ISSN 2522-9818 (print)
ISSN 2524-2296 (online) *Innovative technologies and scientific solutions for industries. 2025. No. 4 (34)*

UDC 004.032.26:004.932.2:004.052 DOI: https://doi.org/10.30837/2522-9818.2025.4.018

S. DANYLENKO, K. SMELYAKOV

CONTENT-BASED IMAGE RETRIEVAL METHOD IN A MULTIDIMENSIONAL MODEL AT BIG DATA SCALE

The **subject** of the study is the method and algorithms for content-based image retrieval within the Multidimensional Cube (MDC) model. The **goal** is to develop a search method based on image descriptor vectors and an algorithm that implements this method in both sequential and parallel versions for MDC. The research **tasks** include: defining requirements for the search method; analyzing the MDC model structure and defining the approach to the search method; developing search methods and algorithms for scenarios where the model is stored in RAM or in a relational database; integrating parallel computing into the algorithm; analyzing alternative models based on multidimensional trees, graphs, hashing, inverted indexing, quantization and inverted multi-index structures; developing evaluation metrics and conducting experiments to compare the efficiency of the MDC-based method with alternative search models. **Methodology**: analytical and comparative methods for search algorithm evaluation, modeling, and experimental verification were applied. Thread-level parallelism and hardware optimization methods were used, along with comparative analysis of model efficiency (KD-tree, Locality-Sensitive Hashing, Hierarchical Navigable Small World, Inverted File with Flat Compression, Inverted Multi-Index). Statistical methods were employed to assess results using recall, search time, and model construction time metrics. Experiments were conducted with both web-sourced and synthetic image descriptors, as well as load testing to evaluate the model's throughput. **Results**: a new search method and the Wave-Search Algorithm were developed. Its parallel version achieves up to a 3x speedup. For top-10 and top-100 queries in a dataset of 1 million descriptors, MDC shows the best overall performance among the compared models based on the metrics and strong stability under load. **Conclusions**: the proposed search method and its implementation (Wave-Search Algorithm) efficiently utilize the MDC model's structure for search tasks, outperforms alternative search models in terms of effectiveness, demonstrates robustness under load, and has significant potential for further development, including the use of hardware acceleration.

Keywords: similarity search; big data; search algorithms; data structures; content-based image retrieval; parallel computing; high-performance computing; search efficiency; algorithm optimization.

Introduction

Information search is a fundamental need of humanity. Even in ancient times, when writing began to develop, there was a need to organize existing works and search for information contained in them. This led to the emergence of alphabetical indexes, catalogs, and indexes, which formed the basis of modern search systems [1].

Search engines have undergone a long period of development. Their implementation involves both classical approaches, such as Boolean, Vector Space, and Probabilistic models, and more modern ones, such as machine learning, deep learning, neural ranking models, contextual and semantic methods [2].

Search is used both in everyday tasks, such as renting accommodation, buying tickets, searching for goods, etc., and in professional fields such as medicine, education, business, e-commerce, and science. It greatly simplifies and speeds up the process of analyzing the necessary information [3].

A search engine is based on a model that uses a specific data structure. Each data structure has its own search algorithms, which are applied depending on the requirements. For example, for graphs, depending on the expected result, the classic Breadth First Search and Depth First Search algorithms can be applied. This makes search models universal and allows them to be widely used in various tasks [4].

With the growth of visual content, there is a need for effective image retrieval and search based on graphic information. Content-based image retrieval (CBIR) systems allow images to be found based on their visual characteristics, such as color, texture, and shape. Modern CBIR systems are actively used in medicine, e-commerce, and other industries. The implementation of such systems requires special data structures and search algorithms [5].

Despite significant progress in the development of CBIR systems, there are challenges related to scalability, efficiency, and search accuracy. In particular, there is the problem of effective representation of visual data in the context of Big Data and performing effective searches in these conditions. Therefore, the development of new models and search algorithms is an important task for modern research.

Сучасний стан наукових досліджень та технологій в промисловості. 2025. № 4 (34)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

Analysis of recent studies and publications

Each CBIR model has its own search methods. They are determined by the data structures underlying the model and implemented by specific search algorithms. Basically, modern models are based on: multidimensional trees, hashing, graphs, and clustering. Special adapted vector databases can also be used. The chosen approach to indexing and searching directly affects the speed and accuracy of the CBIR system. Models use graphical information representation in the form of a descriptor, which often takes the form of a normalized vector [6], and compare them using special metrics [7].

Tree structures organize descriptors hierarchically, which allows you to effectively narrow down the search area. Searching in KD-Tree is performed by recursively comparing coordinates with node boundaries, which allows you to quickly filter out unnecessary branches. Quad-tree uses recursive division of space into quadrants, checking only those that contain the requested point. R-tree checks the minimum rectangles containing objects and recursively examines subtrees for range matching [8]. In B+ Tree, the search goes through internal nodes to leaf nodes, which optimizes range queries [9]. VP-Tree compares the distance to the reference point in each node, filtering out unnecessary areas. Ball-Tree compares the distance to “balls” (nodes) and recursively filters out irrelevant parts of the space [10]. Learned index uses machine learning models to predict the location of an element, significantly speeding up the search [11].

Hash methods organize the search for similar objects in different ways. Locality-Sensitive Hashing (LSH) performs a search by comparing hash codes: the more similar the objects, the more likely their hashes will match [12]. Spherical Hashing improves this process by using hyperspheres to divide the space for more accurate grouping of similar objects [13]. FlyHash speeds up the search thanks to sparse coding, which reduces the number of comparisons [14]. In Deep Hashing, the search is performed using binary codes that simultaneously take into account the content and semantics of images [15]. Adaptive multiscale hashing performs searches based on multi-level features, which allows for better differentiation between similar objects [16].

The most relevant graph-based nearest neighbor search methods are Hierarchical Navigable Small World (HNSW) [17] and NN-Descent [18]. HNSW demonstrates high accuracy and search speed thanks to its multi-level hierarchy and greedy navigation. Its architecture allows scaling to billion-scale datasets while maintaining high performance. It has numerous modifications: bh-DBSCAN uses bidirectional HNSW to reduce redundant queries of neighboring points [19], and HNSWQ improves entry points and adapts the algorithm to work with a graphics processor [20]. On the other hand, NN-Descent is an efficient approach to building search graphs – it does not require parametric fine-tuning and scales well. It is often used in systems in various fields, including bioinformatics [21, 22].

IVFFlat (Inverted File with Flat compression) combines vector space clustering (usually via k-means) with linear scanning only in selected clusters, reducing computational costs. Modifications to IVFFlat are proposed to improve search speed and flexibility. In particular, Fast IVF optimizes cluster selection to speed up search without significant loss of accuracy [23]. Semi-supervised IVF uses auxiliary labels to improve the distribution of vectors across the index, which increases the relevance of results [24]. Reconfigurable Inverted Index allows the index to be dynamically reconfigured, maintaining efficiency even when the number or composition of vectors changes [25].

