International Journal of Computer Science and Applications,
©Technomathematics Research Foundation
Vol. 15, No. 1, pp. 45 – 64, 2018

QUERY BY APPROXIMATE SKETCH MULTIMEDIA DATABASE WITH PARALLEL QUERY PROCESSING

ROMAN STANISŁAW DENIZIAK

Kielce University of Technology, al. Tysiąclecia Państwa Polskiego 7,
Kielce , 25-314, Poland
s.deniziak@tu.kielce.pl

TOMASZ MICHNO

Kielce University of Technology, al. Tysiąclecia Państwa Polskiego 7,
Kielce , 25-314, Poland
t.michno@tu.kielce.pl

This paper presents a new CBIR system which consists of an image representation and a database structure. The main idea of our approach is based on a new object representation which consists of approximation of objects by a set of shapes, called primitives. Because storing images in the multimedia database is a very demanding task, we propose a tree-based structure which consist of two types of nodes: common nodes and data nodes. Common nodes are used to organize graphs in the same parts of the tree, whereas data nodes are used to store graphs. The proposed multimedia database architecture allows easier queries for users, supporting both example image as a query and a sketch drawn by a user. The queries are processed without unnecessary graph comparisons which reduces the time of image retrieval. Another advantage is the ability to use different lower level data storage methods.

Keywords: CBIR; query by sketch; multimedia databases.

1. Introduction

Multimedia databases are becoming more and more popular nowadays and are used in many areas. An example may be social media portals, where billions of images are stored and have to be queried in order to find the desired ones. Another example may be web searching engines which also needs to store and process very huge number of pictures. Another field of multimedia database usage is monitoring system where e.g. number plates or faces have to be recognized and information about them have to be retrieved.

This paper presents a new Content Based Image Retrieval system based on our previous researches [Deniziak and Michno (2017), Deniziak and Michno (2016), Deniziak and Michno (2015a), Deniziak et al. (2015), Deniziak and Michno (2015b)] which consists of an image representation and a database structure. The main idea of the Query by Approximate Shapes algorithm is based on a new object representation which consists of approximation of objects by a set of shapes, called primitives. In Deniziak and Michno (2016) we defined six primitives which should be sufficient for describing most objects. The primitive is described by its attribute e.g. a line slope for line segments or an

angle for arches. Moreover, relations between primitives are stored in order to represent e.g. connected shapes. Because storing images in the multimedia database is a very demanding task, we propose a tree-based structure which is effective and highly reduces the number of comparisons. Additionally, the query processing may be parallelized which also reduces the time which is needed to complete the results of a search. Our tree approach consists of two types of nodes: common nodes and data nodes. Common nodes are used to organize graphs in the same parts of the tree, whereas data nodes are used to store graphs.

The proposed multimedia database architecture allows easier queries for users, because it supports both example image as a query and a sketch drawn by a user. Moreover, the queries are processed without unnecessary graph comparisons which reduces the time of a image retrieval. Another advantage is the ability to use different lower level data storage methods like vector containers, relational databases or NoSQL data stores (e.g. SD2DS).

The paper is organized as follows: the Section 2 presents related works in the area of Image Retrieval and database structures. The Section 3 contains our motivation and assumptions which are made for the system. The Section 4 describes the object representation used in the database. The Section 5 is dedicated to the database structure and contains descriptions of inner structure, operations on nodes and queries. The Section 6 shows initial experimental results. The Section 7 presents the plans for the future works and conclusions. The last section contains bibliography.

2. Related works

The multimedia database Image Retrieval algorithms may be assigned to three types of algorithms:

- based on textual descriptions, most often keywords - Keywords Based Image Retrieval (KBIR) algorithms
- based on semantic information extracted from the image - Semantic Based Image Retrieval (SBIR) algorithms
- based on information which is present in the image - Content Based Image Retrieval (CBIR) algorithms

The Keyword Based Image Retrieval algorithms use textual annotations in order to describe the whole image or their parts. Most often descriptions are made by humans and the precision of keywords is limited to the knowledge and perception of a person [Deniziak and Michno (2015a)]. For objects which are well known it is easy to represent them by annotations and the results of retrieval are very satisfactory. For example, a car object may be named very precisely by the brand, model name, version, production year and color. When the object is not well known, or it is not easy to describe it precisely by keywords, the results may be imprecise. This is due to the fact that annotations are very subjective and different person may use different words as keywords for the same objects [Li and Hsu (2008), Wang et al. (2010)]. For example, a landscape with trees and water may be annotated by one person as a forest and a river, but by another one as trees and a

lake. The third person contrary may use the name of place where the photo was taken. In this situation the results of a query may be imprecise and unsatisfactory for the user. Another disadvantage of the KBIR approach is that it is hard to automatically add keywords without human interaction.

The Semantic Based Image Retrieval algorithms are similar to Keyword Based Image Retrieval algorithms because they use also words to perform queries. However, contrary to them, they allow users to write queries as phrases which are more natural form for them. Such different query interface is used in order to overcome the so called 'semantic gap' which is a difference between what a human could describe and what is present in the image [Wang et al. (2010)]. After defining by a user, the phrases are mapped onto so called semantic features which are correlated with the content of the image [Li and Hsu (2008)]. The use of semantic based textual approach is more comfortable and easier for users but still if they does not have the full knowledge about searched images the results may be insufficient. There are also approaches which uses graphical queries which are then transformed into textual description. One of the most interesting research is Li et al. (2016) which uses a sketch as a query, then extracts textual annotations - semantic features and then finds 3D models of objects which are described by similar or the same set of features.

