 "The Sixth International Workshop on Computer Modeling and Intelligent Systems", pp. 76–85). CEUR-WS. https://doi.org/10.32782/cmis/3392-7
27. Petrychko, M., & Shtovba, S. (2024). Dataset for PhD theses reviewers assignments. *ResearchGate*. http://dx.doi.org/10.13140/RG.2.2.23147.35362

Рукопис отримано – 17/07/2024; прийнято до публікації – 29/07/2024.

Express assignment of reviewers for a PhD thesis defense committee

Serhiy Shtovba, Mykola Petrychko

Abstract

Today PhD thesis defense committee are formed manually. This causes both corruption risks and significant time spent on searching and analyzing candidates with a high chance of missing qualified opponents. Therefore, there is an interest in automating the formation of committees, which would allow to eliminate the mentioned risks of the human factor. The paper focuses on the express committee assignment when there is a need to narrow down a large list of candidates. The resulting short list can be analyzed either manually or processed by a fine-grained assignment procedure which is resource consuming and requires a much larger volume of initial information than the express assignment. A method of assigning a team of reviewers based on their relevance to the topic of the thesis is proposed, which, unlike the isolated assignment of candidates, takes into account the ability of the team of reviewers to jointly evaluate the work in terms of all aspects of its topic. The method is balanced in terms of assignment quality and resource costs criteria for the search of committee members. The method consists of 3 stages. At the first stage, the thesis and potential committee members are categorized by representing their topics with vectors in the space of research specialties from ANZSRC-2020. At the second stage, the level of correspondence of candidates to the topic of the thesis is calculated, taking into account the affinity of the research specialties of ANZSRC-2020. At the third stage, the committee is assigned, which corresponds to the topic of the thesis to the maximum possible extent. To implement the third stage, several optimization algorithms are proposed. Algorithm testing on the generated dataset of 67 PhD theses showed that the best balance in terms of assignment quality and resource costs criteria for team search provides a greedy algorithm without elitism and a complete search on a truncated set of candidates. As a result of the optimization, it was possible to improve the composition of committees by an average of 13-34%, depending on the type of algorithm used.

Keywords: reviewer assignment problem; express assignment; natural language processing; categorization; discrete optimization; data analysis; Dimensions.

References

1. Zhao, X., & Zhang, Y. (2022). Reviewer assignment algorithms for peer review automation: A survey. *Information Processing and Management, 59*(5). https://doi.org/10.1016/j.ipm.2022.103028
2. Petrychko, M., & Shtovba, S. (2024). Avtomatyzatsiia pidboru naukovykh retsenzentiv: Ohliad zadach i metodiv. *Visnyk Vinnytskoho politekhnichnoho instytutu*, (1), 56–64. https://doi.org/10.31649/1997-9266-2024-172-1-56-64
3. Wang, F., Shi, N., & Chen, B. (2010). A comprehensive survey of the reviewer assignment problem. *International Journal of Information Technology and Decision Making*, *9*(4), 645–668. https://doi.org/10.1142/S0219622010003993
4. Aksoy, M., Yanik, S., & Amasyali, M. F. (2023). Reviewer assignment problem: A systematic review of the literature. *Journal of Artificial Intelligence Research*. AI Access Foundation. https://doi.org/10.1613/JAIR.1.14318
5. Tan, S., Duan, Z., Zhao, S., Chen, J., & Zhang, Y. (2021). Improved reviewer assignment based on both word and semantic features. *Information Retrieval Journal*, *24*(3), 175–204. https://doi.org/10.1007/s10791-021-09390-8
6. Yarowsky, D., & Florian, R. (1999). Taking the load off the conference chairs: Towards a digital paper-routing assistant. In *Proceedings of the 1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora, EMNLP 1999* (pp. 220–230). Association for Computational Linguistics (ACL).
7. Karimzadehgan, M., Zhai, C. X., & Belford, G. (2008). Multi-aspect expertise matching for review assignment. In *Proceedings of International Conference on Information and Knowledge Management* (pp. 1113–1122). https://doi.org/10.1145/1458082.1458230

8. Mirzaei, M., Sander, J., & Stroulia, E. (2019). Multi-aspect review-team assignment using latent research areas. *Information Processing and Management*, *56*(3), 858–878. https://doi.org/10.1016/j.ipm.2019.01.007
9. Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent Dirichlet allocation. *Journal of Machine Learning Research*, *3*(4-5), 993–1022. https://doi.org/10.7551/mitpress/1120.003.0082
10. Ekinci, E., & Omurca, S. I. (2020). NET-LDA: A novel topic modeling method based on semantic document similarity. *Turkish Journal of Electrical Engineering and Computer Sciences*, *28*(4), 2244–2260. https://doi.org/10.3906/ELK-1912-62
11. Anjum, O., Gong, H., Bhat, S., Xiong, J., & Hwu, W. M. (2019). Pare: A paper-reviewer matching approach using a common topic space. In *EMNLP-IJCNLP 2019 – 2019 Conference on Empirical Methods in Natural Language Processing and 9th International Joint Conference on Natural Language Processing, Proceedings of the Conference* (pp. 518–528). Association for Computational Linguistics. https://doi.org/10.18653/v1/d19-1049
12. Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013). Distributed representations ofwords and phrases and their compositionality. In *Advances in Neural Information Processing Systems*. Neural information processing systems foundation.
13. Pennington, J., Socher, R., & Manning, C. D. (2014). GloVe: Global vectors for word representation. In *EMNLP 2014 – 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference* (pp. 1532–1543). Association for Computational Linguistics (ACL). https://doi.org/10.3115/v1/d14-1162.
14. Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, *5*, 135–146. https://doi.org/10.1162/tacl_a_00051
15. Sun, C., Ng, K. T. J., Henville, P., & Marchant, R. (2019). Hierarchical word mover distance for collaboration recommender system. In *Communications in Computer and Information Science* (Vol. 996, pp. 289–302). Springer Verlag. https://doi.org/10.1007/978-981-13-6661-1_23
16. Kong, X., Jiang, H., Yang, Z., Xu, Z., Xia, F., & Tolba, A. (2016). Exploiting publication contents and collaboration networks for collaborator recommendation. *PLoS ONE*, *11*(2): e0148492. https://doi.org/10.1371/journal.pone.0148492
17. Howard, J., & Ruder, S. (2018). Universal language model fine-tuning for text classification. In *ACL 2018 – 56th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference* (Long Papers) (Vol. 1, pp. 328–339). Association for Computational Linguistics (ACL). https://doi.org/10.18653/v1/p18-1031
18. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL HLT 2019 – 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies – Proceedings of the Conference* (Vol. 1, pp. 4171–4186). Association for Computational Linguistics (ACL).
19. Alec, R., Jeffrey, W., Rewon, C., David, L., Dario, A., & Ilya, S. (2019). Language models are unsupervised multitask learners | Enhanced Reader. *OpenAI Blog*, *1*(8), 9. Retrieved from https://github.com/codelucas/newspaper
20. Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter. ArXiv 2019. *arXiv preprint arXiv:1910.01108.*
21. Zhao, Y., Tang, J., & Du, Z. (2019). EFCNN: A restricted convolutional neural network for expert finding. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 11440 LNAI, pp. 96–107). Springer Verlag. https://doi.org/10.1007/978-3-030-16145-3_8

