UDC 004.032.26:004.932.2:004.052. **doi: 10.32620/reks.2025.3.10**

Stanislav DANYLENKO, Kirill SMELYAKOV

Kharkiv National University of Radio Electronics, Kharkiv, Ukraine

METHOD FOR OPTIMIZING DESCRIPTORS CLUSTERING IN THE FEATURE DATABASE OF A CONTENT-BASED IMAGE RETRIEVAL SYSTEM

*The **subject** of this study is the method of grouping image descriptors that are placed in the feature database of search systems. This study aims to develop a method for optimizing the clustering of descriptors in Big Data storage, represented by a Multidimensional Cube data model. Further use of the formed clusters for effective search in content-based image retrieval systems. The **task** is to: analyze modern approaches and solutions for forming groups of image descriptors in the feature database; formulate the problem of the clustering method in the Multidimensional Cube and the requirements for its optimization; develop an abstract optimization method; develop specific optimization algorithms for different types of model placement in memory; develop metrics; perform experiments and compare the results with analogs. The **methodology** includes analyzing the process of forming groups using different methods, determining their advantages and disadvantages, applying numerical methods to optimize the clustering process, conducting experiments with data sets available on the Internet, evaluating the effectiveness of the optimization method, and generating result tables for comparison with analogs. The following **results** were obtained: a method for optimizing the clustering of descriptors was developed for use in the Multidimensional Cube data model, and optimization algorithms were developed for model placement in random-access memory and databases. The results of clustering, determined by the metrics of time spent and cluster filling, were compared with those of descriptor clustering performed using the k-means algorithm and the Product Quantization approach with the implementation of Inverted Multi-Index. The results showed that the use of the model with the developed optimizations demonstrates that the quality of descriptor clustering is not worse than when using Inverted Multi-Index and better in terms of time spent than when using k-means or Inverted Multi-Index. **Conclusions**: The developed method for optimizing descriptor clustering significantly improves the distribution of descriptors within the Multidimensional Cube model and makes it a good alternative for use in content-based image retrieval systems.*

Keywords: *information technology; algorithms and data structure; data storage; big data; search system; multidimensional data model; optimization method; clustering; computational complexity; CBIR.*

1. Introduction

1.1. Motivation

Information is abundant in today's world and comes in various forms and formats. Most of the information is digital, processed by computers, and available on the Internet. Every year, such an amount of digital information is created that, with an annual growth of 20%, the number of bits will exceed the number of atoms on Earth in approximately 350 years. In addition to the volume of information, the amount of energy required to process and store information has been increasing significantly, which has a negative impact on the environment [1].

Software that works with large amounts of data is often categorized as Big Data. The 3Vs model can be used to define the concept of big data: volume, the amount of data created and stored; variety, the diversity of data types; and velocity, the speed at which data are generated, transmitted, and processed [2].

These data are generated across various domains, including social, vehicular, healthcare, urban, industrial, and educational sectors, and originate from various sources, such as social media, autonomous vehicles, IoT sensors, medical records, financial transactions, and supply chain logistics [2].

Cloud computing is typically used to work with such data, provided by Amazon Web Services (AWS), Azure, and Google [2]. For example, according to approximate unofficial estimates, as of the 2020s, AWS requires approximately 500 Exabytes of storage for data, whereas Dropbox requires approximately 733 Petabytes of storage. In addition to systems that store data, there are also streaming services that continuously transmit data over the network. For instance, Netflix streams 140 million hours of video per day, with approximately 1 Gigabyte of storage required for each hour. Google has indexed at least 30 billion pages, requiring around 62 Petabytes of storage [3].

Information on the Internet is not unique. A large amount of duplicate and modified original data are present. Searching for the required information in this huge amount of data is not an easy task. In particular, in scenarios where high-rate search queries are generated, the load on the search system is increased. A brute-force

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

search through the entire dataset is not feasible. In this case, approaches are employed to minimize the search space, that is, to reduce the number of potential candidates that must be examined to find the optimal or most suitable candidate [4]. Some of the possible approaches are as follows: Beacon Guided Searching, Genetic Algorithm, Nearest Neighbor Search (NNS) or Approximate Nearest Neighbor Search (ANNS) [5]. The NNS category has advanced significantly using various techniques and data structures. Exact search may require excessive resources; thus, approximate approaches are often used instead [6].

The main one is a group of ANNS approaches where information is grouped within the storage on the basis of certain features or using a certain similarity criterion. Each group is assigned an index by which it can be accessed [7]. The search is performed only among the information that is included in a certain group and not across the entire dataset. This increases the speed at the cost of accuracy. If the search within a specific group is unsuccessful, techniques are applied to identify the closest groups in which the search should continue.

These approaches also apply to content-based image retrieval (CBIR) systems [8]. Such a search can be performed in both publicly available search systems and in closed corporate storage with proprietary information. This task is challenging due to the large number of images in storage, duplicates and modifications, and the complexity of formalized image representation and comparison during search.

Currently, the most common way to represent an image is through a descriptor. This is a compact vector of the normalized values. It is often a one-dimensional vector of a certain length. The values in the vector describe certain characteristics of the image as a whole or a part of it [9]. Image search compares image descriptors using a certain metric and ranks the results according to the calculated similarity score. Adaptations of ANNS approaches have been adapted for use with image descriptors. Similar to other types of data, specialized search models form groups of image descriptors and use them to improve search efficiency [10].

The efficiency of a CBIR model depends on the descriptors it can use, how it groups them in its feature database (FDB), and how it performs search based on the formed groups. However, the FDB structure and the method for grouping descriptors within it have the greatest impact on performance.

Problem statement. The efficient grouping of image descriptors in Big Data storage to ensure high search efficiency in content-based image retrieval systems is problematic. After grouping, the descriptors should be grouped based on certain similarity features. Grouping should be performed quickly to ensure that it can be performed repeatedly when updating the storage content to avoid halting the search system in conditions of high-intensity queries. Moreover, it provides a convenient group structure for applying search methods.

1.2. State of the art

Multidimensional data structures, such as kd-trees [11], are generally not considered state-of-the-art solutions for modern CBIR systems, as they do not scale well to large datasets and high-dimensional spaces.

Graph-based methods, such as HNSW [12] and deep learning-based solutions [13], can be used. Each descriptor is assigned to a specific node in the graph based on the proximity and relationships between the descriptors in the vector space. Similar descriptors are connected by edges according to defined similarity rules, forming clusters or subgraphs. Groups of similar descriptors are effectively represented as interconnected nodes. Navigating the graph structure, leveraging shortcuts and hierarchical connections for efficient traversal and retrieval, the search for neighboring groups is performed.