Product Quantization (PQ) provides high compression and reduces the amount of stored data [26], and its use in Inverted Multi-Index (IMI) allows the application of multiple inverted indexes, which improves search speed and quality [27]. Approaches are proposed to improve efficiency: Hierarchical Hyperbolic Product Quantization uses hyperbolic geometry to preserve hierarchical semantic relationships between images, improving search accuracy in complex semantic structures [28]; Entropy-Optimized Deep Weighted Product Quantization provides balance in the representation of code words, improving search accuracy and coding efficiency [29]; Fuzzy Norm-Explicit Product Quantization proposes the use of second-type fuzzy sets to improve search completeness without increasing computational complexity [30].

Ready-made software products that provide efficient work with vectors, in particular vector databases or additions to relational databases, such as Pgvector [31], are also very popular.

Goals and objectives

A wide selection of search models allows you to use the one that is most effective for a specific task. Since MDC is a unique search model with its own advantages,

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2025. No. 4 (34)

it requires its own search method that will utilize the features of its structure.

The purpose of this work is to develop a method for searching images in the MDC model based on image descriptor vectors and an algorithm that implements this method in sequential and parallel versions.

To achieve this goal, the following tasks must be performed:

1) formulating requirements for the search method;

2) analyzing the MDC structure and determining the approach to the search method;

3) developing a search method and algorithms when placing the model in RAM and in a relational database;

4) implementing parallel computing in the search algorithm;

5) analyzing alternative models based on: multidimensional trees, graphs, hashing, reverse index, quantization, and reverse multi-index;

6) developing metrics and conducting experiments to compare the effectiveness of the MDC model with alternative search models.

The search method must meet the following requirements:

1) effectively use workstation resources;

2) ensure high search speed and quality;

3) adapt to variations in MDC placement in memory.

The research is comprehensive, and its result is to determine the effectiveness of the MDC model in comparison with alternative models in the context of image retrieval in Big Data repositories. It also determines the conditions under which the use of MDC is preferable to other models.

Search method in MDC

In MDC, the initial descriptor vectors are reduced by decreasing the initial dimension through pairwise aggregation of vector values. This results in an additional vector of small dimension. The range of possible values of vector elements is divided into intervals depending on the distribution of vector values within the range. After that, the values of the additional vector are replaced with the indices of the interval to which they belong. Thus, based on the values of the descriptor vector elements, a multidimensional cube (MDC) is formed, in which the dimension of the additional vector specifies the number of measurements, and when the measurements intersect, the aforementioned intervals form cells (clusters). The values of the index vector determine the location of the descriptor on the interval of each dimension and in space as a whole [32].

This structure determines the approach of the search method. The relationships between the values of the vectors of different descriptors are stored as indices in the index vector. For example, if for a given vector value the corresponding measurement interval index has a value of 5, then to find descriptors with slightly smaller and slightly larger values in this measurement, it is necessary to check the MDC cells that have an index equal to 4 and 6, respectively. This operation can be performed for all measurements, going through the cells around the specified one, visually imitating waves.

The algorithm of actions in this search method in MDC boils down to the following steps:

1) process the vector of the searched image descriptor and convert it to a vector of indices.

2) Use the index vector to determine the MDC cell and perform a full search for descriptors in this cell. Compare the descriptors with the searched one based on the selected metric and calculate the difference;

3) if the search was unsuccessful, perform 1 search wave, within which increase and decrease the index values by 1 for all measurements, form cell indexes from the values of all measurements, and perform a full search for descriptors in them;

4) if the search was unsuccessful, repeat step 3 until the search is successful, the maximum number of waves is reached, or all cells are checked;

5) sort the resulting list of descriptors in ascending order of the calculated difference value and return the required number of descriptors as the search result.

Performing a search wave means checking cells that are at a certain distance in terms of measurement index values relative to the initially defined cell for the descriptor being searched for.

The search is successful when the condition for its completion is met, for example, finding a certain number of the first most similar descriptors.

The number of descriptors in the search wave is determined by the formula:

$$wc_i = (j_i)^N \times rc - (j_{i-1})^N \times rc, \quad rc = \frac{dc}{k^N}, \qquad (1)$$

where wc is the number of descriptors on the search wave, i is the wave number, starting from 1, j is an ascending sequence of natural odd numbers: [3, 5, ...], N is the number of measurements, rc is the theoretical number of descriptors in one cell, dc is the number

Сучасний стан наукових досліджень та технологій в промисловості. 2025. № 4 (34)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

of descriptors in the storage, k is the number of intervals in the measurements.

The parameters k and N are selected based on the requirements for the search system so that the value o f rc is close to the size of the search page [32].

The selected similarity metric does not affect the operation of MDC, but the quality of the search may depend on it. A wide range of metrics can be used: Manhattan, Euclidean, Quadratic Euclidean, etc. The metric that is best suited for working with a specific type of descriptor is selected. By default, the Manhattan distance is used, which is determined by the formula:

$$\mu = \sum_{i=1}^{N} \left| v_i^{(1)} - v_i^{(2)} \right|, \quad (2)$$

where μ is the difference between descriptors, i is the ordinal number of the vector element, N is the number of vector elements, $v^{(1)}, v^{(2)}$ are the vectors of the first and second descriptors.

The calculated resulting list is stored for a certain time in the search engine cache, because it usually contains more descriptors than were displayed to the user on the first page of results. This allows you to avoid performing further search waves until all the results found are displayed.

This search algorithm is called the Wave-Search Algorithm (WSA) because of the way it uses the MDC structure and is a hybrid of reverse multi-index search and exhaustive search.

An example of a search is shown in Figure 1. It shows an MDC with 2 dimensions and 10 equal intervals for each dimension

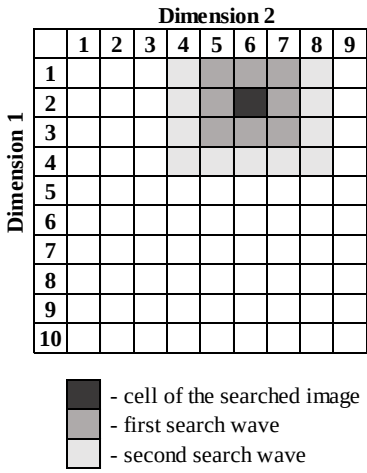

Fig. 1. Wave-Search visualization

Cell [2, 6] corresponds to the desired descriptor, and the search is initially performed in it. Next, one search wave is performed, and the indices are increased/decreased by 1 when possible. In the first wave, the following cells are checked: [1, 5], [1, 6], [1, 7], [2, 5], [2, 7], [3, 5], [3, 6], [3, 7]. In the second wave, the indexes are increased/decreased by 1 again, and so on.

Due to the peculiarities of the MDC structure, the values of the descriptor vector may be located on the border of cells, and therefore the descriptor relevant for the search may end up in a neighboring cell. And when searching only in the identified cell, it will be missed. To solve this, WSA always performs 1 search wave. Placing descriptors in cells is quite effective, and with the right parameters, each cell contains a small number of descriptors. This operation slightly increases the search time but improves the search quality.

The presented algorithm is a general implementation of the search method. For each of the MDC placements in memory, there are variations of the algorithm that technically refine some details of the implementation. There are also separate versions of algorithms that use parallel computing or can apply hardware acceleration.