22. Shtovba, S., & Petrychko, M. (2021). An algorithm for topic modeling of researchers taking into account their interests in Google Scholar profiles. In *CEUR Workshop Proceedings* (Vol. 2864 "The Fourth International Workshop on Computer Modeling and Intelligent Systems", pp. 299–311). CEUR-WS. https://doi.org/10.32782/cmis/2864-26
23. Jie, Y., Amores, J., Sebe, N., & Qi, T. (2006). A new study on distance metrics as similarity measurement. In *2006 IEEE International Conference on Multimedia and Expo, ICME 2006 - Proceedings* (Vol. 2006, pp. 533–536). https://doi.org/10.1109/ICME.2006.262443
24. Cha, S.-H. (2007). Comprehensive survey on distance/similarity measures between probability density functions. *City*, *1*(2), 1.
25. Shtovba, S. & Petrychko, M., (2024). Identyfikatsiia rivnia sporidnenosti naukovykh spetsialnostei na osnovi danykh systemy Dimensions. *Problemy prohramuvannia*, (1), 77–85. https://doi.org/10.15407/pp2024.01.077
26. Shtovba, S., Petrychko, M., & Shtovba, O. (2023). Similarity metric of categorical distributions for topic modeling problems with akin categories. In *CEUR Workshop Proceedings* (Vol. 3392 "The Sixth International Workshop on Computer Modeling and Intelligent Systems", pp. 76–85). CEUR-WS. https://doi.org/10.32782/cmis/3392-7
27. Petrychko, M., & Shtovba, S. (2024). Dataset for PhD theses reviewers assignments. *ResearchGate*. http://dx.doi.org/10.13140/RG.2.2.23147.35362

УДК 004.832.2, 004.023

Методи розв’язання задачі управління запасами на основі нейро-асоціативного навчання і навчання з підкріпленням

Євген Федоров
професор, д-р техн. наук
ORCID: 0000-0003-3841-7373
y.fedorov@chdtu.edu.ua

Черкаський державний технологічний університет

Тетяна Уткіна
доцент, канд. техн. наук
ORCID: 0000-0002-6614-4133
t.utkina@chdtu.edu.ua

Черкаський державний технологічний університет

Ольга Нечипоренко
доцент, канд. техн. наук
ORCID: 0000-0002-3954-3796
o.nechyporenko@chdtu.edu.ua

Черкаський державний технологічний університет

Ключові слова:
нейро-асоціативне навчання;
навчання з підкріпленням;
управління запасами;
обмежена машина Коші;
метод Q-навчання;
метод SARSA.

Сьогодні актуальним завданням є розробка інтелектуальних методів, спрямованих на розв’язання задач управління запасами. Багато сучасних компаній для вдосконалення та оптимізації своїх бізнес-процесів використовують технологію теорії обмежень, яка забезпечує динамічне управління буфером запасів і використовується для управління ланцюгами поставок. Метою роботи є підвищення ефективності управління запасами за допомогою нейро-асоціативного навчання на основі обмеженої машини Коші та навчання з підкріпленням на основі Q-навчання та SARSA. Для досягнення поставленої мети створено метод на основі обмеженої машини Коші для управління буфером запасів, метод на основі Q-навчання та метод на основі SARSA для загальних задач управління запасами. Запропонована нейромережева модель обмеженої машини Коші має гетероасоціативну пам’ять без обмежень на ємність і забезпечує високу точність управління буфером запасів. Модель використовує розподіл Коші, що покращує збіжність методу параметричної ідентифікації, порівняно з традиційною обмежено машиною Больцмана. На відміну від повної машини Коші, обмежена машина Коші дає змогу працювати з більшим обсягом пам'яті. Модифікація методів Q-навчання та SARSA за рахунок динамічних параметрів дає змогу підвищити швидкість навчання за заданого рівня середньоквадратичної помилки. Проведені обчислювальні експерименти показали, що керування важливістю нагороди, параметром швидкості навчання, параметром для ε-жадібного підходу для методів Q-навчання та SARSA дає змогу на початкових стадіях зробити пошук рішення більш глобальним, а на заключних стадіях зробити пошук рішення більш локальним. Запропоновані методи дають змогу розширити сферу застосування нейро-асоціативного навчання та навчання з підкріпленням, що

підтверджується їх адаптацією для задач управління запасами та сприяє підвищенню ефективності інтелектуальних комп'ютерних систем загального і спеціального призначення. Перспективами подальших досліджень є застосування запропонованих методів для інших задач прийняття рішень, зокрема й у області штучного інтелекту.

DOI: 10.31558/2786-9482.2024.1.5

Вступ

Сьогодні актуальним завданням є розробка методів, спрямованих на вирішення задач управління запасами, які використовуються в інтелектуальних комп'ютерних системах загального та спеціального призначення. Все більше компаній прагнуть вдосконалювати та оптимізувати свої бізнес-процеси на основі впровадження технології теорії обмежень, яка забезпечує динамічне управління буфером запасів і використовується для управління ланцюгами поставок [1–4]. Внаслідок цього істотно зростає актуальність розробки методів оптимізації для технології теорії обмежень. Методи оптимізації, що знаходять точний розв'язок, мають високу обчислювальну складність. Методи оптимізації, що знаходять наближений розв'язок за допомогою спрямованого пошуку, часто потрапляють у локальний екстремум. Методи випадкового пошуку не гарантують збіжності. Через це виникає проблема недостатньої ефективності методів оптимізації, яка потребує вирішення. Для прискореного знаходження розв'язку для задач управління запасами та зниження ймовірності попадання в локальний екстремум використовуються нейро-асоціативне навчання та навчання з підкріпленням.

Метою роботи є підвищення ефективності пошуку розв'язків задач управління запасами за допомогою нейро-асоціативного навчання та навчання з підкріпленням. Для досягнення поставленої мети необхідно вирішити такі завдання:

1) створити нейромережеву модель управління буфером запасів;
2) запропонувати критерій ефективності нейромережевої моделі управління буфером запасів;
3) створити метод параметричної ідентифікації нейромережевої моделі управління буфером запасів;
4) запропонувати критерій ефективності управління запасами;
5) створити метод на основі Q-навчання для задач управління запасами;
6) створити метод на основі SARSA для задач управління запасами;
7) провести чисельне дослідження запропонованих методів.