Another approach is to apply hashing to descriptor vectors. These algorithms can be either traditional algorithms, such as LSH, SH, or ITQ [14], or deep hashing algorithms [15]. A special hash function, or several such functions, are trained or selected randomly or based on the labeled data and applied to the descriptor vector. The vector is reduced to a certain value and placed into a specific bucket. Other descriptors are placed in the same or other buckets according to the same principle. Then, based on the hash value of the searched descriptor, the bucket with the closest matching descriptors can be found. However, determining which buckets are most similar to the current one is challenging because the value distribution properties of the hash function do not inherently preserve similarity relationships.

Classical clustering methods, such as k-means, DBSCAN, STING, FCM, and ABC, can also be used [16]. For example, k-means is used to form clusters based on the available descriptor values [4]. Similar descriptors are assigned to the same cluster. Each cluster has a centroid. The distance to the cluster centroids is calculated for the image searched. The closest cluster is determined by this distance. If the search is unsuccessful in a cluster, the search continues in the next cluster in terms of the distance from the vector to the centroid.

Because of their effectiveness, classical clustering methods have been widely used in CBIR tasks. This approach has been continuously improved through various modifications. One of these is related to the use of Product Quantization (PQ) [17] and its modifications, such as Inverted Multi-Index (IMI) [18], Optimized Product Quantization (OPQ) [19], and Enhanced Accumulative Quantization (E-AQ) [20]. The main idea behind these modifications is to divide the initial multidimensional

space of descriptor features into smaller subspaces and perform clustering within the subspaces using classical methods. This makes clustering faster and more accurate. As a result, the image descriptor corresponds to several centroids, which somewhat complicates the search algorithm. However, it allows for increased search speed and efficiency [18]. The Symmetric Distance Computation variation also helps reduce the memory requirements for operation [17].

The Multidimensional Cube (MDC) model follows the idea of dividing descriptor feature space into subspaces. The descriptor groups are defined without using the classical clustering methods. Instead, the values in the generated subspaces are distributed across specific intervals. These intervals form the basis for cells, where descriptors are assigned according to the corresponding feature value range. Using the same principle, descriptors with similar vector values fall into the cell. If the search in a particular cell is unsuccessful, it continues in the neighboring cells, which are determined by the MDC structure [21].

Table 1 compares these methods. The definition of the data dimension in the table corresponds to the descriptor vector length that describes the image features. These approaches are considered in the context of Big Data. The advantages and disadvantages mentioned apply to this particular case.

1.3. Objective and Approach

One main limitation is that classical clustering methods are used to form groups of descriptors within the formed subspaces after the space is split into subspaces [18]. Although the dimensionality of the input vector is significantly reduced, this process is resource- and time-intensive. Popular solutions such as Faiss employ this approach [22].

This problem can be solved by adopting models that do not use classical clustering algorithms. One such model is MDC [21] although it introduces its own challenge, which this paper seeks to address.

The main disadvantage of MDC is the formation of cells (clusters) and the placement of descriptors in them. Because the descriptor values do not participate in determining the cell parameters, this method can be conditionally referred to as clustering. Only the range of possible values is considered. Thus, the cell parameters are determined in a rather abstract manner, resulting in an inefficient distribution of descriptors, which increases the search's computational complexity and negates the model's advantages.

This study aims to develop a method for optimizing the clustering of descriptors in a Big Data storage, represented as an MDC model, for further use of the formed clusters for efficient search in image-based retrieval systems.

To achieve this objective, these tasks must be addressed:

1) develop a method for optimizing the clustering of descriptors in the storage represented by the MDC model, which uses the descriptor values to determine the cluster parameters;

2) develop clustering optimization algorithms for different types of MDC placement in memory: in random-access memory (RAM) and in the database (DB);

Table 1

Comparison of the most common methods of grouping descriptors in CBIR models

Grouping method	Advantages	Disadvantages
Graphs	- Works well with dense and sparse data. - Good performance. - Simple process of determining the closest groups.	- Difficult to implement and configure. - High computational complexity and training time.
Hashing	- Suitable for high-dimensional data. - Good performance on dense data.	- Accuracy is not stable. - Complicated process of finding the closest groups. - Requires significant resources for hash function selection and hashing.
Classical clustering	- Works well on dense and large datasets. - Simple process of identifying the closest groups.	- High complexity and training time. - Does not work well with sparse data.
Product Quantization	- Effective for large datasets. - Simple process of identifying the closest groups. - High accuracy.	- Difficult to implement and configure. - Requires time for classical clustering in the subspaces.
MDC	- High speed of structure formation. - Ease of implementation. - Simple process of determining the closest groups.	- Low quality of the generated groups.

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

3) develop metrics and experimentally test the efficiency of clustering descriptors in MDC using the developed optimization method compared with clustering performed using the k-means algorithm and Product Quantization with the IMI implementation.

After applying the optimizations, the MDC model should provide effective clustering of descriptors within its FDB structure, at a level not lower than when using PQ (IMI). This will allow us to quickly build an efficiently populated search model, rebuild it if necessary without stopping the search system, and perform the search efficiently.

Given the same clustering quality, MDC can be a good competitor to PQ-based implementations because it solves their main problem and has advantages that may be necessary in certain situations.

Section 2.1 describes the MDC clustering problem. Section 2.2 introduces the clustering optimization method. In Sections 2.3 and 2.4, we present the algorithmic implementations of the method for various memory model placement strategies. Section 2.5 offers a theoretical analysis of the computational complexity of different clustering methods. The experimental methodology is outlined in Section 3. Section 4 presents the experimental results and discussion. Section 5 summarizes the study's conclusions and further steps.

2. MDC clustering optimization

2.1. MDC clustering problem details

The main idea of the PQ approach in the context of CBIR systems is to split a descriptor's feature space into subspaces. In practice, this means dividing a descriptor vector of length n to N shorter subvectors of lengths n′ thereby forming N subspaces. Furthermore, execution of the standard clustering algorithm on subvectors, forming K′ clusters in each subspace and K clusters into a total. As a result of the processing, the descriptor vector is replaced by a vector of cluster indices. Each index corresponds to a subvector in a particular subspace. The Cartesian product of the cluster indices from all subspaces describes the formed groups (clusters) in the original space. To determine the closest clusters, the distance to the cluster centroids in each subspace and the sum of the distances of all possible options must be determined. To create a search model, it is necessary to use storage descriptors [18].

MDC performs clustering differently. A vector of length n is also divided into N of subvectors with a length of n′. Each subvector corresponds to one MDC dimension. The values are aggregated in pairs on each subvector to form a single value. This value is called the dimension value. The range of its possible values depends on the number of aggregations performed and the initial range of values of the features of the normalized vector. Next, the range of possible dimension values (for example [0-1]) is evenly divided into k intervals, each of which is assigned an index. The dimension value falls into an interval with a certain index. An index vector is formed from the defined indices, which can be used to identify a group (cluster) of descriptors. The visualization of the descriptor vector processing in MDC is shown in Figure 1.