The Content Based Image Retrieval algorithms use information present in the image to perform queries [Deniziak and Michno (2015a)]. In this area two types of algorithms could be distinguished: low-level and high-level [Deniziak and Michno (2016)]. The first type of algorithms is based on extraction of features for the whole image. There may be statistical image features used, e.g. a normalized color histogram [Mocofan et al. (2011)]. Other methods may be a difference moment and entropy [Kriegel et al. (2006)], a spatial domain image representation [Shih (2002)] or a bag of words histogram [Śluzek (2016)]. There are also approaches which use different MPEG-7 descriptors, e.g. shape and texture descriptors [Lalos et al. (2008)]. Since the features describes the whole image, the low-level CBIR algorithms provide very satisfactory results when a query is performed in order to find similar images. However, when a user would like to obtain images with the same object but with different backgrounds, the low-level algorithms are not efficient and the results may be insufficient. The high-level CBIR algorithms are more suitable for that situations. Their main idea is to separate objects from the background and other parts of the image. Most often the region extraction method is used [Deniziak and Michno (2016)] which is based on gathering similar groups of pixels into uniform areas which are then transformed into a graph, storing the mutual relations between nodes. In order to extract regions, methods based on e.g. color thresholds, moment-based local operators [Śluzek (2005)] or fuzzy patterns recognition [Bielecka and Skomorowski (2007)] may be used. The query process is strictly based on searching subgraphs between a graph which was extracted from the stored image and a graph stored in the database. The main disadvantage of the region-based algorithms is the need of query image which have to store many details. If a user does not have a proper one, it must be prepared which may require drawing skills.

There are also CBIR algorithms which allows performing queries without having the full knowledge about the searched objects. There are algorithms which are low-level e.g. [Kato et al. (1992)]. The approach presented by authors is based on human drawn sketches which are then transformed into lower resolution images and compared with sketches in the database using edge detection techniques. The method provides good results, but is oriented on finding similar paintings, which is not sufficient for querying by objects. [Zhang et al. (2016)]. Other researches use global contour map and salient contour map in order to extract objects and compare them with images in the database. There are also researches which use additionally relevant feedback (e.g. SIFT algotihm) and re-ranking to improve the results precision [Qian et al. (2016)]. In our previous researches [Denizak et al. (2015), Deniziak and Michno (2016), Deniziak and Michno (2015a), Deniziak and Michno (2015b)] we proposed a high-level algorithm which is based on decomposing object into its approximated by predefined shapes representation (Query by Approximate Shape). The shapes are used for creating a graph which is then compared with other graphs stored in the database. In this paper we describe the database structure based on the Query by Approximate Shape method.

All images or objects representations have to be stored in an efficient way. Most often a structure based on cells or trees are used [Lalos et al. (2008)]. One of the most interesting approaches is Kiranyaz and Gabbouj (2007) research which is based on a tree storing cells. The similar images are stored in the same cells, but when the similarity between images is below the determined threshold, another cell is created (a process in the paper called a mitosis). In Deniziak et al. (2015) we proposed the first attempts for the database structure for our object representation which was based on Scalable Distributed Two-Layer Data Structures. The approach provided good results but was prone to rapid tree height increase. Moreover, we would like to prepare the database structure which would be more universal and allow to use different data structures types in the lower implementation level (e.g. SD2DS, but also data containers or relational databases). This would increase the number of possible applications e.g. to use our database on devices with very small resources. Our database structure in some parts is based on the same ideas as [Kiranyaz and Gabbouj (2007)] (e.g. storing similar data in the same part of the tree), but we use different inner structure and different methods of object representation and querying.

3. Motivation

The querying and storing images in the multimedia database is a very complicated problem. Most often there is a very huge amount of data to store, which may require very efficient methods of storage. Moreover, the proper querying also needs some efficient algorithms. During our research, we developed some initial attempts to provide some solutions like in Deniziak et al. (2015), but we decided to provide more universal database structure which allows using different implementations, e.g. based on relational databases or Scalable Distributed Two-Layer Data Structures. The motivation of our research is to achieve the system which will fulfill the following requirements:

- two methods of performing queries by users - by an image example and by a sketch
- easy addition of new images
- fast query processing
- ability to parallelize query processing
- ability to use different lower level implementations of graphs storage (e.g. using vectors, relational databases or SD2DS) in order to customize the database to the current requirements

4. Object graph

The idea of our approach is based on using line segments and arches as an object's representation. During our experiments with querying multimedia databases we observed that most objects may be simplified to resemble sketches drawn by a human and still they provide good searching results. In order to provide object simplification, firstly we proposed to approximate an object using line segments and ellipses [Deniziak and Michno (2015a)] and then line segments, arches, polylines, polyarches, polygons and arc-sided polygons which we called primitives (Fig. 1) [Deniziak and Michno (2016)]. Because each primitive has to be stored in a separation from image resolution and its position, we proposed to use following primitive's description called attribute [Deniziak and Michno (2016)]:

- each line segment may be described using its slope angle (Fig. 2 a)
- each arc may be described using its angle and additionally the relation of its radius to the whole object (Fig. 2 b)
- each polyline, polyarc, polygon and arc-sided polygon may be described using the number of segments and the corresponding attributes of each segment

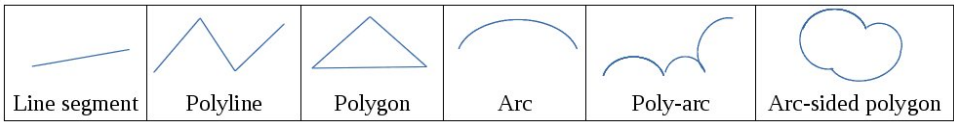

Fig. 2. The primitives used to describe objects.

Fig. 1.The attributes used for line segments (a) and arches (b).

The Content Based Image Retrieval systems most often use an example image as a query. However, there are also algorithms which allows users to query the system using a sketch. Both types have their advantages, therefore we designed our system in order to support both images and hand drawn sketches for query input. Thanks to that, a user may

choose which interface would be more suitable for the specific query or his or her drawing skills. The image query may be useful when an user have the image of an object. The sketch query may be performed very quickly without high drawing skills, using predefined shapes. Additionally, queries performed by some automatic systems (like e.g. monitoring systems) may be supported using captured frame by cameras as a query.