Постановка проблеми

Нехай для моделі управління буфером запасів задано навчальну вибірку $S = \{(\mathbf{x}_m^{in}, \mathbf{d}_m^{out})\}$, $m \in \overline{1, M}$, де $\mathbf{x}_m^{in}$ – m-й вектор ознак, $\mathbf{d}_m^{out}$ – m-й вектор виду дії, яка змінює розмір буферу запасів. Тоді проблему підвищення точності управління буфером запасів за моделлю обмеженої машини Коші (RCM-моделлю) запишемо так: $g(\mathbf{x}^{in}, \mathbf{w})$, де $\mathbf{x}^{in}$ – вектор ознак,

$\mathbf{w}$ – вектор параметрів. Потрібно підібрати такий вектор параметрів $\mathbf{w}^*$, який забезпечує мінімум критерію: $F = \frac{1}{M}\sum_{m=1}^{M}(g(\mathbf{x}_m^{in}, \mathbf{w}^*) - \mathbf{d}_m^{out})^2 \to \min$.

Проблема підвищення ефективності вирішення задач управління запасами на основі методів навчання з підкріпленням (Q-навчання та SARSA) представляється як проблема знаходження такого розв'язку x^*, щоб $F(x^*) \to \min$, де x – вектор кількості закуповуваного товару у постачальника, а $F(\cdot)$ – цільова функція, яка пов'язана з витратами на закупівлю та зберігання товарів.

З погляду економічного зиску підвищення ефективності вирішення задач управління запасами дає змогу зменшити витрати від дефіциту товару та витрати від зберігання товару.

Огляд літератури

Серед конекціоністських методів, що використовуються для управління різними об'єктами [5–8], важливу роль відіграють методи на основі асоціативних нейромереж. Методи нейро-асоціативного навчання мають один або більше з таких недоліків: 1) не мають гетероасоціативної пам'яті [9–11]; 2) не працюють із дійсними даними [12–14]; 3) не мають високої ємності асоціативної пам'яті [15–17]; 4) не мають високої точності [18–20]; 5) мають високу обчислювальну складність [21–23]. Через це постає завдання побудови ефективних методів нейро-асоціативного навчання.

Методи навчання з підкріпленням мають один або більше з таких недоліків: 1) є лише абстрактний опис методу чи опис методу орієнтовано на рішення лише певного завдання [24, 25]; 2) не гарантується збіжність методу [26, 27]; 3) не враховується вплив номера ітерації на процес пошуку рішення [28, 29]; 4) відсутня можливість вирішувати завдання умовної оптимізації [30]; 5) недостатня точність методу [31, 32]; 6) не автоматизовано процедуру визначення значень параметрів [33]. Через це постає завдання побудови ефективних методів навчання з підкріпленням.

Метод нейро-асоціативного навчання для задач управління буфером запасів

Сформуємо вхідні та вихідні змінні. Вхідними змінними обрано:

x_1 – поточний обсяг запасу;

x_2 – час знаходження у червоній зоні буфера запасів;

x_3 – час знаходження у зеленій зоні буфера запасів.

Вихідною змінною y обрано номер виду дії (збільшити, зменшити, не міняти), що змінює розмір буфера запасів.

Вхідні та вихідні змінні подаються у бінарному вигляді.

Структурна схема нейромережевої моделі управління буфером запасів у формі RCM-моделі зображена на рис. 1. Вона являє собою рекурентну нейронну мережу з одним видимим та одним прихованим шарами.

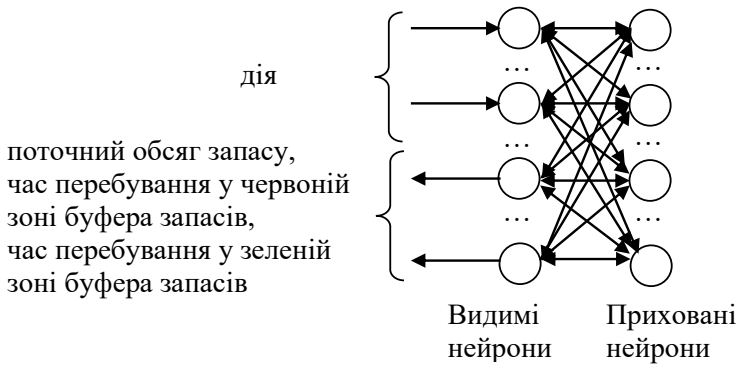

Рисунок 1. Структурна схема RCM-моделі

Компонентами RCM є приховані стохастичні нейрони, стан яких описується на основі розподілу Бернуллі у вигляді:

$$x_j = \begin{cases} 1, & \text{з ймовірністю } P_j \\ 0, & \text{з ймовірністю } 1 - P_j \end{cases}.$$

Ймовірність переходу j-го стохастичного нейрона у стан 1 розраховується так:

$$P_j = \frac{1}{2} + \frac{1}{\pi} \arctan\left(\Delta E_j\right),$$

де ΔE_j – збільшення енергії нейронної мережі внаслідок зміни стану j-го стохастичного нейрона з 0 на 1.

Переваги RCM-моделі:

1) на відміну від більшості нейронних мереж, має гетероасоціативну пам’ять;
2) на відміну від машини Больцмана, не має обмежень на ємність пам’яті;
3) на відміну від машини Больцмана, має меншу обчислювальну складність.

Визначимо нейромережеву модель управління буфером запасів.

Позитивна фаза (кроки 1–2)

1. Ініціалізація стану видимих вхідних нейронів, які відповідають вхідним змінним x_1, x_2 та x_3 у бінарному вигляді $\mathbf{x1}^{in} = \mathbf{x}^{in}$.

2. Обчислення стану прихованих нейронів:

$$P_j = \frac{1}{2} + \frac{1}{\pi} \arctan\left(b_j^h + \sum_{i=1}^{N^{in}} w_{ij}^{in-h} x1_i^{in} \right), \; j \in \overline{1, N^h};$$

$$x1_j^h = \begin{cases} 1, & P_j \geq U(0,1) \\ 0, & P_j < U(0,1) \end{cases}, \; j \in \overline{1, N^h},$$

де $U(0,1)$ – функція, що повертає рівномірно розподілене випадкове число в діапазоні [0, 1].