Wave-Search algorithm for different ways of MDC placement in memory

When MDC is stored in RAM, the model has a "flat" structure – the index vector takes the form of a string with a separator and is then entered into a hash table, where the key is the cell index, for example "1-3-1-4", and the value for the key is a list of descriptors [32].

For such placement of MDC in memory, the following details of WSA implementation are specified:

– to form the indices of cells that fall into a specific search wave, numbers are extracted from the string representation of the cell and the specified index increase/decrease operations are performed on them. The calculated cell indexes are used to form their string representations, which are used to extract lists of descriptors from the hash table;

– the extracted lists are collected into a single resulting list. Next, for all descriptors found, the difference is calculated according to the selected metric, and the list is sorted according to its value.

When placing MDC in a relational database (DB), a "star" schema is used, in which there is a separate table for each dimension, where information about intervals is stored, a table for storing descriptors, and a linking table, in which the descriptor identifier is stored by the index vector [32].

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2025. No. 4 (34)

In this implementation of WSA, the process of extracting descriptors from the repository is transferred to the DB. This is done using an SQL query, where cell indexes are specified as tuples. An example of such a query is shown in Figure 2.

```
SELECT d.prop_1, d.prop_2, d.prop_3, d.prop_4,
       d.file_id, d.original_vector
FROM descriptors d
    INNER JOIN links l ON d.id = l.descriptor_id
WHERE l.prop_1 IN (10, 9, 11) AND
      l.prop_2 IN (18, 17, 19) AND
      l.prop_3 IN (12, 11, 13) AND
      l.prop_4 IN (7, 6, 8)
```

Fig. 2. SQL query to perform a search

The descriptors found are returned as a list. Next, for all descriptors found, the difference is calculated according to the selected metric, and the list is sorted according to its value.

Parallel computations in the Wave-Search algorithm

From the technical details of the algorithm implementation, it follows that in the WSA implementation in RAM, it is possible to simultaneously perform the process of extracting descriptors from the hash table, calculating their similarity to the search term, and sorting the resulting list.

For parallel sorting, any of the currently known algorithms implemented in the programming language used to create MDC can be used. MDC has a basic implementation in the Java programming language. For implementation in Java, this is currently a parallel merge sort implementation.

In the DB implementation, extracting descriptors from different MDC cells is performed with a single query, which is more efficient in terms of working with the DB. Extracting descriptors from a hash table in RAM implementation is not a complex process, and running it in parallel mode is not expected to result in a significant performance gain; on the contrary, performance may decrease due to the cost of parallelization. Therefore, parallelism is not applied here.

Calculating the similarity of descriptors is a field for applying parallel computing. When it is necessary to compare a large number of descriptors or when comparing descriptors with large-dimensional vectors, parallel computing can significantly improve search performance.

In this work, to compare the list of found descriptors with the searched one, we suggest using parallelism at the thread level. The list of descriptors is divided into parts, and the comparison is performed for each part in a separate thread. The results from each thread are used to form the final list. This allows for the efficient use of workstation resources.

Using parallelism at the thread level to compare vectors is not very effective, as there are more specific approaches for this. In particular, hardware tools that are better suited for processing large amounts of similar data can be used, such as SIMD instructions of the central processing unit (CPU), such as SSE, AVX, AVX-512, etc., and graphics processing units (GPUs).

SIMD instructions efficiently utilize the computing power of a single CPU core by performing parallel operations on multiple data elements simultaneously. In contrast, a GPU has an architecture with a large number of simple cores and threads, allowing it to perform a large number of parallel computations. This can provide higher performance compared to CPUs, especially when processing large data arrays, although GPUs have higher overhead costs for initialization and data transfer.

To use SIMD in Java, you can use the Java Vector API, a low-level API for working with vector instructions [33]. There are also high-level libraries, such as ND4J, that can use SIMD or GPUs internally [34]. To perform calculations on GPUs in Java, JOCL (a wrapper over OpenCL) [35] is often used, as well as other libraries, such as JCuda, to access CUDA [36].

It is important to note that in order to process data on a GPU, it must first be transferred to video memory [37], which can become a bottleneck and significantly affect the overall performance of this approach.

This direction is extremely important in the context of MDC optimization, but it requires additional study and testing of a large number of software libraries and solutions. It also requires consideration of a large number of parameters and features of each processor, GPU, or driver version for them to work.

Therefore, this paper does not discuss in detail the improvement of search algorithm performance through the use of low-level hardware optimizations.

Experiment methodology

To verify the effectiveness of WSA, two experiments are conducted:

1) comparison of the effectiveness of sequential and parallel WSA variants;

Сучасний стан наукових досліджень та технологій в промисловості. 2025. № 4 (34)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

2) comparison of the effectiveness of using MDC with sequential WSA with other CBIR search models.

The experiments test a real-life search scenario where the search engine finds a certain number of the most similar images (top-N), and the user determines which images are relevant to them.

In the experiments, WSA always performs a single search wave to ensure high search quality, as mentioned earlier.

All software implementations of the models presented in the experiments were developed using the Java 17 programming language and Spring Framework 6. PostgreSQL 15.2 is used to store MDC in the database. The results of the experiments were exported from the software solution in Excel table format.

All experiments were conducted on a MacBook Pro 2021: M1 Pro processor with ARM architecture, 10 cores with a frequency of up to 3.2 GHz, 16 GB of LPDDR5 SDRAM with a speed of up to 200 GB/s, 512 GB SSD, integrated GPU with 16 cores.

To conduct the experiment comparing the efficiency of the sequential and parallel versions of the algorithm, we used the MDC functionality, which allows generating descriptors and placing them in cells [32]. Artificial generation of descriptors allows creating vectors of different dimensions and the desired number of descriptors. This solves the problems of using real descriptors: the long process of searching for data sets to achieve the desired number of images and creating descriptors of different dimensions for them.

During the experiment, the time spent on the search is compared and the gain from the parallel algorithm is evaluated for different numbers of descriptors, vector dimensions, and number of threads.

The results provide an understanding of the conditions under which the use of a parallel algorithm provides advantages.

The second is an experiment in which the MDC model is compared with other search models for CBIR based on: multidimensional tree (KD-tree), hashing (LSH), graphs (HNSW), inverted index (IVFFlat), quantization and inverted multi-index (IMI), and vector database (pgvector). A comparison with exhaustive search is also made. For the experiment, basic models of each approach were selected because they demonstrate the general properties of a particular approach and are easier to implement or run than existing modifications. We create implementations of other models ourselves or use ready-made solutions if they are available and their configuration is simple.

This experiment is a large-scale test and comparison of the MDC model with other CBIR models in general, including testing the effectiveness of the presented search algorithm. It allows us to identify the strengths and weaknesses of the model and search algorithm compared to competitors and to predict the most favorable scenarios for using the model.

In all models, the metric for comparing descriptor vectors is Manhattan distance, unless otherwise specified.

To conduct this experiment, descriptors created for images from the following datasets are used:

1) 100,000 images from the COCO dataset [38] are used;

2) 750,000 images from Various tagged images [39] are used;

3) 150,000 images from Amazon bin images [40] are used.