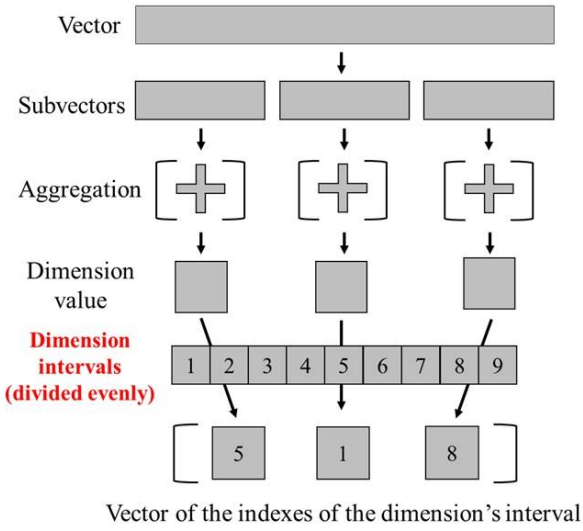

Fig. 1. Descriptor vector processing in MDC, indexes are determined based on uniform defined dimension intervals

The Cartesian product of the interval indices from all dimensions describes the clusters formed in the initial space. To determine the closest clusters, we need to take the closest index values for each dimension and form a Cartesian product of them [21]. As a result, we obtain the total number of clusters described by the following formula:

$$K = k^N, \quad (1)$$

where K – total number of clusters;

N – number of subspaces (dimensions);

k – number of intervals in each subspace (dimension).

Geometrically, the dimensions are in Euclidean space. The area bounded by the descriptor values is divided into intervals to form a multidimensional cube. The cube cells are clusters.

The problem here is that the uniform division of dimensions into intervals does not account for the actual distribution of values within the possible range. Thus, there are more in one interval and fewer in the other. Consequently, more descriptors fall into one cell and fewer into another.

Each interval should contain approximately the same number of values to solve the problem. Then the boundaries of the intervals will not be equal to each other,

and graphically the structure will look like a multidimensional parallelepiped.

2.2. Optimization method

The main idea of the optimization method is to ensure that the same number of dimension values fall into the formed dimension intervals, and approximately the same number of descriptors fall into the cells of the MDC model. For this purpose, the number of clusters should be optimal for the number of descriptors available in the storage, and the boundaries of dimension intervals should be determined based on the values of the descriptor vectors, not on the range of their possible values.

The storage contains D images for which D descriptors are calculated and the MDC is formed. Each of the N dimensions is divided into k intervals.

The first step in the optimization process is to determine the number of intervals in the dimensions rather than changing the interval parameters. According to the formula (1) the value of the number of intervals k together with the value of the number of dimensions N determines the number of MDC cells (clusters) K. The number of cells depends on how many F descriptors should be in one cluster for the convenience of searching. This number can be individual for each search system and is determined by many factors, such as search conditions and available resources. However, the main factor that dictates this value is the number of images that need to be returned to the system during one search iteration, i.e., the size of the search page. This allows searching in only one cell when a query is received. Therefore, the number of descriptors in the cluster maximally satisfy the following equation:

$$F \approx R,\ R = \frac{D}{k^N}, \qquad (2)$$

where F – expected number of descriptors in the cluster;

R – the number of descriptors in the cluster with the current parameters;

D – number of descriptors in the storage;

k – number of intervals on the dimension;

N – number of dimensions.

When optimizing, the value of N and k are iterated over until the obtained value R will not be as close as possible to the specified value F. The general algorithm for optimizing the number of clusters is as follows:

1) count the number of descriptors (D), that will be created for images from the main image storage;

2) determine the expected number of image descriptors (R) in one feature database cluster;

3) iterate over the values N and k until the resulting value R will not be as close as possible to F.

After determining the number of clusters and the intervals, the second stage of optimization begins – optimization of the distribution of descriptors across the formed cells by adjusting the boundaries of the intervals on the dimensions. With the optimal distribution of dimension values, each interval should contain approximately the same number of values, which is determined by the following formula:

$$E = \frac{D}{k}, \qquad (3)$$

where E – expected number of values in each dimension interval;

D – number of descriptors in the storage;

k – number of intervals on the dimension.

During the optimization, the boundaries of the intervals into which each dimension is divided are changed. Therefore, each dimension interval has approximately the same number of values close to E. The general abstract algorithm of this numerical optimization method is executed after the preliminary division of dimensions into equal intervals and does not require descriptors to be loaded into the MDC. The algorithm has the following steps for each dimension separately:

1) determine E by the formula (3) for a particular storage, and the value of Δ, by which the expected number of values in the interval may differ from the actual number;

2) start checking from the first dimension interval;

3) check the number of values that are currently in the interval;

4) if the number is less than E, then increase the interval upper bound, if more – then decrease it. If it is approximately equal to the value of Δ, then stop adjusting the interval boundary;

5) go to the next interval using the index;

6) repeat steps 3-5 if it is not the last interval by the index. If it is, stop the optimization. At the last interval, the value already corresponds to the expected value.

This algorithm is abstract because it implements the ideas of the method and some of its steps can be modified depending on how the MDC is stored in memory.

The values are distributed evenly over the intervals of each dimension as a result of the optimization.

Figure 2 shows an example of the feature space of the initial descriptor and the feature space divided by MDC before and after optimization. For example, for descriptors whose values are in the range [0-1] and MDC parameters are defined as N = 3, k = 3, the boundaries of the intervals after optimization are shown in Figure 3. The dimensions are named as x, y, z.