Негативна фаза (крок 3)

3. Обчислення стану видимих вихідних нейронів, які відповідають виду дії у бінарному вигляді:

$$P_j = \frac{1}{2} + \frac{1}{\pi}\arctan\left(b_j^{out} + \sum_{i=1}^{N^h} w_{ij}^{out-h} x1_i^h\right),\ j \in \overline{1, N^{out}};$$

$$x2_j^{out} = \begin{cases} 1, & P_j \geq U(0,1) \\ 0, & P_j < U(0,1) \end{cases},\ j \in \overline{1, N^{out}},$$

де b_j^h – поріг для j-го нейрона прихованого шару;

b_j^{out} – поріг для j-го нейрона видимого вихідного шару;

w_{ij}^{in-h} – вага зв'язку від i-го нейрона у видимому вхідному шарі до j-го нейрона прихованого шару;

w_{ij}^{out-h} – вага зв'язку від i-го нейрона у видимому вихідному шарі до j-го нейрона прихованого шару;

N^h – кількість нейронів у прихованому шарі;

N^{in} – кількість нейронів у видимому вхідному шарі;

N^{out} – кількість нейронів у видимому вихідному шарі.

Виберемо критерій ефективності нейромережевої моделі управління буфером запасів. Відповідно до цільової функції навчання RCM-моделі у знаходженні таких значень вектора параметрів $\mathbf{w} = (w_{11}^{in-h}, ..., w_{N^{in}N^h}^{in-h}, w_{11}^{out-h}, ..., w_{N^{out}N^h}^{out-h})$, які мінімізують середньоквадратичну помилку на вибірці даних:

$$F = \frac{1}{M(N^{in} + N^{out})} \sum_{m=1}^{M} \left\| \mathbf{x}2_m^{out} - \mathbf{d}_m^{out} \right\|^2 \to \min_{\mathbf{w}},$$

де $\mathbf{x}2_m^{out}$ – m-й оціночний вектор виду дії за моделлю;

$\mathbf{d}_m^{out}$ – m-й вектор виду дії.

Запропонуємо метод параметричної ідентифікації нейромережевої моделі управління буфером запасів на основі алгоритму CD-1 (one-step contrastive divergence). Метод параметричної ідентифікації нейромережевої моделі управління буфером запасів з урахуванням алгоритму CD-1 (рис. 2) складається з восьми блоків.

1. Ініціалізація

Номер ітерації навчання $n = 1$: ініціалізація зсувів (порогів) $b_i^{out}(n)$, $i \in \overline{1, N^{out}}$, $b_j^h(n)$, $j \in \overline{1, N^h}$, і ваг $w_{ij}^{in-h}(n)$, $i \in \overline{1, N^{in}}$, $j \in \overline{1, N^h}$, $w_{ij}^{out-h}(n)$, $i \in \overline{1, N^{out}}$, $j \in \overline{1, N^h}$, $w_{ii}^{in-h}(n) = 0$, $w_{ii}^{out-h}(n) = 0$, $w_{ij}^{in-h}(n) = w_{ji}^{in-h}(n)$, $w_{ij}^{out-h}(n) = w_{ji}^{out-h}(n)$ за рівномірним розподілом на інтервалі (0, 1) або [–0.5, 0.5].

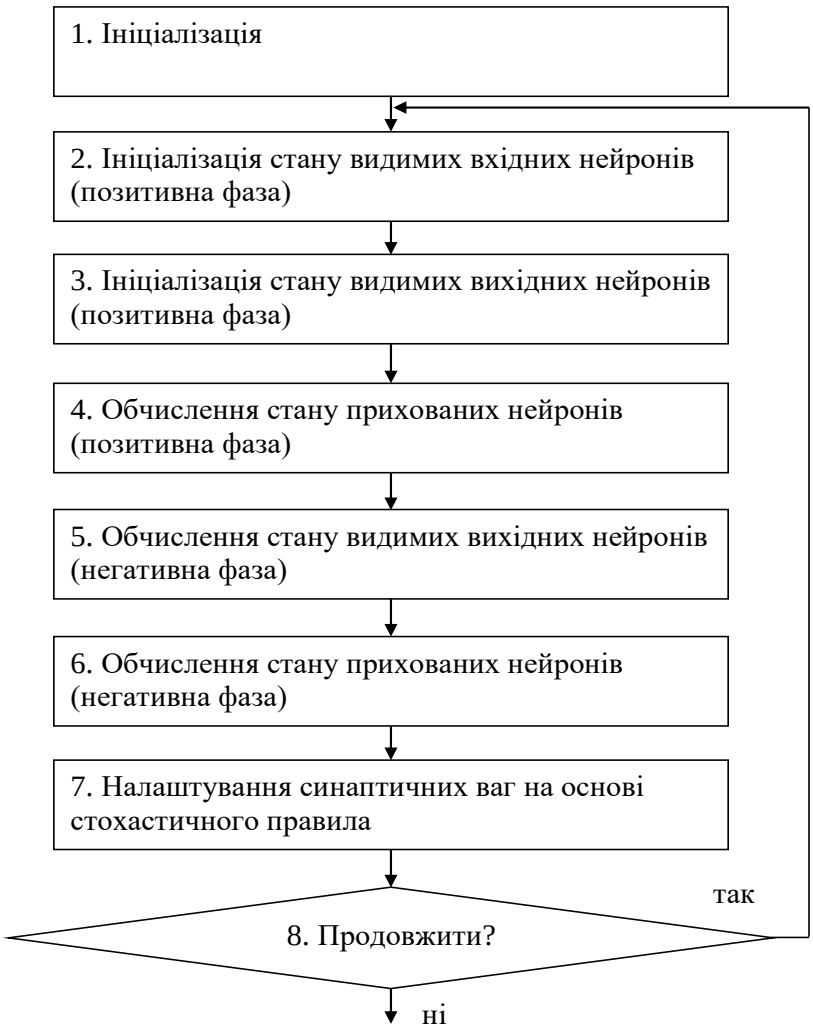

Рисунок 2. Послідовність процедур методу параметричної ідентифікації нейромережевої моделі управління буфером запасів на основі CD-1

Задається навчальна вибірка $\{(\mathbf{x}_m^{in}, \mathbf{x}_m^{out}) \mid \mathbf{x}_m^{in} \in (0,1)^{N^{in}}, \mathbf{x}_m^{out} \in (0,1)^{N^{out}}\}$, $m \in \overline{1,M}$, де $\mathbf{x}_m^{in}$ – m-й вхідний вектор зі значеннями у бінарному вигляді, $\mathbf{x}_m^{out}$ – вектор виду дії в бінарному вигляді, M – розмір навчальної вибірки.