This allows us to perform the experiment for 1 million descriptors. Of course, in real Big Data CBIR systems, the number of images and their corresponding descriptors can be significantly larger, but the results obtained allow us to compare the effectiveness of using different models in different conditions and scale the results as needed.

For this experiment, the main metrics are search time and recall. Recall determines the quality of the search. It is calculated using the following formula:

$$\mathrm{Re}\,call = \frac{TP}{TP + FN}, \tag{3}$$

where TP (true positive) is the number of correctly classified positive examples, and FN (false negative) is the number of positive examples that the model failed to detect.

Another key metric that is usually evaluated for CBIR models is precision, which is not evaluated due to the specifics of the experiment, which will be described below.

The first additional metric is the time it takes to build a search model. This parameter can be critically important if the search engine needs to ensure continuous operation and quickly adapt to updates in the storage content.

The second additional metric is labor intensity, which shows how many descriptors were compared during the search. It is determined when it is possible to obtain values from the model. It is provided for informational purposes.

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2025. No. 4 (34)

Invariant Brightness Histogram descriptors were created for all images, as described in more detail in [32]. 100 images were randomly selected for the search. Two modifications were created for each image: a 180-degree rotation and a 2x reduction in scale. During the search, a positive result is finding a group of these three images among the top 10 and top 100 search results. For this reason, calculating the accuracy metric is inappropriate in this experiment, as the number of relevant results under the conditions of the experiment is small.

Each CBIR model has its own parameters that affect its structure and search efficiency. For each model, different combinations of parameters are experimentally tested to show the best result in terms of quality and search speed when using 1 million descriptors and the top 10 and top 100 search results. Testing the use of a larger number of found results is inappropriate due to the inefficiency of processing the obtained results. Determining the parameters of other models is not the subject of this work, so it will be briefly discussed in the next section. Detailed comparison results can be found on Google Drive [41]. The configurations of the models that showed the best results are compared with MDC.

Additionally, load testing is performed using the Postman utility [42]. 100 virtual users simultaneously searched for the top 100 most similar images for randomly selected descriptors from the repository for 1 minute, with a delay of 1-2 seconds between attempts. This allows us to determine the actual throughput and efficiency of the model in conditions close to real life, rather than simply testing them synthetically. During testing, the total number of requests made during this period, the throughput of the search engine, and the average response time are evaluated.

Parameters of alternative models participating in the experiment

For the multidimensional tree model, we implemented a KD-tree. The implementation is unbalanced, and the hierarchy is preserved as specified by the order in which elements are added to the tree. Each node contains one descriptor. The axis along which the space is divided at each level depends on the depth of that level, which is limited by the logarithm of the number of descriptors. An alternative branch is checked during the search if fewer than the required number of results are found or if the difference between the vectors is greater than the difference between the elements at the current level. Due to the last mentioned check, the selected vector similarity metric plays an important role. Therefore, for this model, the results for Manhattan and Quadratic Euclidean distances are given in the experiments. Other model parameters are not configured.

For hashing, we independently implemented the LSH model. Its main parameters are the number of tables t and the number of hash functions h. Each table has its own set of hash functions. The more hash functions and tables there are, the more accurate the results are and the smaller the search area is, but it takes more time to add elements to the table and search. It is important to determine a balanced value for these parameters. Combinations from the following parameter groups were tested: $t = (5,\ 10)$ and $h = (50,\ 75,\ 100)$. The following parameters were selected: $t = 10$, $h = 75$.

To test the vector database, we used an add-on to the PostgreSQL 15 database called Pgvector, which provides extensive capabilities for working with vectors within the database. In a simple usage scenario (exhaustive search), it does not require additional configuration and is ready to use immediately after installation in the DBMS.

For the reverse index, two IVFFlat implementations with k-means clustering are tested: one is implemented independently in RAM and uses the Manhattan metric during the search, the other is an implementation with Pgvector and uses the Euclidean metric during the search (Pgvector does not provide the ability to use the Manhattan metric for IVFFlat). For this type of model, the main parameter is the number of clusters. All descriptors are divided into clusters, and the search takes place only in one of the clusters or continues in the next cluster similar to the centroid. The following values were tested (10, 100, 100), and 100 was selected as the most effective.

For MDC, the main parameters are the number of measurements N and the number of intervals per measurement k. Their values change the number of cells in MDC and, accordingly, the number of descriptors in one cell. Combinations from the following parameter groups were predefined and tested: $N = 4$, $k = (18,\ 20)$, which provide about 10 descriptors in a cell. The parameters $N = 4$, $k = 20$ were selected.

The IMI model's own implementation is used to quantize the vector and the inverse multi-index. Its main parameters are the number of subspaces m and the number of clusters in each subspace k. These parameters change the number of combinations of

Сучасний стан наукових досліджень та технологій в промисловості. 2025. № 4 (34)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

clusters from different subspaces, thereby changing the number of descriptors corresponding to each of them. Combinations from the following parameter groups were tested: $m = (2,\ 4,\ 8)$ and $k = (5,\ 10,\ 18,\ 20)$, and the parameters $m = 4$, $k = 10$ were selected.

The main parameters of HNSW are m – the maximum number of connections per layer, and efc – the size of the dynamic list of candidates for graph construction, which affect the quality of graph construction. Combinations from the following parameter groups were tested: $m = (8,\ 16)$, $efc = (32,\ 64,\ 128)$. The parameters $m = 8$, $efc = 64$ were selected.

The parameters selected for each model were chosen individually by experiment using a specific set of descriptors with a size of 1 million. All defined parameters or combinations of parameters showed the best results. For a different number of descriptors or descriptors from other data sets, these parameters may be different.

Experimental results and discussion

The first experiment compared the use of sequential and parallel WSA algorithms in MDC implementation in RAM.

The search for the top 100 artificially created descriptors with lengths of 32, 128, and 1024 was tested. The number of threads for the parallel implementation of the algorithm was 2, 4, and 8. The number of descriptors in MDC was equal to 1 and 10 million. The MDC parameters were $N = 4$, $k = 10$. This allowed the MDC to be filled evenly and, using 1 search wave, to obtain a specific number of descriptors for comparison with the searched one. For 1 million descriptors, this number is 6,000, and for 10 million, it is 60,000. The speed of comparing candidate descriptors with the searched one is the main operation that is performed either sequentially or in parallel for the found descriptors.

The results of comparing the application of WSA for sequential and parallel variants of the algorithm and 1 million descriptors are shown in Table 1, and for 10 million descriptors in Table 2. The graphical representation of the benefits of using different numbers of threads is shown in Figure 3 for 1 million descriptors and in Figure 4 for 10 million descriptors.

For 10 million descriptors, the length of the 1024 vector was not tested because there were insufficient resources on the computer where the experiment was conducted to store them in memory. The same applies to testing a larger number of low-dimensional descriptors, for example, 100 million descriptors with a dimension of 32.