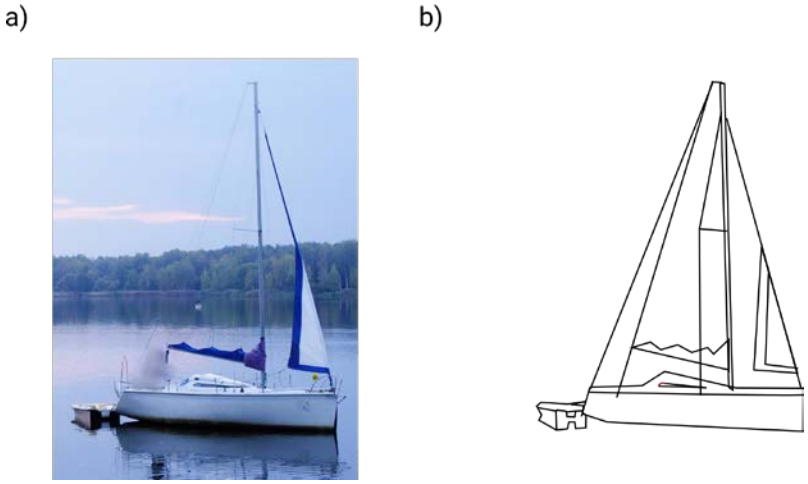

Fig. 3. The example sailboat object representation: a) an image, b) an object drawn with lines(black color) and arches (red color).

When using an image as a query, firstly the line segments and arches are extracted. In our implementation we used LSWMS algorthm and Circular Hough Transform. The Circular Hough Transform only detects circles, thus, all detected line segments are checked if they may construct an arc, checking angles between them. After the simplest shapes detection, more complex are constructed: from line segments - polylines and polygons, from arches - polyarches and arc-sieded polygons [Deniziak and Michno (2017)]. When the query is a hand-drawn sketch, all primitives are drawn like in a vector graphics, storing all information about position and size. All detected or drawn primitives are then transformed into a graph which stores the mutual correlations between them which are used during query process to compare them with other graphs in the database. Additionally, a graph may contain also metadata like the filename or textual description.Queries are based on comparing query graph with graphs stored in the database. As a measurement of two graphs silimilarity, a coefficient called similarity is used:

$$similarity \in <0,1> \quad (1)$$

If two graphs contain the same number, types and attributes of primitives, the similarity takes the value 1. If they have completely different set of primitives, the

coefficient is equal to 0. All other situations are covered by fraction values, from very small similarity (similarity→0) to very high similarity (similarity→1).

5. The structure of the database

In order to store graphs in a more efficient way, we propose to use a tree-based database structure. The tree is designed using three types of nodes:

- root node - which is only used as an entry point to the tree
- nodes which stores graphs which describes images - called data nodes
- nodes which are used to gather similar graphs in the same subtrees and data nodes - called common nodes

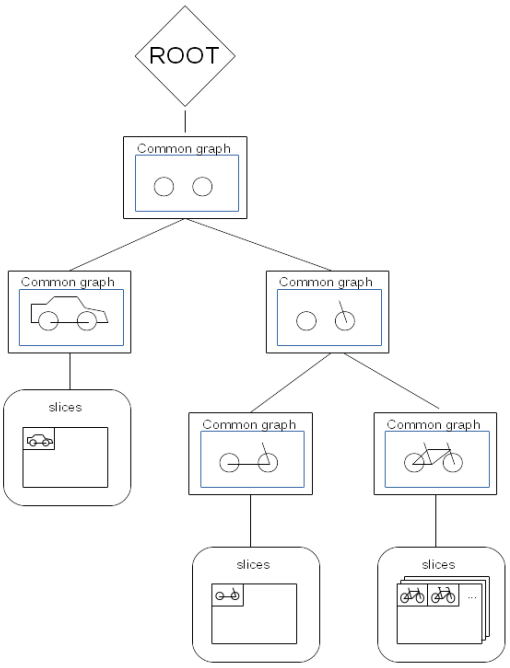

Fig. 4.The overview of the tree database structure

As described above, the tree tries to gather similar graphs in the same part of it. When graphs are very similar (up to a chosen threshold), they are stored in the same data node,

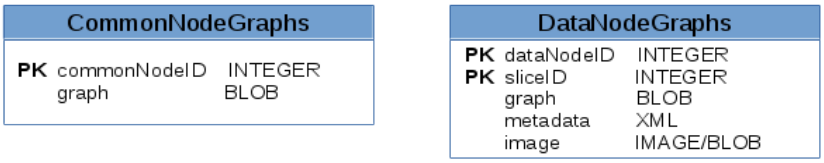

Fig. 5.Tables used in the SQL-based implementation.

which causes that a data node may contain a small subset of graphs. Thanks to that, the height of the tree does not grow very fastly. If the similarity is high, but not enough, a