Позитивна фаза (кроки 2–4)

2. Ініціалізація стану видимих вхідних нейронів, які відповідають вхідним змінним x_1, x_2 та x_3 у бінарному вигляді:

$$\mathbf{x}1_m^{in} = \mathbf{x}_m^{in},\ m \in \overline{1,M}.$$

3. Ініціалізація стану видимих вихідних нейронів, які відповідають виду дії у бінарному вигляді:

$$\mathbf{x}1_m^{out} = \mathbf{x}_m^{out},\ m \in \overline{1,M}.$$

4. Обчислення стану прихованих нейронів:

$$P_{mj} = \frac{1}{2} + \frac{1}{\pi}\arctan\left(b_j^h(n) + \sum_{i=1}^{N^{in}} w_{ij}^{in-h}(n) x1_{mi}^{in} + \sum_{i=1}^{N^{out}} w_{ij}^{out-h}(n) x1_{mi}^{out} \right),\ m \in \overline{1, M};$$

$$x1_{mj}^h = \begin{cases} 1, & P_{mj} \geq U(0,1) \\ 0, & P_{mj} < U(0,1) \end{cases},\ m \in \overline{1, M},\ j \in \overline{1, N^h}.$$

Негативна фаза (кроки 5–6)

5. Обчислення стану видимих вихідних нейронів, які відповідають дії у бінарному вигляді:

$$\mu_{mj}^{out} = b_j^{out}(n) + \sigma_j^{out} \sum_{i=1}^{N^h} w_{ij}^{out-h}(n) x1_{mi}^h,\ m \in \overline{1, M},\ j \in \overline{1, N^{out}};$$

$$x2_{mj}^{in} = \mu_{mj}^{in} + \sigma_j^{in} N(0,1),\ m \in \overline{1, M},\ j \in \overline{1, N^{out}}.$$

6. Обчислення стану прихованих нейронів:

$$P_{mj} = \frac{1}{2} + \frac{1}{\pi}\arctan\left(b_j^h(n) + \sum_{i=1}^{N^{in}} w_{ij}^{in-h}(n) x1_{mi}^{in} + \sum_{i=1}^{N^{out}} w_{ij}^{out-h}(n) x2_{mi}^{out} \right),\ m \in \overline{1, M};$$

$$x2_{mj}^h = \begin{cases} 1, & P_{mj} \geq U(0,1) \\ 0, & P_{mj} < U(0,1) \end{cases},\ m \in \overline{1, M},\ j \in \overline{1, N^h}.$$

7. Налаштування порогів та синаптичних ваг на основі стохастичного правила:

$$b_i^{out}(n+1) = b_i^{out}(n) + \eta\left(\frac{1}{M}\sum_{m=1}^{M} x1_{mi}^{out} - \frac{1}{M}\sum_{m=1}^{M} x2_{mi}^{out} \right),\ i \in \overline{1, N^{out}};$$

$$b_i^h(n+1) = b_i^h(n) + \eta\left(\frac{1}{M}\sum_{m=1}^{M} x1_{mi}^h - \frac{1}{M}\sum_{m=1}^{M} x2_{mi}^h \right),\ i \in \overline{1, N^h};$$

$$\rho_{ij}^+ = \frac{1}{M}\sum_{m=1}^{M} x1_{mi}^{in} x1_{mj}^h,\ i \in \overline{1, N^{in}},\ j \in \overline{1, N^h};$$

$$\rho_{ij}^- = \frac{1}{M}\sum_{m=1}^{M} x1_{mi}^{in} x2_{mj}^h,\ i \in \overline{1, N^{in}},\ j \in \overline{1, N^h};$$

$$w_{ij}^{in-h}(n+1) = w_{ij}^{in-h}(n) + \eta(\rho_{ij}^+ - \rho_{ij}^-),\ i \in \overline{1, N^{in}},\ j \in \overline{1, N^h};$$

$$\rho_{ij}^{+} = \frac{1}{M}\sum_{m=1}^{M} x1_{mi}^{out}\, x1_{mj}^{h}\,,\ i \in \overline{1, N^{out}}\,,\ j \in \overline{1, N^{h}}\,;$$

$$\rho_{ij}^{-} = \frac{1}{M}\sum_{m=1}^{M} x2_{mi}^{out}\, x2_{mj}^{h}\,,\ i \in \overline{1, N^{out}}\,,\ j \in \overline{1, N^{h}}\,;$$

$$w_{ij}^{out-h}(n+1) = w_{ij}^{out-h}(n) + \eta(\rho_{ij}^{+} - \rho_{ij}^{-})\,,\ i \in \overline{1, N^{out}}\,,\ j \in \overline{1, N^{h}}\,.$$

8. Перевірка умови завершення:

якщо $\dfrac{1}{M \cdot N^{out}}\sum_{m=1}^{M}\sum_{i=1}^{N^{out}} | x1_{mi}^{out} - x2_{mi}^{out} | > \varepsilon$, тоді $n = n+1$ і перехід до 2.

Методи навчання з підкріпленням для задач управління запасами

Визначимо цільову функцію для задач управління запасами як добуток двох функцій:

$$F(x,z) = F1(x,z) + F2(x,z) \to \min_{x}\,;$$

$$F1(x,z) = \sum_{m=1}^{M} w1 \cdot \max\left(0, z^{\min} - \left(x_m + z_{m-1} - D_m\right)\right);$$

$$F2(x,z) = \sum_{m=1}^{M} w2 \cdot \max\left(0, x_m + z_{m-1} - D_m - z^{\max}\right),$$

$$z_m = x_m + z_{m-1} - D_m\,,$$

де $F1(\cdot)$ – витрати від дефіциту товару,

$F2(\cdot)$ – витрати від зберігання товару,

$w1$ – прибуток від продажу однієї одиниці товару;

$w2$ – витрати на зберігання однієї одиниці товару;

x_m – кількість товару, що закуповується у постачальника протягом m-го етапу;

z_m – кількість запасів товару в кінці m-го етапу;

z_0 – вихідна кількість запасів товару;

$z^{\min}, z^{\max}$ – мінімально допустимі та максимально допустимі запаси товару в кінці кожного етапу;

D_m – кількість товару, що продається протягом m-го етапу;

M – кількість етапів.

Кількість закуповуваного товару в постачальника обмежено в такий спосіб:

$$x^{\min} \le x_m \le x^{\max},\ m \in \overline{1,M},$$

де $x^{\min}, x^{\max}$ – мінімальна і максимальна кількість товару, що закуповується у постачальника протягом кожного етапу.

Пропонований метод на основі Q-навчання з динамічними параметрами складається з 13 кроків.

1. Ініціалізація

1.1. Задаємо параметри системи управління запасами – N, M, $w1$, $w2$, $x^{\min}, x^{\max}$, $z^{\min}, z^{\max}$, z_0, D_m, $m \in \overline{1,M}$ а також параметри алгоритму: $\rho^{\min}, \rho^{\max}$ для управління швидкістю навчання, $0 < \rho^{\min} < \rho^{\max} < 1$; $\varepsilon^{\min}, \varepsilon^{\max}$ для ε-жадібного підходу, $0 < \varepsilon^{\min} < \varepsilon^{\max} < 1$; $\theta^{\min}, \theta^{\max}$ для визначення важливості майбутньої винагороди, $0 < \theta^{\min} < \theta^{\max} < 1$.