Table1. *Search time for top 100 results in MDC with 1 million descriptors, seconds*

Count of the threads	Vector length		
	32	128	1024
1	0.00169	0.00238	0.00942
2	0.00168	0.00194	0.00550
4	0.00172	0.00198	0.00368
8	0.00160	0.00175	0.00293

Table 2. *Search time for top 100 results in MDC with 10 million descriptors, seconds*

Count of the threads	Vector length	
	32	128
1	0.02068	0.09020
2	0.01856	0.07378
4	0.00931	0.05805
8	0.00648	0.03163

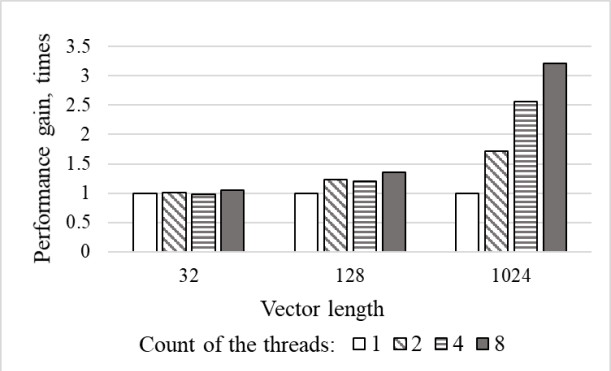

Fig. 3. Time gains when searching among 1 million descriptors using different numbers of threads for descriptors with vectors of different lengths

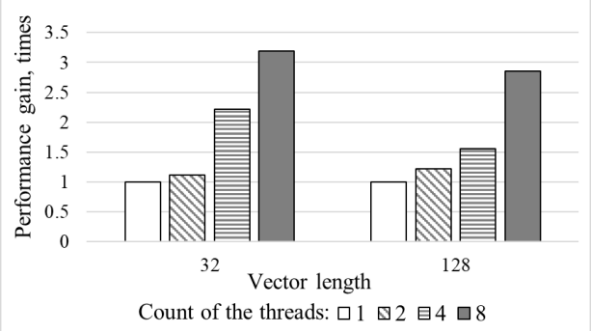

Fig. 4. Time gains in searching among 10 million descriptors when using different numbers of threads for descriptors with vectors of different lengths

The results show that the use of parallel computing when comparing a small number of descriptors (6,000)

ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

and vectors of small dimensions (32 or 128) does not make sense, as the time savings are insignificant.

When using higher-dimensional descriptors (1024) or a larger number of descriptors (10 million), it is possible to achieve a gain in using parallel computing up to a 3-fold difference when using 8 threads. This is a significant acceleration, confirming the good adaptability of MDC to the use of parallel computing and the feasibility of its use in real search scenarios.

In the second experiment, MDC in two memory implementations: in a database – MDC (DB) and in RAM – MDC (RAM) with sequential WSA is compared with:

– full search in RAM – BF RAM, which is performed using parallelism at the thread level;

– exhaustive search in a vector database – BF (pgvector);

– a model based on Inverted Multi-Index – IMI;

– a model based on KD-Tree with Manhattan metric – KD-Tree (Mh) and Quadratic Euclidean metric – KD-Tree (Sq);

– a model based on Local-sensitive hashing – LSH;

– Hierarchical Navigable Small World (HNSW) model;

– Reverse index model implemented in Pgvector with Euclidean metric – IVFFlat (pgvector).

– a model based on a reverse index implemented in RAM – IVFFlat (RAM).

For each model, metrics of completeness, labor intensity, search time, and creation time are determined. A search is conducted for the top 10 and top 100 most similar descriptors among 1 million descriptors.

The results of the experiment for searching the top 10 results are shown in Table 3. The results of searching the top 100 results are shown in Table 4. These values are averages for searching 100 selected descriptors after 10 rounds of the experiment.

Table 3. *Average search results for the top 10 results for 100 selected images among 1 million descriptors*

Model	Completeness	Labor intensity	Search time, s	Creation time, s
BF (RAM)	0.97667	1 000 000.00	0.09533	–
BF (pgvector)	0.97667	1 000 000.00	0.06888	–
MDC (RAM)	**0.97000**	**3 881.230**	**0.00129**	**2.623**
MDC (DB)	**0.96000**	**3 505.960**	**0.07668**	**812.562**
IMI	0.87333	4 354.180	0.00268	342.653
KD-Tree (Sq)	0.85600	808.645	0.00043	1.762
KD-Tree (Mh)	0.97667	967 798.782	0.49160	1.827
LSH	0.96567	32 524.953	0.02039	26.085
HNSW	0.97667	–	0.00582	376.410
IVFFlat (RAM)	0.91667	15 289.790	0.00674	3265.941
IVFFlat (pgvector)	0.91600	–	0.00530	2.380

Table 4. *Average search results for the top 100 results for 100 selected images among 1 million descriptors*

Model	Completeness	Labor intensity	Search time, s	Creation time, s
BF (RAM)	0.99333	1 000 000.00	0.09940	–
BF (pgvector)	0.99333	1 000 000.00	0.07259	–
MDC (RAM)	**0.98000**	**3 881.230**	**0.00142**	**2.623**
MDC (DB)	**0.97000**	**3 505.960**	**0.07284**	**812.562**
IMI	0.89333	4 360.230	0.00256	342.653
KD-Tree (Sq)	0.93033	3 091.157	0.00182	1.762
KD-Tree (Mh)	0.99333	989 919.981	0.50460	1.827
LSH	0.97133	40 103.314	0.02567	26.085
HNSW	0.99000	–	0.00661	376.410
IVFFlat (RAM)	0.93667	15 289.790	0.00675	3265.941
IVFFlat (pgvector)	0.92500	–	0.00839	2.380

The results obtained remain consistent when searching for the top 10 and top 100 results, so the results from Tables 3 and 4 are analyzed together. They show that even when using a full search, the completeness metric is not always 100%. This is due to the characteristics of the specific descriptor type selected.

Сучасний стан наукових досліджень та технологій в промисловості. 2025. № 4 (34)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

The maximum possible completeness result, apart from exhaustive search, was also shown by the KD-Tree model with the Manhattan metric, but the labor intensity of such a search is close to exhaustive search, and the same applies to the search time. This indicates that it is not advisable to use this model with this descriptor similarity metric. The HNSW model also showed the maximum completeness result, demonstrating an average search and model construction time. Next, MDC and LSH showed similar results in terms of completeness. However, MDC (RAM) showed much better performance in terms of search time, model construction time, and labor intensity. The implementation of MDC (DB) showed worse results than BF (pgvector) in terms of both search quality and time, which raises doubts about the advisability of its use.

Other models showed worse search results. IMI showed good results in terms of search time, but poor completeness. KD-Tree with Quadratic Euclidean Metric showed the best results in terms of search time and labor intensity, but the lowest completeness. IVFFlat implementations showed low completeness results because the search was performed in only one cluster, while the search time indicator is high.

In terms of model construction speed, the leaders are MDC (RAM), both KD-Tree and IVFFlat (pgvector) variants. In MDC, this is due to the peculiarities of the structure and the construction process, KD-Tree works fast due to the low dimension of vectors, and IVFFlat (pgvector) has implemented optimizations for clustering, because IVFFlat (RAM) without any optimizations showed the worst result. In MDC (DB), the result is average due to the cost of communication with the database.