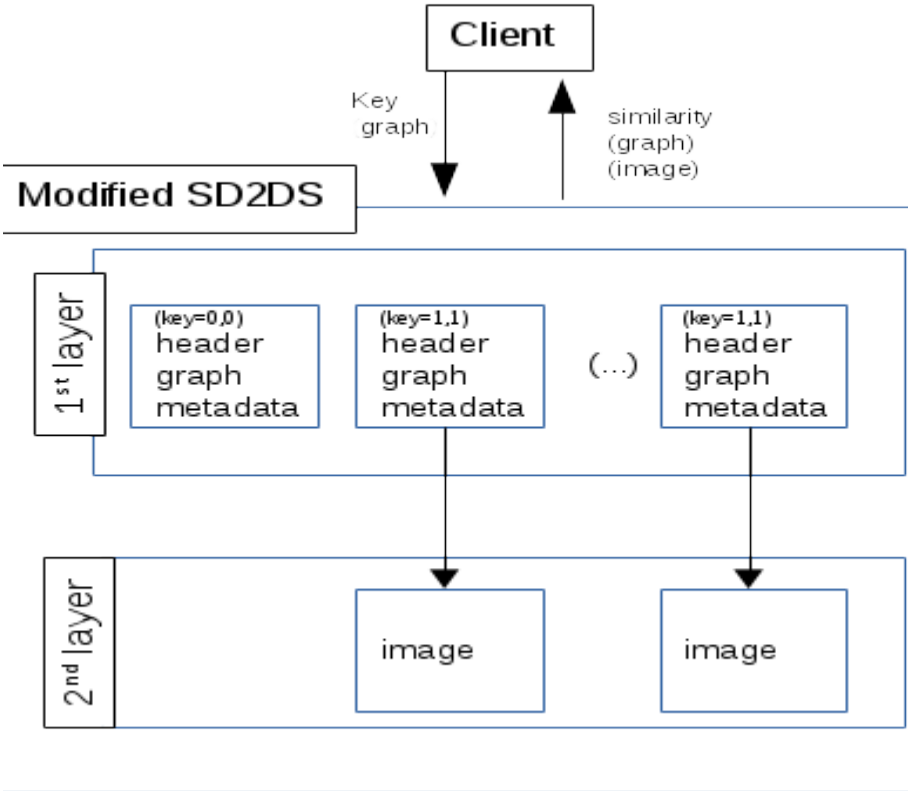

Fig. 6.The overview of the modified SD2DS data store.

new data and common nodes are created. The data node stores the graph and the common node stores a newly created graph which contains only similar parts of these two data nodes. Thanks to that, during query, some parts of the tree may be skipped very early

without unnecessary comparisons which highly increase the efficiency and time of query processing. The structure and different types of nodes are shown in the Fig. 4. The figure shows an example database which stores a car, a scooter and a bicycle graphs. The first tree element, the root node, is used to store the element from which the tree should be constructed or queried. Next, we can see the common graph stored in the common node which contains only the common parts of each graphs - their wheels (which are strictly two circles). Its children are two other common graphs which contains a car graph and a common parts of a scooter and a bicycle graphs (wheels circles and a steering wheel line segment). Since there is no other car graph, the data node is created and attached to the car common node. Similarly, as previous, there are also two common nodes for a scooter and for the bicycle graphs which have their data nodes attached. In the figure, in the data nodes there could be noted the "slices" word which stands for inner data node organization. Each data node stores graphs in as vectors. Because the structure of the database should be independent from the low-level data storage method and it should provide parallelism in e.g. queries, we decided that graphs vector should be stored in so-called slice which is strictly an array of vectors. Thus, there may be more than one vectors which may be e.g. processed independently. The vector may have maximum capacity, and when it is achieved, the next vector slice (an element of vectors array) is created (Alg. 1).

5.1. *Graphs storage*

The database was designed in a higher level in order to allow using different implementations to store graphs or even tree nodes without modifying the algorithms. In this paper we present 3 different methods: vector-based, SQL-based and SD2DS-based.

The first implementation, vector-based, is based on using vectors for storing slices in data nodes. Moreover, a slice could be also a vector of graphs. This implementation was used in our prototype application. Its advantages are simplicity, speed and no need of external applications/systems. Contrary, it strongly depends on the available RAM memory, which may be problematic for huge number of graphs and images connected with them.

The SQL-based implementation allows storing graphs and metadata (e.g. images connected with them) in the relational database, e.g. MySQL. All types of nodes, common and data nodes, stores theirs graphs in the database, in appropriate tables (for data nodes DataNodesGraphs table, for common node CommonNodesGraphs table - Fig. 5). The graph may be stored in two ways - as a binary large object (blob) or as a serialized text, which after retrieving should be deserialized. During a query when comparison with graph should be performed, firstly the query to the SQL database is performed (for common nodes using their id as a where clause, for data nodes using the node id and slice id) and then retrieved graph is being used for tests. The advantage of this implementation is easy adoption of the existing SQL-based image database and persistence of the stored graphs.

The SD2DS-based implementation is similar to the SQL-based approach and is based

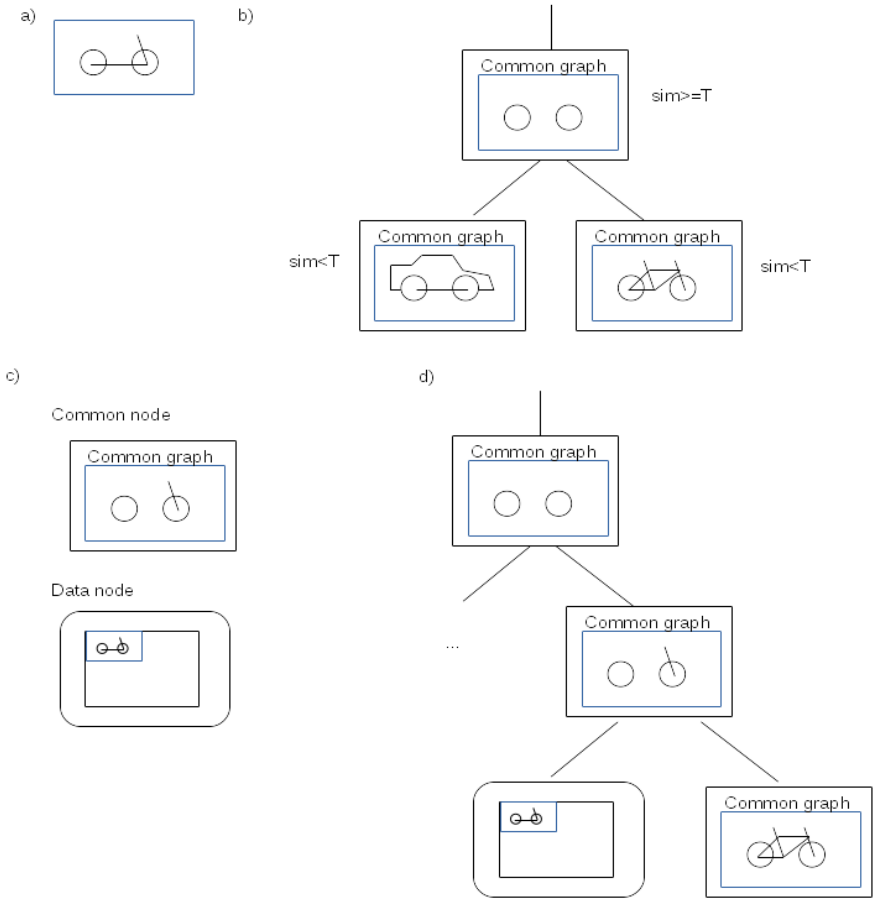

Fig. 7.The insertion of a new node with graph to the tree: a) the graph which has to be inserted, b) the tree structure, c) two types of nodes after creation, d) the tree after insertion of the nodes from c)