1.2. Ініціалізуємо таблицю винагороди $Q = [Q(i, j)]$, $Q(i, j) = 0$, $i, j \in \overline{1,M}$.

2. Встановлюємо номер ітерації $n = 1$.

3. Обчислюємо параметри:

$$\rho(n) = \rho^{\max} - (\rho^{\max} - \rho^{\min}) \frac{n-1}{N-1};$$

$$\varepsilon(n) = \varepsilon^{\max} - (\varepsilon^{\max} - \varepsilon^{\min}) \frac{n-1}{N-1};$$

$$\theta(n) = \theta^{\min} + (\theta^{\max} - \theta^{\min}) \frac{n-1}{N-1}.$$

4. Задаємо початковий стан $s = z_0$.

5. Задаємо початковий номер етапу $m = 1$.

6. Вибирається дія a (кількість товару, що закуповується у постачальника), за якою виходимо зі стану s за ε-жадібним підходом. Якщо $U(0,1) < \varepsilon(n)$, тоді обираємо дію a випадковим способом із множини дозволених дій $\{x^{\min}, x^{\max}\}$, інакше обираємо дію a, таку, щоб $a = \arg\max_b Q(s, b)$, $b \in \{x^{\min}, x^{\max}\}$. Обрана дія a стає новим компонентом вектора кількості товару, що закуповується у постачальника, тобто $x_m = a$.

7. Якщо відбувається останній етап, тобто $m = M$, тоді переходимо до кроку 13, інакше $m = m + 1$.

8. Обчислюємо елемент таблиці поточної винагороди $R(s, a)$:

$$R(s,a) = -\left(w1 \cdot \max\left(0, z^{\min} - \left(a + s - D_m\right)\right) + w2 \cdot \max\left(0, a + s - D_m - z^{\max}\right)\right).$$

9. Оновлюємо запаси товару: $e = a + s - D_m$.

10. Обчислюємо елемент таблиці винагороди $Q(s,a)$:

$$Q(s,a) = (1-\rho(n))Q(s,a) + \rho(n)\left(R(s,a) + \theta(n) \cdot \max_b Q(e,b) \right),\ b \in \{x^{\min}, \ldots, x^{\max}\}.$$

11. Встановлюємо новий поточний стан $s = e$ і переходимо до кроку 6.

12. Якщо найкраще значення цільової функції на поточній ітерації менше від кращого значення за попередніми ітераціям, тобто $F(x) < F(x^*)$, то оновити вектор кількості товару, що закуповується у постачальника: $x^* = x$.

13. Якщо $n < N$, то перейти до кроку 3, інакше завершити роботу.

Пропонований метод на основі SARSA з динамічними параметрами збігається з попереднім методом на основі Q-навчання з динамічними параметрами за всіма кроками, окрім дев'ятого. Це крок реалізується так:

9. Вибирається дія c (кількість товару, що закуповується у постачальника), за якою потрібно перемістити зі стану e, використовуючи ε-жадібний підхід (якщо $U(0,1) < \varepsilon(n)$, то вибрати дію c випадковим способом із множини дозволених дій $\{x^{\min}, \ldots, x^{\max}\}$, інакше вибрати дію c як $c = \arg\max_b Q(e,b)$, $b \in \{x^{\min}, \ldots, x^{\max}\}$). Вибрана дія c стає новим компонентом вектора кількості товару, що закуповується у постачальника, тобто $x_m = c$. Обчислюється елемент таблиці винагороди за формулою:

$$Q(s,a) = (1-\rho(n))\, Q(s,a) + \rho(n)\left(R(s,a) + \theta(n)\, Q(e,c) \right).$$

Експерименти

Чисельне дослідження запропонованих методів проведемо з використанням пакету Python. Для методів Q-навчання та SARSA встановимо такі параметри навчання $\rho^{\min} = 0.1, \rho^{\max} = 0.9$ $\theta^{\min} = 0.1, \theta^{\max} = 0.9$. Експерименти проведемо на основі даних логістичної компанії "Ekol Ukraine"– це інтегрована логістична компанія, яка надає інтелектуальні клієнтоцентричні рішення в управлінні ланцюгом постачання, спрямовані на масштабування та розвиток бізнесу. Обсяг датасету становить 1000 реалізацій комп'ютерної техніки протягом одного року.

Залежність параметра $\theta(n)$ задамо зростаючою: $\theta(n) = \theta^{\min} + (\theta^{\max} - \theta^{\min})\frac{n-1}{N-1}$. Залежності параметрів $\rho(n)$ та $\varepsilon(n)$ задамо спадними $\rho(n) = \rho^{\max} - (\rho^{\max} - \rho^{\min})\frac{n-1}{N-1}$ та $\varepsilon(n) = \varepsilon^{\max} - (\varepsilon^{\max} - \varepsilon^{\min})\frac{n-1}{N-1}$. Графіки залежностей цих параметрів представлено на рис. 3. Зміна параметра $\theta(n)$ початкових ітераціях зменшує важливість нагороди, що робить пошук рішення більш глобальним, а на заключних ітераціях підвищує важливість нагороди, що робить пошук рішення більш локальним. Зміна параметра $\rho(n)$ на початкових ітераціях підвищує швидкість навчання, що робить пошук рішення більш глобальним, а на заключних

ітераціях зменшує швидкість навчання, що робить пошук рішення більш локальним. Зміна параметра $\varepsilon(n)$ на початкових ітераціях підвищує ймовірність випадкового вибору стану, що робить пошук рішення більш глобальним, а на заключних ітераціях зменшує ймовірність випадкового вибору стану, що робить пошук рішення більш локальним.