After analyzing the results, the conclusion is as follows: exhaustive search demonstrates the best search quality, but requires a lot of time to search; MDC shows some of the best performance in terms of time and quality when placed in RAM; IMI with a small search output is not always effective; KD-Tree is ineffective in terms of either search quality or speed; LSH demonstrates balanced average results, HNSW demonstrates the best results in terms of completeness, but the search and model building time is longer than in MDC, IVFFlat demonstrates average results in terms of completeness and search time, but the structure building time strongly depends on the implementation. Based on the set of indicators, it can be concluded that under these conditions, MDC (RAM) demonstrates the best results among the models considered. However, the use of MDC (DB) is not promising due to results that are worse than those of BF (pgvector).

The load testing results are shown in Table 5. An example of the load testing results for MDC (RAM) is shown in Figure 5. The graph shows that with a constant load of 100 users, the search engine provides stable response times and consistently processes a large number of queries without any noticeable drop in performance.

Table 5. *Load testing results*
(100 users made requests to search for the top 100 results within 1 minute at intervals of 1–2 seconds)

Model	Total number of requests sent	Throughput (requests/second)	Average response time (ms)
BF (RAM)	302	4.45	9 813
BF (pgvector)	2 248	32.81	1 763
MDC (RAM)	**6 377**	**94.13**	**6**
MDC (DB)	**1 495**	**21.92**	**3 018**
IMI	6 403	95.21	4
KD-tree (Sq)	6 418	95.52	6
KD-tree (Mh)	101	1.48	10 758
LSH	5 765	84.76	60
HNSW	6 209	92.04	16
IVFFlat (RAM)	6 386	94.48	15
IVFFlat (pgvector)	6 301	92.19	15

ISSN 2522-9818 (print)
ISSN 2524-2296 (online)
Innovative technologies and scientific solutions for industries. 2025. No. 4 (34)

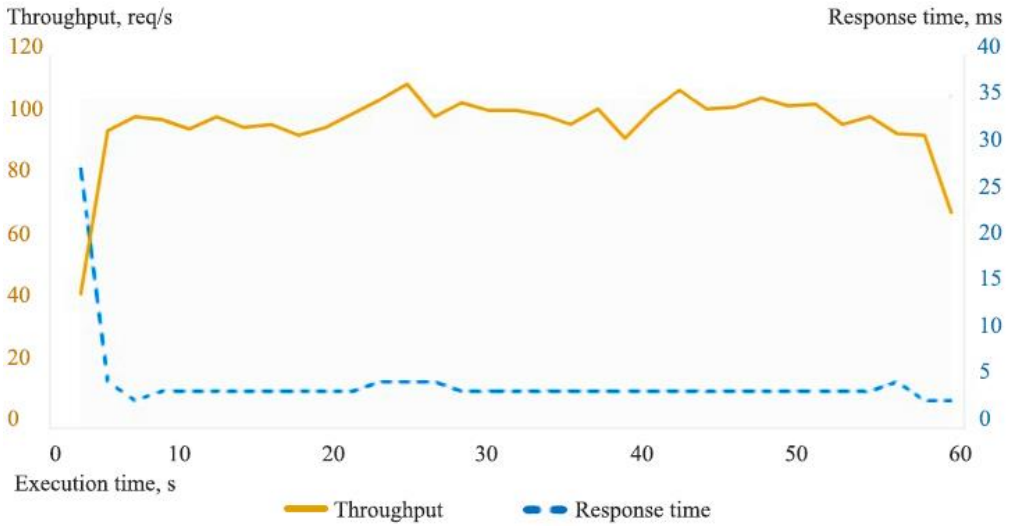

Fig. 5. Load testing results for the MDC (RAM) model

The use of exhaustive search is expected to be inefficient, as is searching in a multidimensional tree with a Manhattan metric, which approximates exhaustive search.

MDC (DB) shows weak results, slightly worse than exhaustive search in a vector database, so the futility of this MDC modification is confirmed by the results of this study.

LSH showed slightly worse results than MDC (RAM), IMI, KD-tree (Sq), HNSW, and IVFFlat in both modifications. These models show similar results in this experiment, which indicates approximately the same throughput of the models. Therefore, the choice between them should be made based on the results in Tables 3 and 4, which include completeness as an indicator of search quality.

Conclusions and prospects for further development

This paper presents a method for searching for similar images based on image descriptor vectors stored in the MDC model, which functions on the basis of the MDC structure and data stored at the descriptor vector processing stage. The algorithm that implements the method is called the Wave-Search Algorithm (WSA). It performs a wave search relative to the cell identified for the desired descriptor. It has adapted versions for placing MDC in RAM and databases.

The algorithm has a parallel version that uses parallelism at the thread level and efficiently utilizes workstation resources. It speeds up the search by up to 3 times when using a large number of descriptors (more than 10 million) and/or descriptors with vectors of considerable size (from 1024).

Compared to other CBIR models, MDC with WSA placed in RAM demonstrates very competitive results. Together with the LSH and HNSW models, it shows a search accuracy of about 97-98% according to the completeness metric. And one of the best search speeds among the tested models, spending an average of 0.0012 seconds when searching for the top 10 and top 100 results among 1 million descriptors. It also has one of the best model construction times, at around 2.5 seconds.

The MDC version stored in a database demonstrates worse results than when using a vector database (pgvector extension for PostgreSQL) in terms of both search time and quality. Therefore, its further development is not promising.

Load testing shows that the model works effectively under load and does not reduce performance, which confirms its suitability for use in real search engines, including in Big Data environments.

The use of MDC with the presented search algorithm is recommended in situations where search quality and speed are important, as well as the time it takes to create or update the content of the repository. In other words, the scope of application is quite broad.

A further direction of research is the creation of versions of the search algorithm that use SIMD operations and a graphics processor to perform parallel vector comparisons. And conducting experiments using a larger number of descriptors and descriptors of higher dimensioni.

Сучасний стан наукових досліджень та технологій в промисловості. 2025. № 4 (34)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