on storing graphs (from common and data nodes) in the Scalable Two-Layer Data Structures which is a highly scalable and efficient NoSQL data store [Deniziak and Krechowicz (2017)]. In this paper we propose to use modified data store architecture which in the first layer stores the graph and its metadata and in the second layer stores images connected with graphs (for data nodes). As a SD2DS data key, the merge of node id and slice id is used (for common nodes as a slice id may be used ’0’ value). Moreover, we propose to implement the graphs comparisons in the first SD2DS layer which improves the efficiency (e.g. because of ability to distribute the data to different computer nodes). When a comparison of graphs should be performed, e.g. during query, there is a message sent to the SD2DS with the key and the graph to compare. Then the data store

performs the comparison and as a result, depending on a type of a graph (a common node graph or a data node graph), the similarity and optionally for data nodes the graph with an image attached to it is returned. The advantage of the implementation based on SD2DS is scalability without the need of shutting down the whole system, speed of processing and retrieving data (thanks to the RAM data storage and parallelism in conjunction with distribution).

5.2. *Inserting a new graph into the tree*

The graph (g) insertion to the database starts from the root element and then all its children are compared with the graph g. The comparison algorithm tries to find the corresponding nodes between two graphs and then it computes the graphs similarity dividing all matched nodes similarities by the minimum number of nodes in both graphs. When the similarity is smaller than chosen threshold, a new common and data node are created and attached to the root element. When the similarity is equal or higher than chosen threshold, the comparison with all children of the proper common graph is performed. When more than one similarity with common graph equal or higher than threshold is obtained, the children of the highest similarity are chosen for the next comparisons. Such tree traversal is carried out until a data node is reached and the new graph g is being inserted into the suitable slice, leaving the most similar graph to the parent common graph as a first element in the first slice and the least similar as the last element in the last slice. The new graph addition process is shown in the Fig. 7 and in the Alg. 2.

Algorithm 1 Splitting a vector of graphs when the maximum size is reached

Ensure: T_s - maximum number of graphs in the slice; d_h - data node, vec – vector with graphs
vSize ← size(vec);
2: **if** *vSize* > T_s **then**
create new vector *vec2*;
4: copy into *vec2* graphs from *vec*[T_s] to *vec*[vSize];
remove *vec*[T_s] to *vec*[vSize];
6: add *vec2* to d_h;
end if

Algorithm 2 Inserting a new graph into the tree

```
Ensure: g - the graph which has to be inserted; T_sim - minimal similarity of graphs
    node ← root;
2: while node is not NULL do
        traverse ← true;
4:      if node is a data node then
            compare g with first graph in first slice of node [compare graphs using Alg. 3];
6:          insert g into node in the position related to the similiarity;
            exit
8:      end if
        if node is not a root then
10:         sim ← compareCommon(g, node->commonGraph) [compare graphs using Alg. 3];
            if sim < T_sim then
12:             traverse ← false;
            end if
14:     end if
        if traverse then
16:         compare all node's children common graphs with g, choose maximum similarity as sim
[compare graphs using Alg. 3];
            if sim < T_sim then
18:             traverse ← false;
            else
20:             node ← child with max sim;
                continue;
22:         end if
        end if
24:     if !traverse then
            commGraph ← common part of g and node's commonGraph;
26:         create new common node th
            insert commGraph into th
28:         create new data node dh and add as child to th
            insert g into dh
30:         add node as child to th
            exit
32:     end if
    end while
```