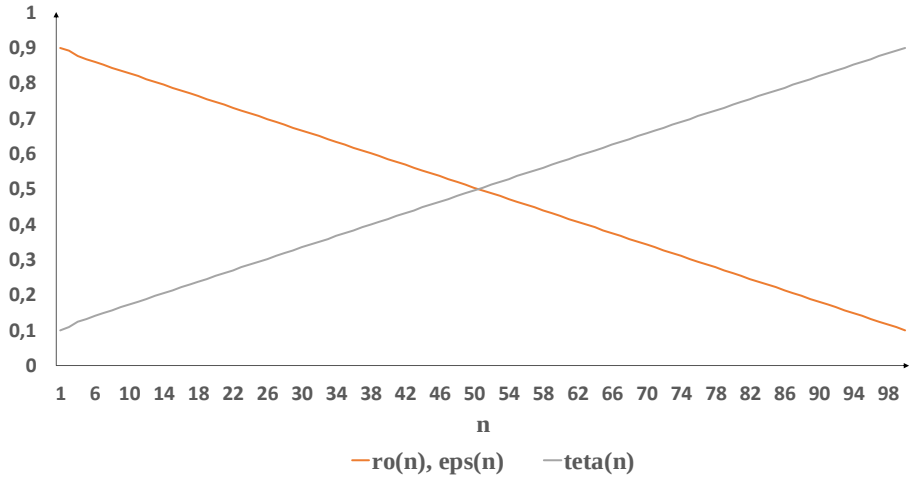

Рисунок 3. Залежність параметрів $\theta(n)$, $\rho(n)$ та $\varepsilon(n)$ від номера ітерації n

Результати навчання по тестовій вибірці запропонованої нейромережевої RCM-моделі з нейромережевою моделлю багатошарового персептрона (MLP) з двома шарами на основі критерію середньоквадратичної помилки представлено в табл. 1. З таблиці видно, що використання RCM зменшує середньоквадратичну помилку, чим підвищує точність управління буфером запасів. Порівняння по тестовій вибірці запропонованого методу Q-навчання з динамічними параметрами та з традиційним методом Q-навчання за середньоквадратичною помилкою і за кількістю ітерацій для вирішення задачі управління запасами здійснено в табл. 2. Аналогічні результати отримано для запропонованого методу SARSA з динамічними параметрами та з традиційним методом SARSA (табл. 3). З табл. 2–3 видно, що модифікація методів Q-навчання та SARSA за рахунок динамічних параметрів збільшує швидкість навчання за збереження середньоквадратичної помилки методу.

Таблиця 1. Порівняння нейромережевих моделей RCM та MLP

Середньоквадратична помилка нейромережевої моделі	
RCM	MLP
0.05	0.1

Таблиця 2. Порівняння запропонованого методу Q-навчання з традиційним

Середньоквадратична помилка методу		Кількість ітерацій	
запропонованого	традиційного	запропонованого	традиційного
0.05	0.05	300	2000

Таблиця 3. Порівняння запропонованого методу SARSA з традиційним

Середньоквадратична помилка методу		Кількість ітерацій	
запропонованого	традиційного	запропонованого	традиційного
0.05	0.05	300	2000

Висновки

Запропонована нейромережева модель обмеженої машини Коші має гетероасоціативну пам’ять та не має обмежень на ємність пам’яті. Вона використовує розподіл Коші, що підвищує збіжність параметричної ідентифікації та забезпечує високу точність управління буфером запасів.

Запропонована модифікація методів Q-навчання та SARSA за рахунок використання динамічних параметрів у правилі оновлення таблиці винагороди дає змогу підвищити швидкість навчання. Запропоновані модифікація завдяки дослідженню всього простору пошуку на початкових ітераціях та спрямованості пошуку на заключних ітераціях забезпечують високу точність розв’язання задачі управління запасами.

Запропоновані методи розширюють сферу застосування нейро-асоціативного навчання та навчання з підкріпленням, що підтверджується їх адаптацією для задач управління запасами. Це сприятиме підвищенню ефективності інтелектуальних комп’ютерних систем загального та спеціального призначення. Перспективами подальших досліджень є масштабування запропонованих методів на широкий клас задач штучного інтелекту.

Література

1. Mayo-Alvarez, L., Del-Aguila-Arcentales, S., Alvarez-Risco, A., Chandra Sekar, M., Davies, N. M., & Yáñez, J. A. (2024). Innovation by integration of Drum-Buffer-Rope (DBR) method with Scrum-Kanban and use of Monte Carlo simulation for maximizing throughput in agile project management. *Journal of Open Innovation: Technology, Market, and Complexity*, *10*(1). https://doi.org/10.1016/j.joitmc.2024.100228
2. Melendez, J. R., Zoghbe Nuñez, Y. A., Malvacias Escalona, A. M., Almeida, G. A., & Layana Ruiz, J. (2018). Theory of constraints: A systematic review from the management context. *Espacios*, *39*(48).
3. Stopka, O., Zitrický, V., Ľupták, V., & Stopková, M. (2023). Application of specific tools of the theory of constraints – a case study. *Cognitive Sustainability*, *2*(1). https://doi.org/10.55343/cogsust.48
4. Bart, A., Delahaye, B., Fournier, P., Lime, D., Monfroy, É., & Truchet, C. (2018). Reachability in parametric interval Markov chains using constraints. *Theoretical Computer Science*, *747*, 48–74. https://doi.org/10.1016/j.tcs.2018.06.016
5. Haykin, S. (2008). Neural Networks and Learning Machines. *Pearson Prentice Hall New Jersey USA 936 pLinks* (vol. 3, p. 906). https://doi.org/978-0131471399
6. Shlomchak, G., Shvachych, G., Moroz, B., Fedorov, E., & Kozenkov, D. (2019). Automated control of temperature regimes of alloyed steel products based on multiprocessors computing systems. *Metalurgija*, *58*(3–4), 299–302.
7. Shvachych, G. G., Ivaschenko, O. V., Busygin, V. V., & Fedorov, Y. Y. (2018). Parallel computational algorithms in thermal processes in metallurgy and mining. *Naukovyi Visnyk Natsionalnoho Hirnychoho Universytetu*, (4), 129–137. https://doi.org/10.29202/nvngu/2018-4/19
8. Fedorov, E., Utkina, T., Nechyporenko, O., & Korpan, Y. (2020). Development of technique for face detection in image based on binarization, scaling and segmentation methods. *Eastern-*

European Journal of Enterprise Technologies, *1*(9–103), 23–31. https://doi.org/10.15587/1729-4061.2020.195369