References

1. American Society for Indexing, "*History of Information Retrieval*", available at: https://asindexing.org/about-indexing/history-of-information-retrieval/ (last accessed 27.04.2025).
2. Maña, N., Babiera, J., Bayloces, K., Palmer, X.-L., Potter, L., Lavilles, R., Velasco, L. (2024), "Information Retrieval Systems: A Methodological Review", *Proceedings of the Future Technologies Conference (FTC) 2024, Volume 3*, Springer Nature Switzerland, Cham, P. 572–591. DOI: 10.1007/978-3-031-73125-9_36
3. Rostami, C., Hosseini, E., Saberi, M. (2021), "Information-seeking behavior in the digital age: use by faculty members of the internet, scientific databases and social networks", *Information Discovery and Delivery*, Vol. 50, No. 1, P. 87–98. DOI: 10.1108/idd-02-2020-0014
4. Jain, R. (2023). "A Comparative Study of Breadth First Search and Depth First Search Algorithms in Solving the Water Jug Problem on Google Colab", *SSRN Electronic Journal*. DOI: 10.2139/ssrn.4402567
5. Li, X., Yang, J., Ma, J. (2021), "Recent developments of content-based image retrieval (CBIR)", *Neurocomputing*, Vol. 452, P. 675–689. DOI: 10.1016/j.neucom.2020.07.139
6. Alsmadi, M. (2020), "Content-Based Image Retrieval Using Color, Shape and Texture Descriptors and Features", *Arabian Journal for Science and Engineering*, Vol. 45, No. 4, P. 3317–3330. DOI: 10.1007/s13369-020-04384-y.
7. Zhang, Q., Canosa, R. (2014), "A comparison of histogram distance metrics for content-based image retrieval", *Imaging and Multimedia Analytics in a Web and Mobile World 2014*, SPIE, Vol. 9027. DOI: 10.1117/12.2042359
8. Li, M., Wang, H., Dai, H., Li, M., Chai, C., Gu, R., Chen, F., Chen, Z., Li, S., Liu, Q., G. Chen. (2024), "A Survey of Multi-Dimensional Indexes: Past and Future Trends", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 36, P. 3635–3655. DOI: 10.1109/tkde.2024.3364183
9. Samoladas, D., Karras, C., Karras, A., Theodorakopoulos, L., Sioutas S. (2022), "Tree Data Structures and Efficient Indexing Techniques for Big Data Management: A Comprehensive Study", *Proceedings of the 26th Pan-Hellenic Conference on Informatics*, ACM, New York, USA, P. 123–132. DOI: 10.1145/3575879.3575977
10. Rakotondrasoa, H.M., Bucher, M., Sinayskiy, I. (2023), "Quantitative Comparison of Nearest Neighbor Search Algorithms", *arXiv*. DOI: 10.48550/arXiv.2307.05235
11. Liu, Q., Li, M., Zeng, Y., Shen, Y., Chen, L. (2025), "How good are multi-dimensional learned indexes? An experimental survey", *The VLDB Journal*, Vol. 34. DOI: 10.1007/s00778-024-00893-6
12. Li, D., Esquivel, J. (2025), "Trust-Aware Hybrid Collaborative Recommendation with Locality-Sensitive Hashing", *Tsinghua Science and Technology*, Vol. 30, No. 4, P. 1421–1434. DOI: 10.26599/tst.2023.9010096
13. Weng, Z., Zhu, Y., Lan, Y., Huang. L.-K. (2019), "A fast online spherical hashing method based on data sampling for large scale image retrieval", *Neurocomputing*, Vol. 364, P. 209–218. DOI: 10.1016/j.neucom.2019.06.053
14. Ryali, C., Hopfield, J., Grinberg, L., Krotov, D. (2020), "Bio-Inspired Hashing for Unsupervised Similarity Search", *arXiv*. DOI: 10.48550/arXiv.2001.04907
15. Jiang, X., Hu, F. (2024), "Multi-scale Adaptive Feature Fusion Hashing for Image Retrieval", *Arabian Journal for Science and Engineering*. DOI: 10.1007/s13369-024-09627-w
16. Liu, R., Zhao, J., Chu, X., Liang, Y., Zhou, W., He, J. (2023), "Can LSH (locality-sensitive hashing) be replaced by neural network?", *Soft Computing*, Vol. 28, P. 1041–1053. DOI: 10.1007/s00500-023-09402-3
17. Malkov, Y., Yashunin D. (2020), "Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 42, No. 4, P. 824–836. DOI: 10.1109/tpami.2018.2889473
18. Dong, W., Moses, C., Li, K. (2011), "Efficient k-nearest neighbor graph construction for generic similarity measures", *Proceedings of the 20th International Conference on World Wide Web*, ACM, New York, USA. DOI: 10.1145/1963405.1963487
19. Weng, S., Fan, Z., Gou, J. (2024), “A fast DBSCAN algorithm using a bi-directional HNSW index structure for big data”, *International Journal of Machine Learning and Cybernetics*, Vol. 15, No. 8, P. 3471–3494. DOI: 10.1007/s13042-024-02104-8
20. Zhibing, H. (2024), "Quick and Efficient Large Scale Approximate Nearest Neighbor Search on High-Dimensional Data", *2024 21st International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*, IEEE, P. 1–5. DOI: 10.1109/iccwamtip64812.2024.10873693
21. Zhao, J., Pierre Both, J., Konstantinidis, K. (2024), "Approximate nearest neighbor graph provides fast and efficient embedding with applications for large-scale biological data", *NAR Genomics and Bioinformatics*, Vol. 6, No. 4. DOI: 10.1093/nargab/lqae172.
22. Yousaf, M., Shakoor Khan, M., Ullah, S. (2024), "An Extended-Isomap for high-dimensional data accuracy and efficiency: a comprehensive survey", *Multimedia Tools and Applications*, Vol. 83, No. 38, P. 85523–85574. DOI: 10.1007/s11042-024-19917-y

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) *Innovative technologies and scientific solutions for industries. 2025. No. 4 (34)*