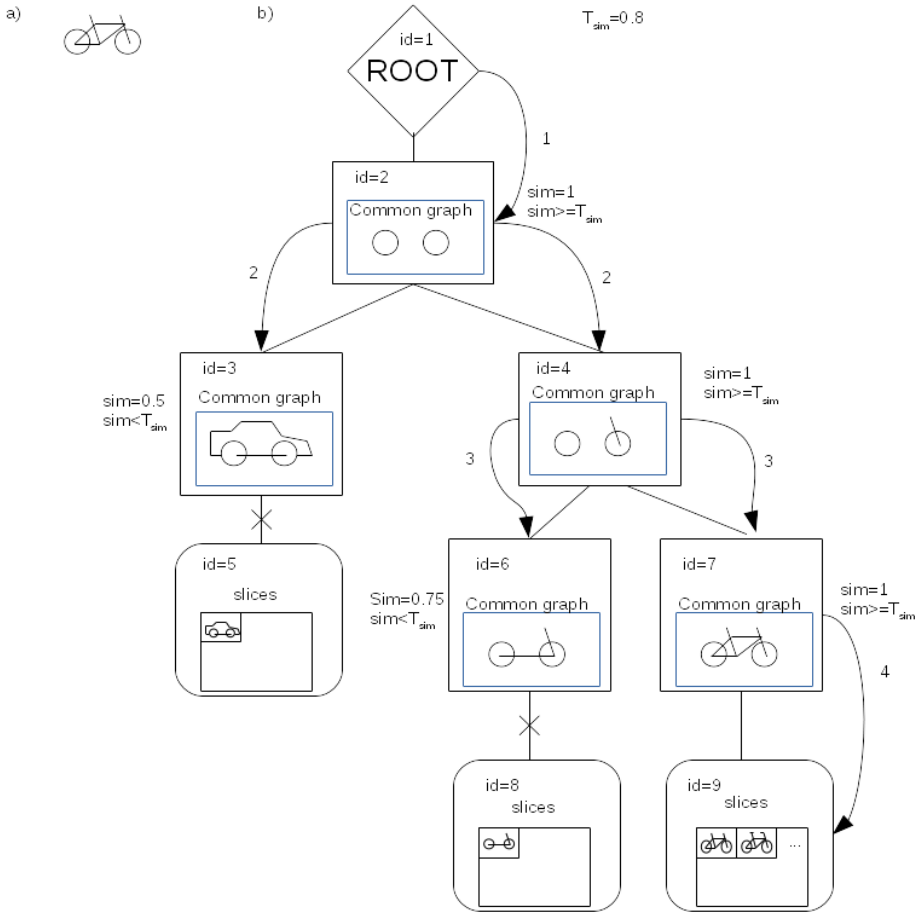

Fig. 8.The example query tree traversal: a) the query object graph, b) the query

5.3. *Performing queries*

Performing queries in the proposed database is based on traversing the tree abandoning not similar sub trees which is much faster than linear comparing with each stored graph. Similarly, to graph insertion, the whole process starts with the root element and comparisons with its children (common graphs). For each common graph with similarity equal or higher than chosen threshold, tests of all children are performed recursively until a data node is reached or the similarity is lower than threshold. When the data node is achieved, all slices are tested. For each slice, the first and last graph is compared with the query graph. When all of them have similarity equal of higher than threshold, all graphs from slice are added to the results set. Contrary, when the condition is not met, then searching similar to the binary search is performed in order to find the

Algorithm 3 Comparing a graph which has to be inserted with the graph in the tree node

Ensure: g_i - the graph which has to be inserted; g_{db} - graph which has to be matched to g_i (the graph which is stored in a tree); T_{conn} - minimal similarity threshold for connections test

```
    countNodes ← number of nodes in g_i;
2: for each node_gi in g_i do
       for each node_gdb in g_db do
4:         sim_gi,gdb ← 0
           if nodes types are different then
6:             continue;
           end if
8:         sim_Conn ← how many connections to other nodes in g_i has the same type as in g_db ;
           sim_Conn ← sim_Conn ÷ countNodes;
10:        if simConn < T_conn then
               continue;
12:        end if
           sim_Prim ← the similarity of primitives stored in nodes (returned by Alg. 4);
14:        try to match all connected nodes to node_gdb onto the counterparts in node_gi checking the
similarity of primitives stored in nodes (by Alg. 4) and relative positions to other nodes, store the
similarity result in simPos;
           sim_gi,gdb ← sim_Conn·simPrim·simPos; store as similarity between node_gi
and node_gdb ;
16:    end for
    end for
18: sim ← 0
    for each node_gi in g_i do
20:        choose the match with nodes in g_db with highest sim_gi,gdb value and add to sim ;
    end for
22: sim ← sim ÷ min( number of nodes in g_i, number of nodes in g_db ;
    return sim;
```

subset of the graphs in the vector slice (Alg. 6). The query algorithm is presented in the Alg. 5. The query process will be shown using example database described in the previous subsection. As a query graph, a graph of a bicycle b will be used Fig. 8 a). The first algorithm step, as mentioned earlier, is comparison with root child (Fig. 8 b) arrow 1, node id=2). In this example, the similarity (sim) is higher than chosen threshold, therefore next children are tested (Fig. 8 b) arrow 2, nodes with id=3 and id=4). The similarity with common node with id=3 was smaller than threshold, thus this subtree is going to be omitted during further tree traversal. The comparison with common node with id=4 gives similarity equal 1, so all of its children are going to be checked (Fig. 8 b) arrow 3, nodes with id=6 and id=7). The comparison with common node with id=6 does not met the threshold requirements, so this subtree is going to be omitted. The comparison with common node with id=7 gives the similarity equal to 1, which causes that its children is going to be tested (Fig. 8 b) arrow 4, node with id=9). Because node with id=9 is a data node, all slices are compared with the query graph. In the example there is only one slice and the first and last graph in it is similar enough with the query, thus all graphs from it are added to the return result set.

Algorithm 4 Comparing graphs nodes between each others. As a result the similarity coefficient is returned (values: <0,1>).

```
Ensure: pa, pb - primitives to compare;
  if nodes types are different then
2:    return 0
  end if
4: if nodes types are line segments then
      diff ← |angle slope of pa − angle slope of pb|
6:    return sim ← 1 − diff
   end if
8:  if nodes types are arches then
      diff ← |angle of pa − angle of pb|
10:   return sim ← 1 − diff
    end if
12: if nodes types are polylines, polygons, polyarches or arc-sided polygons
    then
      diff ← |number of segments in pa − number of segments in pb|
14:   try to match all segments between pa and pb, choosing the smallest difference of their
attributes, sum all corresponding differences and add to diff
      sim ← (1 + minimum number of segments(pa, pb)) − diff
16:   if sim > 1 then
            sim ← 1
18:         return sim
      end if
20: end if
```

Algorithm 5 Querying the database

```
Ensure: g - the query graph; T_sim - minimal similarity of graphs; stack - a stack which is used to store
nodes to check;
  put root into the stack;
2: while stack is not empty do
      node ← pop element from stack
4:    if node is a data node then
            choose all graphs from node slices using Alg. 6;
6:          continue;
      end if
8:    if node is not a root then
            sim ← similarity of g and common graph in node [compare graphs using Alg. 3];
10:         if sim >= T_sim then
                put all node children to the stack;
12:         end if
            else
14:             put all node children to the stack;
            end if
16:   end if
    end while
```

5.4. *Parallelizing the query processing*

Thanks to that, the processing time may be highly reduced. Each comparison of nodes described in the previous subsection may be performed using different threads. In this paper we propose to use a pool of thread, because it also reduces the time due to the omitting the thread creation process. The parallel query algorithm is described in the Alg. 7.