9. Singh, U. P., Jain, S., Tiwari, A., & Singh, R. K. (2019). Gradient evolution-based counter propagation network for approximation of noncanonical system. *Soft Computing*, *23*(13), 4955-4967. https://doi.org/10.1007/s00500-018-3160-7
10. Sonika, Pratap, A., Chauhan, M., & Dixit, A. (2017). New technique for detecting fraudulent transactions using hybrid network consisting of full-counter propagation network and probabilistic network. In *Proceeding – IEEE International Conference on Computing, Communication and Automation, ICCCA 2016* (pp. 177–182). Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/CCAA.2016.7813713
11. Baggenstoss, P. M. (2019). Applications of projected belief networks (PBN). In *European Signal Processing Conference* (Vol. 2019 – September). European Signal Processing Conference, EUSIPCO. https://doi.org/10.23919/EUSIPCO.2019.8902708
12. Baggenstoss, P. M. (2019). On the duality between belief networks and feed-forward neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, *30*(1), 190–200. https://doi.org/10.1109/TNNLS.2018.2836662
13. Sountsov, P., & Miller, P. (2015). Spiking neuron network Helmholtz machine. *Frontiers in Computational Neuroscience*, *9*(APR). https://doi.org/10.3389/fncom.2015.00046
14. Kohonen, T. (2012). *Self-Organization and Associative Memory*, (3rd ed.). Berlin; New York: Springer-Verlag. 311 p. https://doi.org/10.1007/978-3-642-88163-3
15. Kohonen, T. (2013). Essentials of the self-organizing map. *Neural Networks*, *37*, 52–65. https://doi.org/10.1016/j.neunet.2012.09.018
16. Lobo, R. A., & Valle, M. E. (2020). Ensemble of binary classifiers combined using recurrent correlation associative memories. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (Vol. 12320 LNAI, pp. 442–455). Springer Science and Business Media Deutschland GmbH. https://doi.org/10.1007/978-3-030-61380-8_30
17. Kobayashi, M. (2017). Quaternionic Hopfield neural networks with twin-multistate activation function. *Neurocomputing*, *267*, 304–310. https://doi.org/10.1016/j.neucom.2017.06.013
18. Du, K. L., & Swamy, M. N. S. (2014). *Neural Networks and Statistical Learning* (Vol. 9781447155713, pp. 1–824). Springer-Verlag London Ltd. https://doi.org/10.1007/978-1-4471-5571-3
19. Javidmanesh, E. (2017). Global stability and bifurcation in delayed bidirectional associative memory neural networks with an arbitrary number of neurons. *Journal of Dynamic Systems, Measurement and Control, Transactions of the ASME*, *139*(8). https://doi.org/10.1115/1.4036229
20. Park, Y. (2010). Optimal and robust design of brain-state-in-a-box neural associative memories. *Neural Networks*, *23*(2), 210–218. https://doi.org/10.1016/j.neunet.2009.10.008
21. Khristodulo, O. I., Makhmutov, A. A., & Sazonova, T. V. (2017). Use algorithm based at hamming neural network method for natural objects classification. In *Procedia Computer Science* (Vol. 103, pp. 388–395). Elsevier B.V. https://doi.org/10.1016/j.procs.2017.01.126
22. Fischer, A., & Igel, C. (2014). Training restricted Boltzmann machines: An introduction. *Pattern Recognition*, *47*(1), 25–39. https://doi.org/10.1016/j.patcog.2013.05.025

23. Wang, Q., Gao, X., Wan, K., Li, F., & Hu, Z. (2020). A novel restricted Boltzmann machine training algorithm with fast Gibbs sampling policy. *Mathematical Problems in Engineering*, *2020*. https://doi.org/10.1155/2020/4206457
24. Bertsekas, D. P. (2019). *Reinforcement Learning and Optimal Control*. Belmont, MA: Athena Scientific.
25. François-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G., & Pineau, J. (2018). An introduction to deep reinforcement learning. *Foundations and Trends in Machine Learning*, *11*(3–4), 219–354. https://doi.org/10.1561/2200000071
26. Goldberg, D. A., Katz-Rogozhnikov, D. A., Lu, Y., Sharma, M., & Squillante, M. S. (2016). Asymptotic optimality of constant-order policies for lost sales inventory models with large lead times. *Mathematics of Operations Research*, *41*(3), 898–913. https://doi.org/10.1287/moor.2015.0760
27. Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., & Silver, D. (2018). Rainbow: Combining improvements in deep reinforcement learning. In *32nd AAAI Conference on Artificial Intelligence, AAAI 2018* (pp. 3215–3222). AAAI Press. https://doi.org/10.1609/aaai.v32i1.11796
28. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533. https://doi.org/10.1038/nature14236
29. Graesser, L., & Keng, W. L. (2019). *Foundations of Deep Reinforcement Learning: Theory and Practice in Python*. Boston: Addison-Wesley Professional.
30. Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction*, (2nd ed.). Adaptive Computation and Machine Learning. Cambridge: The MIT Press.
31. Matta, M., Cardarilli, G. C., Di Nunzio, L., Fazzolari, R., Giardino, D., Re, M., Silvestri, F., & Spanò, S. (2019). Q-RTS: A real-time swarm intelligence based on multi-agent Q-learning. *Electronics Letters*, *55*(10), 589–591. https://doi.org/10.1049/el.2019.0244
32. Kar, S., Moura, J. M. F., & Poor, H. V. (2013). 2D-learning: A collaborative distributed strategy for multi-agent reinforcement learning through consensus + innovations. *IEEE Transactions on Signal Processing*, *61*(7), 1848–1862. https://doi.org/10.1109/TSP.2013.2241057
33. Ottoni, A. L., Nepomuceno, E. G., Oliveira, M. S., & Oliveira, D. C. (2022). Reinforcement learning for the traveling salesman problem with refueling. *Complex and Intelligent Systems*, *8*(3), 2001–2015. https://doi.org/10.1007/s40747-021-00444-4

Рукопис отримано – 15/05/2024; прийнято до публікації – 20/06/2024.

Methods of solving the problem of stock management based on neuro-associative learning and reinforcement learning

Eugene Fedorov, Tetyana Utkina, Olga Nechyporenko

Abstract

Today, the development of intelligent methods aimed at solving inventory management problems is an urgent task. Many modern companies use the technology of constraint theory to improve and optimize their business processes, which provides dynamic inventory buffer management and is used for supply chain management. The aim of the work is to improve the efficiency of inventory management through neuro-associative learning based on the constrained Cauchy machine and reinforcement learning based on Q-learning and SARSA. To achieve this goal, a method based on a constrained Cauchy machine for inventory buffer management, a method based on Q-learning, and a method based on SARSA for general inventory management tasks are created. The proposed neural network model of the constrained Cauchy machine has a hetero-associative memory with no capacity limitations and provides high accuracy for inventory buffer management. The model uses the Cauchy distribution, which improves the convergence of the parametric identification method by comparison with the traditional restricted Boltzmann machine. Compared to the full Cauchy machine, the constrained Cauchy machine allows working with a larger memory size. Modification of the Q-learning and SARSA methods by means of dynamic parameters allows to increase the learning speed at a given level of the mean square error. Computational experiments have shown that controlling the importance of the reward, the learning rate parameter, and the parameter for ε-greedy approach for the Q-learning and SARSA methods allows making the solution search more global at the initial stages and more local at the final stages. The proposed methods allow expanding the scope of neuro-associative learning and reinforcement learning, which is confirmed by their adaptation for inventory management tasks, and contributes to the efficiency of intelligent computer systems for general and special purposes. Prospects for further research are the application of the proposed methods to other decision-making tasks, including the ones in the field of artificial intelligence.

Keywords: neuro-associative learning; learning with reinforcement; inventory management; bounded Cauchy machine; Q-learning method; the SARSA method.