23. Liu, Y., Pan, Z., Wang, L., Wang Y. (2022), "A new fast inverted file-based algorithm for approximate nearest neighbor search without accuracy reduction", *Information Sciences*, Vol. 608, P. 613–629. DOI: 10.1016/j.ins.2022.06.086
24. Bazdyrev, A. (2023), Semi-supervised inverted file index approach for approximate nearest neighbor search. *System Research and Information Technologies*, No. 4, P. 69–75. DOI: 10.20535/srit.2308-8893.2023.4.05
25. Matsui, Y., Hinami, R., Satoh, S. (2018), "Reconfigurable Inverted Index", *Proceedings of the 26th ACM International Conference on Multimedia*, ACM, New York, USA, P. 1715–1723. DOI: 10.1145/3240508.3240630
26. Jégou, H., Douze, M., Schmid, C. (2011), "Product Quantization for Nearest Neighbor Search", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 33, No. 1, P. 117–128. DOI: 10.1109/tpami.2010.57
27. Babenko, A., Lempitsky, V. (2012), "The inverted multi-index", *2012 IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, P. 3069–3076. DOI: 10.1109/cvpr.2012.6248038
28. Qiu, Z., Liu, J., Chen, Y., King, I. (2024), "HiHPQ: Hierarchical Hyperbolic Product Quantization for Unsupervised Image Retrieval", *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38, No. 5, P. 4614–4622. DOI: 10.1609/aaai.v38i5.28261
29. Gu, L., Liu, J., Liu, X., Wan, W., Sun, J. (2024), "Entropy-Optimized Deep Weighted Product Quantization for Image Retrieval", *IEEE Transactions on Image Processing*, Vol. 33, P. 1162–1174. DOI: 10.1109/tip.2024.3359066
30. Jamalifard, M., Andreu-Perez, J., Hagras, H., López, L. (2024), "Fuzzy Norm-Explicit Product Quantization for Recommender Systems", *IEEE Transactions on Fuzzy Systems*, Vol. 32, No. 5, P. 2987–2998. DOI: 10.1109/tfuzz.2024.3365722
31. pgvector, GitHub. "*pgvector: Open-source vector similarity search for Postgres*", available at: https://github.com/pgvector/pgvector (last accessed 27.04.2025).
32. Danylenko, S., Smelyakov, S. (2025), "Development of a Multidimensional Data Model for Efficient Content-based Image Retrieval in Big Data Storage", *Radioelectronic and Computer Systems*, Vol. 2025, No. 1, P. 137–152. DOI: https://doi.org/10.32620/reks.2025.1.10
33. Sandoz, P. "*JEP 448: Vector API (Sixth Incubator)*", available at: https://openjdk.org/jeps/448 (last accessed 01.05.2025).
34. Deeplearning4j, "*Deeplearning4j Suite Overview*", available at: https://deeplearning4j.konduit.ai/ (last accessed 01.05.2025).
35. jocl.org, "*Java Bindings for OpenCL*", available at: http://www.jocl.org/ (last accessed 01.05.2025).
36. jcuda.org, "*JCuda*", available at: http://www.jcuda.org/jcuda/JCuda.html (last accessed 01.05.2025).
37. Nevliudov, I., Yevsieiev, V., Maksymova, S., Gopejenko, V., Kosenko, V. (2025), "Development of mathematical support for adaptive control for the intelligent gripper of the collaborative robot manipulator", *Advanced Information Systems*, Vol. 9, No. 3, P. 57–65. DOI: https://doi.org/10.20998/2522-9052.2025.3.07
38. COCO, "*Common Objects in Context*", available at: https://cocodataset.org/ (last accessed 02.05.2025).
39. Greg, "*Various Tagged Images*", available at: https://www.kaggle.com/datasets/greg115/various-tagged-images (last accessed 02.05.2025).
40. Hyun, W. "*Amazon Bin Image Dataset (536,434 images, 224×224)*", available at: https://www.kaggle.com/datasets/williamhyun/amazon-bin-image-dataset-536434-images-224x224 (last accessed 02.05.2025).
41. Danylenko, S. "*MDC-2025-WSA*", Available at: https://drive.google.com/drive/folders/1LNgV8MzNkhXC7elWOQemvFFkBddW_wPM?usp=sharing/ (last accessed 02.05.2025).
42. Postman API Platform, "*Postman: The World's Leading API Platform*", available at: https://www.postman.com/ (last accessed 02.05.2025).

Received (Надійшла) 06.06.2025
Accepted for publication (Прийнята до друку) 30.11.2025
Publication date (Дата публікації) 28.12.2025

Відомості про авторів / About the Authors

Danylenko Stanislav – Kharkiv National University of Radio Electronics, PhD Student, Software Engineering Department, Kharkiv, Ukraine; e-mail: stanislav.danylenko@nure.ua, ORCID ID: https://orcid.org/0000-0002-8142-3018; Scopus ID: https://www.scopus.com/authid/detail.uri?authorId=57816229800

Smelyakov Kyrylo – Doctor of Sciences (Engineering), Professor, Kharkiv National University of Radio Electronics, Head of the Software Engineering Department, Kharkiv, Ukraine; e-mail: kyrylo.smelyakov@nure.ua, ORCID ID: https://orcid.org/0000-0001-9938-5489; Scopus ID: https://www.scopus.com/authid/detail.uri?authorId=57203149663

Сучасний стан наукових досліджень та технологій в промисловості. 2025. № 4 (34)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

Даниленко Станіслав Дмитрович – Харківський національний університет радіоелектроніки, аспірант, кафедра програмної інженерії, Харків, Україна.

Смеляков Кирило Сергійович – доктор технічних наук, професор, Харківський національний університет радіоелектроніки, завідувач кафедри програмної інженерії, Харків, Україна.

МЕТОД ПОШУКУ ЗОБРАЖЕНЬ НА ОСНОВІ ВМІСТУ В БАГАТОВИМІРНІЙ МОДЕЛІ ДАНИХ У МАСШТАБІ ВЕЛИКИХ ДАНИХ

Предметом роботи є метод і алгоритми пошуку зображень за вмістом у моделі багатовимірного куба (MDC). **Мета дослідження** – розробити метод пошуку зображень у моделі MDC на основі векторів дескрипторів зображень та алгоритму, що реалізує цей метод послідовної та паралельної версії. **Завдання статті** передбачають: формування вимог до методу пошуку; аналіз структури MDC та визначення підходу до методу пошуку; розроблення методу й алгоритмів пошуку за умови розміщення моделі в оперативній пам'яті та в реляційній базі даних; упровадження паралельних обчислень в алгоритмі пошуку; аналіз альтернативних моделей, основаних на багатовимірних деревах, графах, гешуванні, зворотному індексі, квантуванні та зворотному мультиіндексі; розроблення метрик і проведення експериментів для порівняння ефективності моделі MDC з альтернативними моделями пошуку. **Методи дослідження**: аналіз можливостей застосування паралельних обчислень на рівні потоків і оптимізацій за допомогою апаратного забезпечення; розроблення методу пошуку, послідовної та паралельної версій алгоритму його реалізації; аналіз сучасних моделей пошуку, зокрема *KD-tree, Locality-sensitive hashing, Hierarchical Navigable Small World, Inverted File with Flat Compression, Inverted Multi-Index*; експерименти з дескрипторами зображень з мережі Інтернет та штучно генерованими для визначення ефективності паралельної версії алгоритму та порівняння результативності пошуку зі згаданими моделями за метриками повноти, часу пошуку й часу створення; проведення експерименту з навантажувального тестування для оцінювання пропускної здатності моделі. **Результати дослідження**. Розроблено новий метод пошуку й алгоритм *Wave-Search*. Його паралельна версія пришвидшує виконання пошуку майже втричі; під час пошуку топ-10 і топ-100 результатів у сховищі з 1 млн дескрипторів MDC демонструє найкращі сумарні результати за метриками серед розглянутих моделей, має гарну продуктивність під навантаженням. **Висновки**: запропоновано метод пошуку й для його реалізації алгоритм *Wave-Search,* що ефективно використовує структуру MDC; модель багатовимірного куба перевищує показники ефективності альтернативних моделей пошуку, демонструє стабільність під навантаженням і має потенціал для подальшого розвитку із застосуванням апаратного прискорення.

Ключові слова: пошук за подібністю; великі дані; алгоритми пошуку; структури даних; пошук зображень на основі вмісту; паралельні обчислення; високопродуктивні обчислення; ефективність пошуку; оптимізація алгоритму.

Бібліографічні описи / Bibliographic descriptions

Даниленко С. Д., Смеляков К. С. Метод пошуку зображень на основі вмісту в багатовимірній моделі даних у масштабі великих даних. *Сучасний стан наукових досліджень та технологій в промисловості*. 2025. № 4 (34). С. 18–31. DOI: https://doi.org/10.30837/2522-9818.2025.4.018

Danylenko, S., Smelyakov, K., (2025), "Content-based image retrieval method in a multidimensional data model at big data scale", *Innovative Technologies and Scientific Solutions for Industries*, No. 4 (34), P. 18–31. DOI: https://doi.org/10.30837/2522-9818.2025.4.018