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

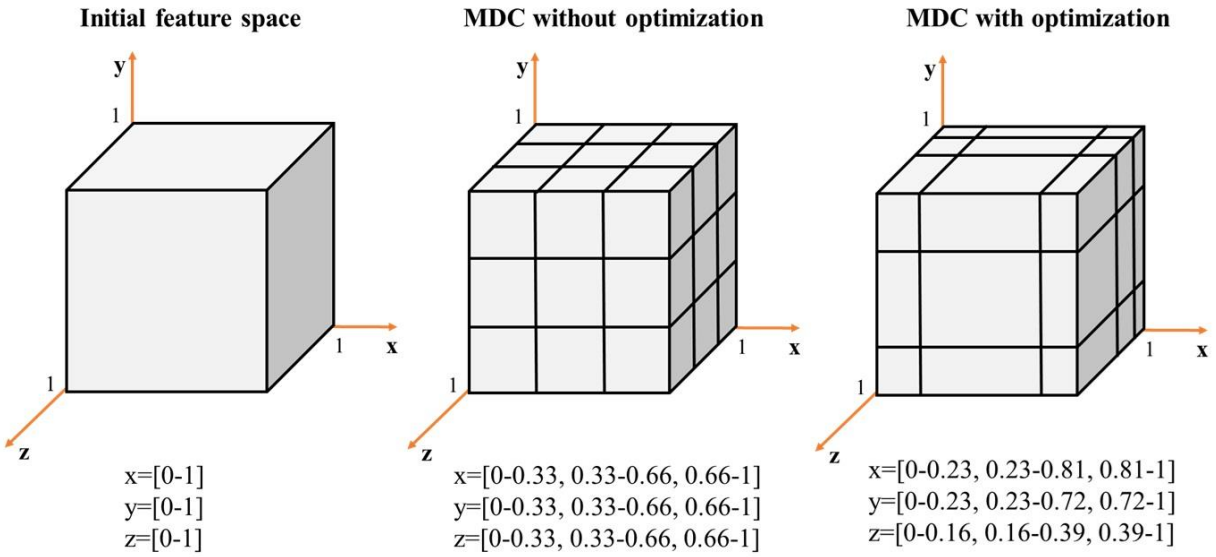

Fig. 2. Descriptor feature space in its original form, in form of MDC without optimization, and in form of MDC with optimization

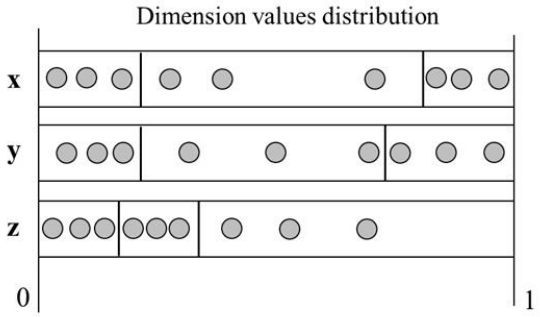

Fig. 3. Adjusting of interval boundaries based on the number of dimension values in the range of possible values. Example for 9 values

However, this does not necessarily guarantee that the descriptors are evenly distributed across the clusters. The uniformity of descriptor placement across clusters is affected by the characteristics of a particular type of descriptor and the specifics of images in the particular storage. The number of groups of related images can be either larger or smaller than the number of clusters formed in MDC.

The presented algorithm of the optimization method is called DINOA (Dimension Intervals Numeric Optimization Algorithm).

2.3. Optimization algorithm for placing MDC in database

The algorithm closest to the abstract one is used when the MDC is stored in an external storage, such as a relational database (DB). In this case, the descriptors are located in the database, and when executing the algorithm, the tools provided by the DB – SQL queries must be used. This allows us to effectively evaluate the number of dimension values that fall into the specific interval and adjust the boundaries of the intervals based on the values.

The algorithm in this implementation does not require a preliminary division of dimensions into equal intervals, but requires loading descriptors into MDC and aggregating vectors according to the value of N by the MDC rules. It has the following steps for each dimension separately:

1) determine E by the formula (3) for a specific storage and value of Δ, by which the expected number of values in the interval may differ from the actual number. Define steps of a fixed size. By default, the following step sizes are used: 0.1, 0.01, 0.001. These are the values by which the interval boundary will move within the possible range of dimension values;

2) optimization for the first dimension interval is started. The step value is specified as the largest of the defined;

3) the upper bound of the interval is increased by the step value;

4) the number of values that are currently in the interval after changing the bound is checked.

5) if the value is less than E, then the upper bound of the interval is increased by the same step. If it is greater, then the upper bound of the interval is decreased by the specified step, and the step value is replaced with the next (smaller) one from the list, and repeat steps 3-5. If the smallest defined step value is currently being used and the number of values in the interval is approximately equal, considering the value Δ, then stop the adjusting of the interval boundary;

6) go to the next interval using the index;

7) repeat steps 3-7 if it is not the last interval by the index. If it is, stop the optimization.

This optimization algorithm modification is called DINOA-in-DB. It is graphically depicted in Figure 4.

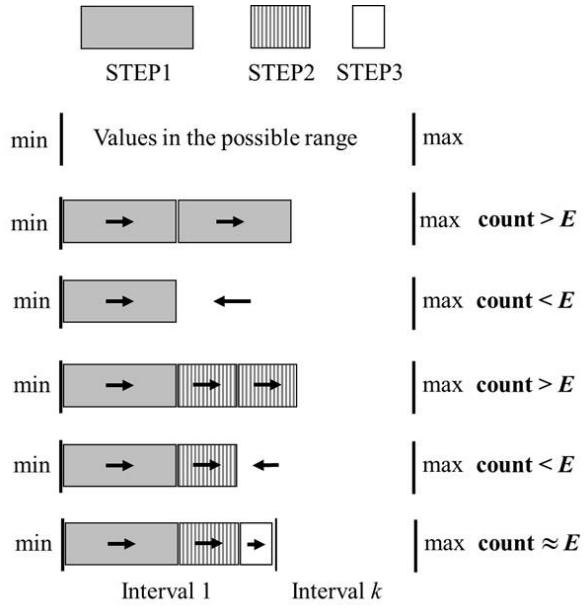

Fig. 4. Steps of the DINOA-in-DB

2.4 Optimization algorithm for placing MDC in random-access memory

The algorithm used when the MDC is placed in the random-access memory (RAM) of the workstation hosting the search system is another implementation. In this situation, the proposed algorithm is significantly simplified.

This implementation's algorithm does not require a preliminary division of dimensions into equal intervals but requires loading descriptors into MDC and aggregating vectors according to the value of N by the MDC rules. It has the following steps for each dimension separately:

1) read all dimension values and present them as a list of values;

2) sort the list in ascending order (in our implementation by the TimSort algorithm [23]);

3) divide the sorted list of values into k intervals;

4) the value of the upper bound of the previous interval plus the minimum possible value or 0 sets the lower bound of the interval. The upper bound of the interval is determined by the last dimension value in the interval or the maximum possible dimension value after aggregation.

This algorithm modification is called DINOA-in-RAM. It is graphically depicted in Figure 5.

It requires sorting the values that fall within a possible range of values instead of gradually pulling the values from the storage. This greatly simplifies and speeds up optimization.

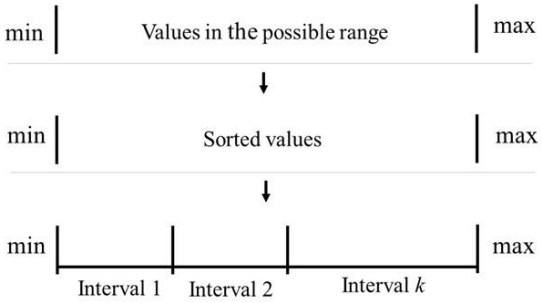

Fig. 5. Steps of the DINOA-in-RAM

For both of these implementations, all the descriptors that should be placed in the FDB should not be used. It is possible to load some descriptors, perform optimization based on them, and then distribute the remaining descriptors to the formed clusters.