5.5. *Deleting graphs from the database*

The graph which should be deleted from the database could be in two situations. First is very easy to perform, because when there are many graphs in the slice or it is the only one, but there are other slices, it may be removed without any other actions. The second

Algorithm 6 Querying the slice in data node

Ensure: *g* - the query graph; *slices*[1..n][1..m] - the *n* slices which stores vectors of *m* graphs; T_{sim} - minimal similarity of graphs;

```
   for each slice in slices do
2:     L ← slice[1];
       R ← slice[m];
4:     while L <= R do
           sim_L ← similarity of g and first graph in slice[L] [compare graphs using Alg. 3];
6:         sim_R ← similarity of g and last graph in slice[R] [compare graphs using Alg. 3];
           if sim_L >= T_sim and sim_R >= T_sim then
8:           add all graphs from slice between L and R indexes into the result set;
             break while loop;
10:        else
             if sim_L >= T_sim then
12:              L ← L + 1;
             end if
14:          if sim_R >= T_sim then
                 R ← R − 1;
16:          end if
           end if
18:    end while
     end for
```

situation occurs when the graph is the only one in the data node. In this case, the data node and its parent common node should be removed.

6. Experimental results

Our database structure and object representation were tested using an implementation written in C++ with OpenCV library image processing. As images used for tests we used images of cars, motorbikes, bicycles and scooters. For the initial test 111 number of images was used. As a measurement of quality of retrieved results, two common known coefficients were used: precision and recall. The precision is defined as a number of relevant results images divided by the total number of results. The recall is defined as the number of relevant results images divided by the total number of relevant images in the database. The chosen results are shown in the Table 1. During experiments we observed that some object classes have lower precision results which was caused by high similarity with some other classes (for example the scooter and bicycle classes). Moreover, there

are also some classes with lower recall values which was caused by many differences between object images and usage of real life images.

Algorithm 7 The parallel query

Ensure: *g* - the query graph; *ThreadPool* - the pool of threads; as a node firstly the root is used; *results* - the set of query results graphs for the function instance

```
   FUNCTION query(node):
2:     if node is a data node then
           add all graphs from slices using algorithm mentioned in the previous subsection to the
results set
4:     else
           compute the similarity sim of g and node
6:         if sim >= T_sim then
               T ← first free thread from ThreadPool;
8:             for each child in all children of node do
                   subresults ← execute on T query(child)
10:                add subresults to the results
               end for
12:        end if
       end if
14: return results
```

Table 1. The precision and recall results for each class of objects

object class	precision	recall
a bicycle	0.93	0.37
a scooter	0.67	1.0
a motorbike	0.86	0.40
a car	0.86	0.73

Table 2. The query time for different number of graphs (in microseconds)

query algorithm type	number of graphs				
	23	**110**	**173**	**282**	**596**
tree parallel	55	362	1534	3915	8377
tree not parallel	101	797	3442	8892	18814

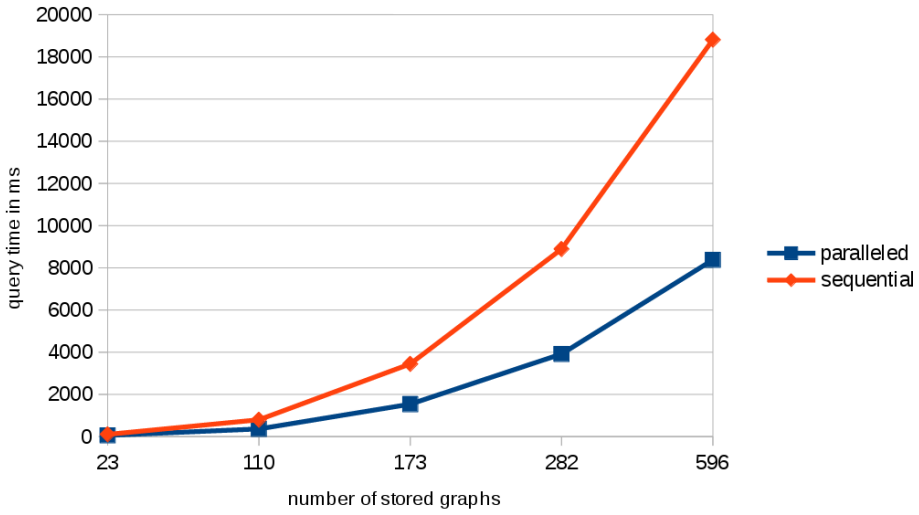

Fig. 9.The differences between query execution time for parallel and not parallel query.

In order to test the efficiency of paralleling the query processing we implemented such ability in our prototype application. We performed test in comparison querying using our database tree structure but without parallelism for 23, 110, 173, 282 and 596 graphs. The test results are presented in the Table 2 and Fig. 9. The test were performed on a computer with 4 thread and 2 cores CPU. In average, the parallel query was about 2 times faster than query the tree without parallelism. It may be noticed that if the CPU have more cores, the differences could be much higher. Moreover, the GPGPU computing could be also used.

7. Conclusion and future works

This paper presents a new CBIR Query by Sketch database structure. Our approach is based on object representation proposed in [Deniziak and Michno (2017)] which is based on decomposing an object into simple shapes, called primitives. There are two basic primitives: line segments and arches. Based on them, more complex shapes are constructed: polylines, polygons, polyarches and arc-sided polygons. After object decomposition, a graph is created which stores the mutual relations between primitives. All graphs are stored in the database which has a tree structure. There are two types of tree nodes defined: common nodes and data nodes. Common nodes are used to organize the structure of the tree and to gather similar graphs in the same parts of it. The data nodes are used to store graphs. Thanks to the tree structure, the query may be processed faster, because some parts of the tree are abandoned very fastly. Moreover, the parallelisation of the processing may be done which was also tested. Additionally, the database structure allows using different implementations of the lower level graphs

storage (e.g. based on vectors, sql relational database or SD2DS data structures), adapted to the current requirements.

As a future research we plan to test different implementations of the database with higher number of graph images. Additionally, some works on improving the recall coefficient should be done. Another direction of the future research is testing different algorithms for graphs comparisons, e.g. using optimization methods with constrains [Sitek and Wikarek (2016)].