2.5. Computational complexity of clustering algorithms

The estimation of the maximum computational complexity of the k-means, PQ (IMI), and MDC with DINOA-in-RAM optimization clustering algorithms is as follows:

$$\begin{aligned} O_{k-means} &= O(D \times K \times n \times i), \\ O_{PQ(IMI)} &= O(D \times K' \times n' \times N \times i), \\ O_{MDC(DINOA-in-RAM)} &= O(D \times \log(D) \times N), \end{aligned} \quad (4)$$

where D – the number of descriptors in the storage;

K – number of clusters in the feature space;

K′ – number of clusters in the feature subspace;

n – dimensionality of the space;

n′ – dimensionality of the subspace;

N – number of the subspaces;

i – number of the iterations.

Formula (4) does not consider the complexity of the DINOA-in-DB because it is difficult to estimate, since it depends on the chosen parameters of the interval boundary movement step and the specific data set. Different data sets may require a different number of steps.

The PQ (IMI) clustering estimate is the product of each subspace's clustering complexity. The MDC clustering estimate is the product of the complexity of sorting all dimensions' values. The estimates for k-means and PQ (IMI) have a variable – the number of iterations that must be performed before clusters are formed. In MDC clustering, there is only one iteration.

Optimizing the complexity and time of clustering is possible by using a part of the available data set rather

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

than the entire data set. Interval boundaries or centroids are formed on a part of the data, and then the FDB is loaded with all the data. Of course, in this case, the clustering accuracy may be lower.

3. Experiments Methodology

The experiments evaluate the effectiveness of image descriptor clustering in the FDB, which is performed as follows:

1) k-means algorithm;
2) PQ approach in the implementation of IMI;
3) MDC without optimization;
4) MDC with DINOA-in-RAM optimization;
5) MDC with DINOA-in-DB optimization.

During the experiments 2 metrics are evaluated:

1) time spent on clusters formation;
2) occupancy of the formed clusters: the number of populated clusters and the number of descriptors in the them.

The time spent plays a key role in building and rebuilding the feature database, the key component of the search model. Only the time spent on clustering is evaluated. The time spent loading descriptors into RAM is excluded from the results. For the MDC with DINOA-in-DB optimization, the time spent on loading descriptors into the database is not included, but the time spent transferring data from the database during optimization is included.

Cluster occupancy is critical to search performance. The expected number of descriptors in a cluster guarantees the expected search speed and time spent. However, the final search efficiency depends on the search method using the formed clusters.

The COCO2017 image set was used in the experiments [24]. It has different images: in terms of size, content, or processing level. In the experiments, 100 000 randomly selected images are used.

Descriptors of the Invariant Brightness Histogram type are generated in advance for the selected images and provided to the experiment [21]. These are one-dimensional vectors with fractional numbers of size 8 bytes and dimension 32, normalized in the range of [0-1].

The search system results page should display the 10 most similar images; therefore, the cluster size is set to 10. For the selected 100 000 images, 10 000 clusters will be formed. For MDC, the parameters are defined as follows: the number of dimensions is 4, the number of intervals per dimension is 10, and the expected number of values on each dimension's interval is 10 000. For PQ (IMI), the number of subspaces is 4, and the number of clusters in a subspace is up to 10. Up to 10 000 clusters can be formed for k-means clustering.

The plan of the experiment is as follows:

1) form clusters using each of the above methods; Evaluate the speed of their creation and occupancy;
2) check the clustering efficiency when using 20% of available descriptors;
3) make tables to compare the results;
4) compare and analyze the obtained results, draw conclusions.

All clustering methods were implemented using the Java 17 programming language. The software solution is a separate application for conducting clustering task experiments.

Except for DINOA-in-DB, all implementations use descriptor placement in RAM. DINOA-in-DB uses the default interval boundary movement step values. The JDBC interface is used to communicate with the database. PostgreSQL 15 is used as the DBMS.

The experiments were performed on the following computer: MacBook Pro 2021: M1 Pro processor on ARM architecture, 10-cores up to 3.2 GHz, 16 GB of LPDDR5 SDRAM up to 200 Gb/s, 512 GB SSD, integrated GPU with 16 cores.

4. Results and Discussion

4.1. Results

Figure 6 shows the resulting boundaries of the MDC dimension intervals after applying DINOA-in-RAM. These boundaries are not uniform.

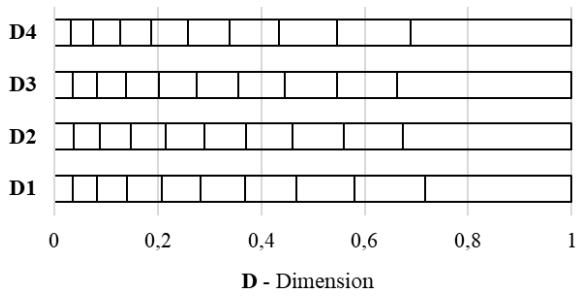

Fig. 6. The intervals after the optimization

Figure 7 shows the uneven distribution of values across intervals in one MDC dimension after uniformly defining interval boundaries. The even distribution with expected count of values achieved after applying DINOA, both when using all descriptors and when initially using 20% of the descriptors before loading the entire dataset.

Table 2 presents detailed statistics on the cluster occupancy. The first value in each table cell represents the result when all descriptors from the dataset were used. The second value shows the result when only 20% of the descriptors were initially used, followed by loading the entire dataset. Table 3 shows the main clustering metrics results using different methods.

Table 2

Distribution of descriptors among the formed clusters, all values in units

Number of descriptors in the cluster	Number of such clusters in MDC without optimization	The number of such clusters in MDC (DINOA-in-DB)	The number of such clusters in MDC (DINOA-in-RAM)	The number of such clusters after applying k-means	The number of such clusters after applying PQ (IMI)
0	9 539	7 234 7 227	7 232 7 227	455 2 140	7 304 7 202
1-10	130	985 996	988 996	5 460 4 076	1 562 1 667
11-20	50	447 444	444 444	3 284 2 236	339 347
21-30	32	275 259	278 257	692 1 030	213 167
31-40	20	214 234	210 235	104 376	102 104
41-50	12	167 161	171 163	5 116	65 91
51-100	49	447 449	446 446	0 26	182 192
101-200	57	201 199	201 201	0 0	121 129
201-300	35	25 27	25 27	0 0	55 41
301-400	18	3 2	3 2	0 0	20 15
401-500	10	1 1	1 1	0 0	12 14
501-1000	24	1 1	1 1	0 0	18 24
1001+	24	0 0	0 0	0 0	7 7

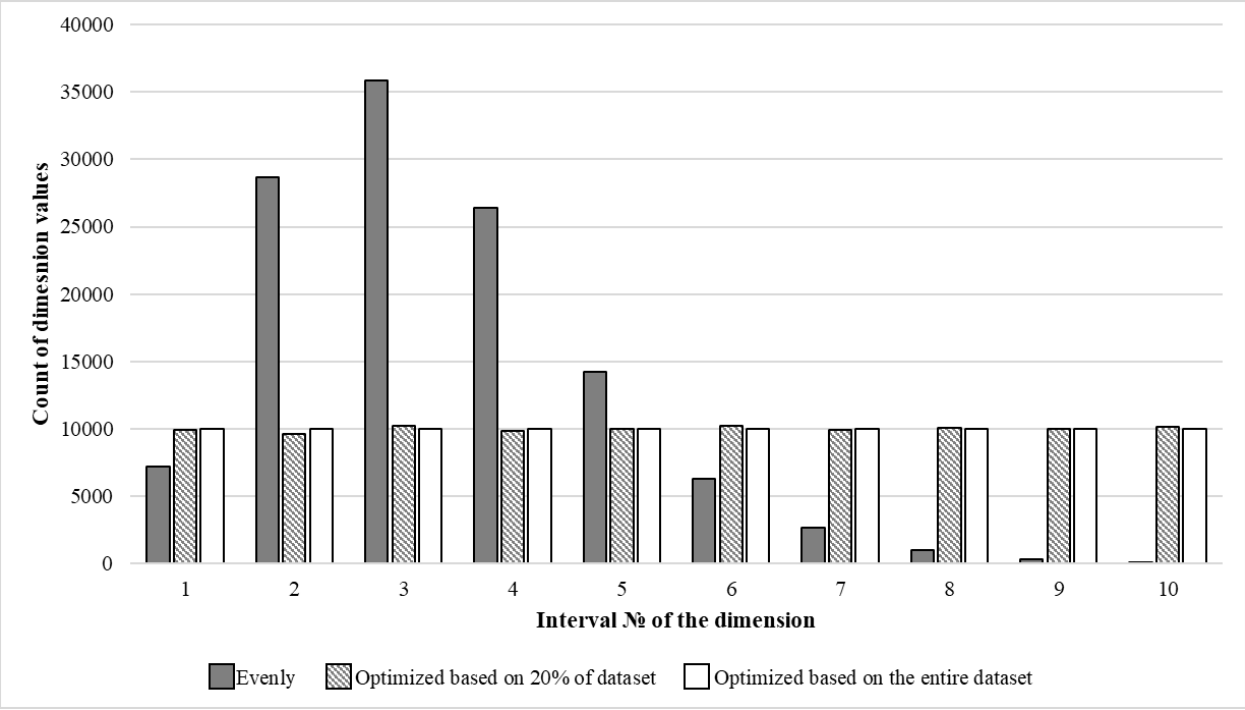

Fig. 7. The distribution of dimension values between intervals, which are: evenly divided, optimized by DINOA based on entire dataset, and optimized by DINOA based on 20% of the dataset

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

Table 3

Results of descriptor clustering – main metrics

Clustering method	Time spent, seconds	Number of populated clusters, units	Maximum number of descriptors in a cluster, units
Classic (K-means)	21 780.062 3 602.29	9 545 7 860	50 71
PQ (IMI)	43.279 10.562	2 696 2 798	1 881 1 447
MDC without optimization	0.0026	461	5 543
MDC (DINOA-in-DB)	242.738 110.354	2 766 2 773	721 717
MDC (DINOA-in-RAM)	1.758 1.527	2 768 2 773	721 717

4.2. Discussions

The k-means algorithm provides the most effective clustering in terms of cluster occupancy. When all available descriptors were used, 9 545 populated clusters were formed out of the maximum possible 10 000. The maximum number of descriptors in a cluster is 50, whereas the expected number is 10. When 20% of the descriptors were used, 7 860 populated clusters were formed, with a maximum of 71 descriptors in a cluster. That is, when using an incomplete data set, the clustering quality decreased by approximately 20%. However, the time for cluster formation decreased from 21 780 s to 3 602 s, i.e., 6 times.

In terms of cluster occupancy efficiency, the second place is taken by the MDC with DINOA-in-RAM optimization. When all descriptors and 20% of them are used, the number of populated clusters is almost the same and amounts to 2 770. The same holds true for the maximum number of descriptors in a cluster, which is 720. This means that when using a part of the available dataset, the clustering quality decreases slightly. However, the number of populated clusters is significantly lower, and the maximum number of descriptors in a cluster is significantly higher than when using k-means. The time spent using the data in both variants shows good results and does not exceed 2 s.

In the same terms, the third place is taken by the MDC with DINOA-in-DB optimization. It provides the same results in terms of the number of populated clusters and the maximum number of descriptors in a cluster as the RAM modification. However, it shows worse results of the time spent on clustering: 242 s when using all the data and 110 s when using part of the data, respectively. However, as noted in the experimental methodology, this result is expected because this variation considers the time of sending data from/to the DB. Using 20% of the descriptors does not negatively affect the clustering quality and speeds it up by 2.2 times.

The fourth most efficient approach is PQ(IMI). The clustering speed is slightly worse than that in MDC using DINOA-in-RAM but better than that when using DINOA-in-DB. The number of populated clusters at the MDC level with optimizations, with a small deviation. However, the maximum number of descriptors in a cluster is worse than that when using MDC and is 1 881 when using all the data and 1 447 when using 20% of the data. The use of part of the data does not negatively affect the clustering quality and speeds it up by 4 times.

The clustering performed by MDC without optimizations demonstrates the best cluster formation speed. It took less than 1 s; however, the number of populated clusters was the smallest among the competitors (461), and the maximum number of descriptors in a cluster was significantly higher than that of the competitors – 5 543.

Since k-means clustering can be considered a benchmark in terms of clustering quality, Table 4 compares the main metrics of the experiment with it.

Table 4

Comparison of the results of the approaches with the results of clustering performed using k-means

Clustering method	How many times is faster than k-means	How many times the number of populated clusters is less than in k-means	How many times is the maximum number of descriptors in the cluster larger than in k-means
MDC without optimization	2 178 006.2 360 229	20.705 17.049	110.86 78.07
MDC (DB)	89.727 32.643	3.451 2.834	14.42 10.099
MDC (RAM)	12 389.114 2 359.064	3.448 2.834	14.42 10.099
PQ (IMI)	503.248 341.061	3.54 2.809	37.62 20.38

This highlights how other approaches perform worse in terms of clustering quality but achieve higher speed. The first value in each table cell represents the result when all descriptors from the dataset were used. The second value shows the result when only 20% of the descriptors were initially used, followed by loading the entire dataset.