Acknowledgments

The research used equipment funded by the European Union in the Innovative Economy Programme, MOLAB - Kielce University of Technology.

References

Bielecka, M.; Skomorowski, M. (2007) Fuzzy-aided Parsing for Pattern Recognition. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 313–318. ISBN 978-3-540-75175-5. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-75175-5_39

Deniziak, R.S.; Krechowicz, A. (2017): "New content based image retrieval database structure using query by approximate shapes," in Communication Papers of the 2017 Federated Conference on Computer Science and Information Systems, M. Ganzha, L. Maciaszek, M. Paprzycki (eds). ACSIS, 13, pp. 177–182. doi: 10.15439/2017F195

Deniziak, R.S.; Michno, T. (2015): Query by shape for image retrieval from multimedia databases. Kozielski, S., Mrozek, D., Kasprowski, P., Malysiak-Mrozek, B., Kostrzewa, D. (eds.) Beyond Databases, Architectures and Structures. CCIS, Springer, Ustroń, **521,** pp. 377–386. doi: 10.1007/978-3-319-18422-7 33

Deniziak, R.S.; Michno, T. (2015): Query-by-shape interface for content based image retrieval. 2015 8th International Conference on Human System Interaction (HSI), IEEE, Warsaw, pp. 108–114. doi: 10.1109/HSI.2015.7170652

Deniziak, R. S.; Michno, T. (2016): Content based image retrieval using query by approximate shape. In: 2016 Federated Conference on Computer Science and Information Systems (FedCSIS), IEEE, Gdańsk, pp. 807–816. doi: 10.15439/2016f233

Deniziak, R. S.; Michno, T. (2017): New content based image retrieval database structure using query by approximate shapes. 2017 Federated Conference on Computer Science and Information Systems (FedCSIS), IEEE, Prague, pp. 613–621. doi: 10.15439/2017F457

Deniziak, R.S.; Michno, T.; Krechowicz, A. (2015): The scalable distributed two-layer content based image retrieval data store. 2015 Federated Conference on Computer Science and Information Systems (FedCSIS), IEEE, Lódź, pp. 827–832. doi: 10.15439/2015F272

Kato, T.; Kurita, T.; Otsu, N.; Hirata, K. (1992): A sketch retrieval method for full color image database-query by visual example. 11th IAPR International Conference on Pattern Recognition, IEEE, The Hague, pp. 530-533. doi: 10.1109/ICPR.1992.2016167

Kiranyaz, S.; Gabbouj, M. (2007): Hierarchical cellular tree: An efficient indexing scheme for content-based retrieval on multimedia databases. Multimedia, IEEE Transactions on, 9(1), pp. 102–119.

Kriegel, H. P.; Kroger, P.; Kunath, P.; Pryakhin, A. (2006): Effective similarity search in multimedia databases using multiple representations. 2006 12th International Multi-Media Modelling Conference. IEEE, Beijing. doi:10.1109/MMMC.2006.1651355

Lalos, C.; Doulamis, A.; Konstanteli, K.; Dellias, P.; Varvarigou, T. (2008): An innovative content-based indexing technique with linear response suitable for pervasive environments. 2008

International Workshop on Content-Based Multimedia Indexing, IEEE, London (London), pp. 462–469. doi: 10.1109/CBMI.2008.4564983

Li, C.-Y.; Hsu, C.-T. (2008): Image retrieval with relevance feedback based on graph-theoretic region correspondence estimation. IEEE Transactions on Multimedia, IEEE, 10(3), pp. 447-456., doi: 10.1109/tmm.2008.917421

Li, B.; Lu, Y.; Shen, J. (2016): A semantic tree-based approach for sketch-based 3d model retrieval. In 2016 23rd International Conference on Pattern Recognition (ICPR), EEE, Cancun, pp. 3880-3885. doi: 10.1109/ICPR.2016.7900240

Mocofan, M.; Ermalai, I.; Bucos, M.; Onita, M.; Dragulescu, B. (2011): Supervised tree content based search algorithm for multimedia image databases. 2011 6th IEEE International Symposium on Applied Computational Intelligence and Informatics (SACI), IEEE, Timisoara, pp. 469-472. doi: 10.1109/SACI.2011.5873049

Qian, X.; Tan, X.; Zhang, Y.; Hong, R.; Wang, M. (2016): Enhancing sketch-based image retrieval by re-ranking and relevance feedback. IEEE Transactions on Image Processing, 25(1), pp. 195–208. doi: 10.1109/TIP.2015.2497145

Shih, T. K. (2002): Distributed multimedia databases, T. K. Shih, Ed. Hershey, PA, USA: IGI Global, 2002, ch. Distributed Multimedia Databases, pp. 2–12. ISBN 1-930708-29-7. [Online]. Available: http://dl.acm.org/citation.cfm?id=510695.510697

Sitek, P.; Wikarek, J. (2016): A hybrid programming framework for modeling and solving constraint satisfaction and optimization problems. Scientific Programming, Article ID 5102616. (2016). doi:10.1155/2016/5102616

Śluzek, A. (2016): Machine vision in food recognition: Attempts to enhance CBVIR tools. Ganzha, M., Maciaszek, L. A., Paprzycki, M. (eds.) Position Papers of the 2016 Federated Conference on Computer Science and Information Systems, FedCSIS. PTI, Gdańsk (2016). doi: 10.15439/2016f579

Śluzek, A. (2005): On moment-based local operators for detecting image patterns. Image and Vision Computing, 23(3), pp. 287 – 298.

Wang, H. H.; Mohamad, D.; Ismail, N. A. (2010): Approaches, challenges and future direction of image retrieval. In: Journal of Computing, New York, 2(6).

Zhang, Y.; Qian, X.; Tan, X.; Han, J.; Tang, Y. (2016): Sketch-based image retrieval by salient contour reinforcement. IEEE Transactions on Multimedia, 18(8), pp. 1604–1615. doi: 10.1109/TMM.2016.2568138