The summary from Table 4 is as follows: classical clustering methods such as k-means show the best result in terms of cluster occupancy, offering approximately a threefold advantage over competitors in terms of the number of populated clusters and about 20 times in terms of the maximum number of descriptors in a cluster. It requires much more time than its competitors to perform clustering.

Since the variants of MDC and PQ (IMI) implementations demonstrate almost the same results in terms of cluster occupancy, we can conclude that these results directly depend on the properties of a particular type of image descriptor and dataset. The obtained results do not contradict the theoretical estimates of the computational complexity. The clustering performed using MDC with DINOA-in-RAM optimization demonstrated the shortest clustering timeamong similar techniques and was not inferior in quality to the PQ (IMI).

The distribution of descriptors across clusters (Table 2) shows that most of the populated clusters have an expected number of descriptors in k-means clustering. No cluster has more than 100 descriptors. When applying MDC with both optimizations, most of the populated clusters have the expected number of descriptors, and the maximum number of descriptors (up to 1 000) is located in only one cluster. The number of clusters with the expected number of descriptors is higher when using PQ (IMI) than in MDC. However, 7 clusters contain more than 1 000 descriptors. This suggests that clustering using PQ (IMI) is not as uniform in terms of cluster occupancy as MDC. When clusters are formed based on 20% of the descriptors of the dataset and the entire dataset is assigned to these clusters, the distribution of descriptors across clusters significantly deteriorates when using k-means. However, with MDC and PQ (IMI), this deterioration is negligible.

5. Conclusions

In this paper, we present a method for optimizing the clustering of descriptors in the feature database of an image content-based retrieval system represented by the Multidimensional Cube (MDC) model. An abstract optimization algorithm, DINOA, and two specific implementations were developed: DINOA-in-RAM and DINOA-in-DB for storing MDC in RAM and in a relational database, respectively.

Experiments were conducted using descriptors of the type Invariant Brightness Histogram for images from the COCO2017 dataset, comparing clustering performed by MDC with optimizations, the k-means algorithm, and Product Quantization with the implementation of Inverted Multi-Index.

The results showed that if the clustering time is not critical for the retrieval system (the storage is rarely updated and clusters are not rebuilt), then using classical clustering methods and algorithms such as k-means is the best solution. The cluster formation speed is significantly worse compared to alternative methods. Alternative clustering methods should be used in other cases. When comparing the speed and quality of the formed clusters under the same conditions, using MDC with optimizations and PQ (IMI), MDC was found to deliver better results. This confirms the high efficiency of the developed optimization method and the achievement of the work's objectives.

The obtained results demonstrate that the developed method and clustering optimization algorithms provide MDC with high cluster creation and occupancy efficiency. This will allow the model to be quickly rebuilt if necessary and should ensure high search efficiency.

The future research directions are as follows:

1) conducting detailed experiments to determine the efficiency of searching in MDC using the formed clusters with the help of DINOA.

2) conducting additional optimization for descriptor clusters with a significantly higher number of descriptors than expected. During this optimization, an internal division of such a cluster into smaller ones is also performed.

Contributions of authors: conceptualization, methodology – **Kirill Smelyakov**; formulation of tasks, analysis – **Kirill Smelyakov**, **Stanislav Danylenko**; review and analysis of references, development of software, verification – **Stanislav Danylenko**; analysis of results – **Kirill Smelyakov**, **Stanislav Danylenko**, visualization, writing – original draft preparation – **Stanislav Danylenko**, writing – review and editing – **Kirill Smelyakov**.

Conflict of Interest

The authors declare that they have no conflict of interest in relation to this research, whether financial, personal, authorship or otherwise, that could affect the research and its results presented in this paper.

Financing

This study was conducted without any financial support.

Data Availability

The work has associated data in the data repository. The software source code will be made available upon reasonable request.

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

Use of Artificial Intelligence

The authors confirm that they did not use artificial intelligence technologies when creating the current work.

All the authors have read and agreed to the published version of this manuscript.

References

1. Vopson, M. M. The information catastrophe. *AIP Advances*, 2020, vol. 10, no. 8. DOI: 10.1063/5.0019941.

2. Badshah, A., Daud, A., Alharbey, R., Banjar, A., Bukhari A., & Alshemaimri, B. Big data applications: overview, challenges and future. *Artificial Intelligence Review*, 2025, vol. 57, no. 11. DOI: 10.1007/s10462-024-10938-5.

3. Clissa, L., Lassnig, M., & Rinaldi, L. How big is Big Data? A comprehensive survey of data production, storage, and streaming in science and industry. *Frontiers in Big Data*, 2023, vol. 6. DOI: 10.3389/fdata.2023.1271639.

4. Chembian, W. T., Senthilkumar, G., Prasanth, A., & Subash, R. K-means Pelican Optimization Algorithm based Search Space Reduction for Remote Sensing Image Retrieval. *Journal of the Indian Society of Remote Sensing*, 2025, vol. 53, no. 1, pp. 101–115. DOI: 10.1007/s12524-024-01994-z.

5. Jatakia, V., Korlahalli, S., & Deulkar, K. A survey of different search techniques for big data. *2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, Coimbatore, India, 2017, pp. 1-4. DOI: 10.1109/iciiecs.2017.8275939.

6. RezaAbbasifard, M., Ghahremani, B., & Naderi, H. A Survey on Nearest Neighbor Search Methods. *International Journal of Computer Applications*, 2014, vol. 95, no. 25, pp. 39-52. DOI: 10.5120/16754-7073.

7. Wu, Q., Yu, Y., Zhou, L., Lu, Y., Chen, H., & Qian, X. Storage and Query Indexing Methods on Big Data. *Arabian Journal for Science and Engineering*, 2023, vol. 49, iss. 5, pp. 7359-7374. DOI: 10.1007/s13369-023-08175-z.

8. Gupta, D., Loane, R., Gayen, S., & Demner-Fushman, D. Medical image retrieval via nearest neighbor search on pre-trained image features. *Knowledge-Based Systems*, 2023, vol. 278. DOI: 10.1016/j.knosys.2023.110907.

9. Alsmadi, M. K. Content-Based Image Retrieval Using Color, Shape and Texture Descriptors and Features. *Arabian Journal for Science and Engineering*, 2020, vol. 45, pp. 3317-3330. DOI: 10.1007/s13369-020-04384-y.

10. Li, X., Yang, J., & Ma, J. Recent developments of content-based image retrieval (CBIR). *Neurocomputing*, 2021, vol. 452, pp. 675-689. DOI: 10.1016/j.neucom.2020.07.139.

11. Tiwari, V. R. Developments in KD Tree and KNN Searches. *International Journal of Computer Applications*, 2023, vol. 185, no. 17, pp. 17-23. DOI: 10.5120/ijca2023922879.

12. Malkov, Y. A., Yashunin, D. A. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2020, vol. 42, no. 4, pp. 824-836. DOI: 10.1109/tpami.2018.2889473.

13. Liu, J., Zhao, M. & Zhan, C. Deep Representation-Based Fuzzy Graph Model for Content-Based Image Retrieval. *International Journal of Fuzzy Systems*, 2024, vol. 26, no. 6, pp. 2011–2022. DOI: 10.1007/s40815-024-01682-7.

14. Jiang, X., & Hu, F. Multi-scale Adaptive Feature Fusion Hashing for Image Retrieval. *Arabian Journal for Science and Engineering*, 2024, vol. 50, pp. 12027-12036. DOI: 10.1007/s13369-024-09627-w.

15. Chen, Y., Long, Y., Yang, Z., & Long, J. Unsupervised random walk manifold contrastive hashing for multimedia retrieval. Complex & Intelligent Systems, 2025, vol. 11, no. 4. DOI: 10.1007/s40747-025-01814-y.

16. Bano, S., & Khan, M. N. A. A Survey of Data Clustering Methods. *International Journal of Advanced Science and Technology*, 2018, vol. 113, pp. 133-142. DOI: 10.14257/ijast.2018.113.14.

17. Jégou, H., Douze, M., & Schmid, C. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2011, vol. 33, no. 1, pp. 117-128. DOI: 10.1109/tpami.2010.57.

18. Babenko, A., & Lempitsky, V. The inverted multi-index. *2012 IEEE Conference on Computer Vision and Pattern Recognition*, Providence, RI, USA, 2012, pp. 3069-3076. DOI: 10.1109/cvpr.2012.6248038.

19. Ge, T., He, K., Ke, Q., & Sun, J. Optimized Product Quantization for Approximate Nearest Neighbor Search. *2013 IEEE Conference on Computer Vision and Pattern Recognition*, Portland, OR, USA, 2013, pp. 2946-2953. DOI: 10.1109/cvpr.2013.379.

20. Ai, L., Cheng, H., Wang, X., Chen, C., Liu, D., Zheng, X., & Wang, Y. Approximate Nearest Neighbor Search Using Enhanced Accumulative Quantization. *Electronics*, 2022, vol. 11, no. 14. DOI: 10.3390/electronics11142236.

21. Danylenko, S., & Smelyakov, S. Development of a Multidimensional Data Model for Efficient Content-based Image Retrieval in Big Data Storage. *Radioelectronic and Computer Systems*, 2025, vol. 2025, no. 1. pp. 137-152. DOI: 10.32620/reks.2025.1.10.

22. *GitHub, facebookresearch/faiss: A library for efficient similarity search and clustering of dense vectors*. Available at: https://github.com/facebookresearch/faiss (accessed 09.02.2025).

23. Beckert, B., Bubel, R., Drodt, D., Hähnle, R., Lanzinger, F., Pfeifer, W., Ulbrich, M., & Weigl, A. The Java Verification Tool KeY: A Tutorial. *Lecture Notes in Computer Science*, Springer Nature Switzerland, Cham, 2024, pp. 597-623. DOI: 10.1007/978-3-031-71177-0_32.

24. *COCO, Common Objects in Context*. Available at: https://cocodataset.org/#home (accessed 09.02.2025).

Received 02.03.2025, Accepted 25.08.2025

МЕТОД ОПТИМІЗАЦІЇ КЛАСТЕРИЗАЦІЇ ДЕСКРИПТОРІВ В БАЗІ ДАНИХ ВЛАСТИВОСТЕЙ СИСТЕМИ ПОШУКУ ЗОБРАЖЕНЬ НА ОСНОВІ ВМІСТУ

С. Д. Даниленко, К. С. Смеляков

Предметом дослідження є методи групування дескрипторів зображень, які розміщуються у базі даних властивостей систем пошуку. **Метою** є розробка методу оптимізації процесу кластеризації дескрипторів у сховищі Big Data, представленого моделлю даних Багатовимірного Кубу, для подальшого використання утворених кластерів для ефективного пошуку у системах пошуку на основі вмісту зображень. **Завдання** полягає у: аналізі сучасних підходів і рішень для групування дескрипторів зображень у базі даних властивостей; формулювання проблеми методу кластеризації в Багатовимірному Кубі і вимог до його оптимізації; розробці абстрактного методу оптимізації; розробці конкретних алгоритмів оптимізації для різного типу розміщення моделі в пам'яті; розробці метрик; виконанні експериментів і порівнянні отриманих результатів з аналогами. **Методологія** включає в себе аналіз процесу формування груп за допомогою різних методів, визначення їх переваг та недоліків; застосування чисельних методів оптимізації для оптимізації процесу кластеризації; проведення експериментів з наявними в мережі Інтернет наборів даних; оцінка ефективності методу оптимізації і формування результуючих таблиць для порівняння з аналогами. Були отримані такі **результати**: розроблено метод оптимізації кластеризації дескрипторів, який використовується в моделі даних Багатовимірний Куб; розроблено алгоритми оптимізації, що реалізують цей метод для розміщення моделі в оперативній пам'яті та в базі даних. Результати формування кластерів, визначені метриками витраченого часу та наповненістю кластерів були порівняні з кластеризацією дескрипторів, виконаним за допомогою алгоритму k-means та підходу Product Quantization з реалізацією Inverted Multi-Index. Результати показали, що використання моделі з розробленими оптимізаціями демонструє ефективність кластеризації дескрипторів не гіршу, ніж при застосуванні Inverted Multi-Index та краще по витраченому часі ніж при застосуванні k-means чи Inverted Multi-Index. **Висновки**: розроблений метод оптимізації кластеризації дескрипторів значно покращує розподілення дескрипторів всередині моделі Багатовимірного Кубу та робить її гарною альтернативою для використання в системах пошуку зображень на основі вмісту.

Ключові слова: інформаційні технології; алгоритми і структури даних; сховище даних; великі дані; пошукова система; багатовимірна модель даних; метод оптимізації; кластеризація; обчислювальна складність.

Даниленко Станіслав Дмитрович – асп. каф. програмної інженерії, Харківський національний університет радіоелектроніки, Харків, Україна.

Смеляков Кирило Сергійович – д-р техн. наук, проф., зав. каф. програмної інженерії, Харківський національний університет радіоелектроніки, Харків, Україна.

Stanislav Danylenko – PhD Student, Software Engineering Department, Kharkiv National University of Radio Electronics, Kharkiv, Ukraine,
e-mail: stanislav.danylenko@nure.ua, ORCID: 0000-0002-8142-3018, Scopus Author ID: 57816229800.

Kirill Smelyakov – Doctor of Technical Sciences, Professor at the Software Engineering Department, Kharkiv National University of Radio Electronics, Kharkiv, Ukraine,
e-mail: kyrylo.smelyakov@nure.ua, ORCID: 0000-0001-9938-5489, Scopus Author ID: 57203149663.