Detecting Intrinsic Dissimilarities in Large Image Databases through Skylines

Nikolaos Georgiadis
Department of Informatics
Aristotle University
54124 Thessaloniki, Greece
nikosgeorgiadis@gmail.com

Eleftherios Tiakas
Department of Informatics
Aristotle University
54124 Thessaloniki, Greece
tiakas@csd.auth.gr

Yannis Manolopoulos
Department of Informatics
Aristotle University
54124 Thessaloniki, Greece
manolopo@csd.auth.gr

ABSTRACT

In this paper we try to detect dissimilar images in image databases without defining a similarity ranking function by capturing the intrinsic dissimilarities of image descriptor vectors. To this end, we apply the skyline operation using their multi-dimensional descriptor vectors. We implemented a number of skyline methods, combined with four hashing state-of-the-art algorithms for data partitioning to create efficient indexing to secondary memory. We compared and evaluated their results by using two real image datasets to measure the performance and the effectiveness of the implemented algorithms. Detailed results show the efficiency and effectiveness of our approach.

CCS CONCEPTS

• **Information systems** → **Image search**;

KEYWORDS

image databases, skyline algorithms, hashing methods

1 INTRODUCTION

Every minute more than 20 hours of video are uploaded to YouTube and 100,000 photos to Facebook. Real-time methods are incapable to manage such amounts of data. Therefore, researchers are focusing in new representation and searching approaches in big datasets. In content-based information retrieval (CBIR), a key query is: given an object, to retrieve similar database objects. Seeking for similar images is often raised and, thus, nearest neighbor finding could be applied in a wide range of important applications [6].

An image database does not only store images at their source format, but also with appropriate transformations in the form of a multi-dimensional vector representation of the image characteristics, which are called *descriptor vectors* and have a dimensionality from a few tens up to a few thousands.

In this study, we seek for vectors not dominated by others, i.e. vectors that belong in the skyline set. An object dominates another one, when all its dimensions have better or equal value, whereas they have better values in at least one dimension. Important issues for such an operation in a large image database are: (i) the method efficiency, e.g. the required time, and (ii) the size of the skyline, which should be moderate even in high-dimensional spaces.

By finding the skyline of an image high-dimensional dataset, we can capture the intrinsic dissimilarities of images. In simple words, by using the image descriptor vectors, we can detect dissimilar images without defining any similarity or distance ranking function. Therefore, we can bypass any restrictions and distortions in the similarity notion that distance functions can embed, as well as to avoid complex indexing schemes and algorithms that are used on specific distance functions. Moreover, a concrete application of our approach is that the images found in the skyline can be used for grouping or clustering the whole image database, e.g. to initialize a clustering algorithm like k-means or as a cut-off parameter in a hierarchical clustering algorithm. More concrete applications are: clustering of visually similar images that can improve the efficiency of content-based image search engines and information systems, clustering of medical images for medical research, pattern recognition, face recognition and biometrics, automatic image tagging and categorization, and many other applications.

The sequel of this paper is organized as follows. In Section 2 we present the skyline ecosystem and how it is used to reach our aim: to extract a set of dissimilar images from a large image database. This ecosystem includes: the operation, the inherent dimensionality and the proposed algorithms to implement the operation. Section 3 elaborates further in the implemented algorithms; in particular, the hashing algorithm used for data partitioning in the branch-and-bound skyline algorithm is introduced. In Section 4 we present the experimental testbed (i.e. machine and datasets), as well as we present results of exhaustive experiments by examining wide ranges of all parameters. In addition, a visual evaluation of the result is presented and discussed. Finally, Section 5 summarizes and concludes this study.

2 SKYLINE ALGORITHMS

Assume a 2-dimensional dataset as depicted in Figure 1. Also assume that the objects with lower values in both dimensions are preferred. As mentioned before, the skyline set contains those objects that are not dominated by others. More formally, an object dominates another one if it has a better or equal value in all dimensions, whereas in at least one dimension its value is strictly better [16]. In our example, the skyline set comprises of the points a, b, e, h, o.

The skyline size mainly depends on the dataset nature. For a dataset of N d-dimensional objects, we can approximate the dimension d_0 from which on, all data items will be included in the

MEDES '17, November 7–10, 2017, Bangkok, Thailand

ACM ISBN 978-1-4503-4895-9/17/11…$15.00
https://doi.org/10.1145/3167020.3167050

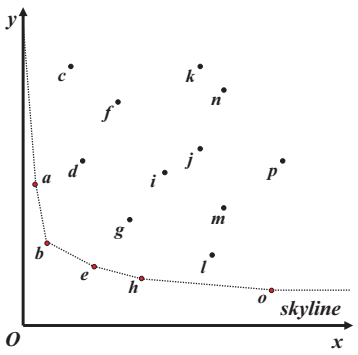

Figure 1: Skyline of a dataset.

skyline, because no object will dominate any other. This dimension is called *eliminating dimension* [15], and can be approximated by the formula: $d_0 \approx 2\lceil \ln N \rceil + 2$.

For large datasets, d_0 takes relatively small values. For example, for $N = 10^5, 10^6, 10^7, 10^8$ and 10^9, we derive 26, 30, 36, 40 and 44, respectively. Thus, a dataset with $N \leq 10^9$ objects and dimensionality greater than 44, almost all its objects are included in the skyline. However, vectors that describe images may reach a dimensionality of a few thousand. One idea to address this problem is to keep only K dimensions that contain the most information of the images.

Based on the above ideas, at a pre-processing step, we could analyze the dataset to select the K best dimensions having the highest distinct value cardinality, i.e. the dimensions with the most distinct values in the dataset. This way, the dimensions containing the most information are exploited in a more efficient manner. Reference [17] presents a content based image search method, which rearranges the dimensions of the dataset according to their distinct value cardinalities, using a hashing algorithm with multiple criteria to increase the probability that two similar images might be at a nearby storing space.

2.1 Skyline Calculation Algorithms

Until now many different methods have been suggested for skyline calculation. A comprehensive survey has been presented in [16]. The skyline algorithms fall into two main categories: (i) without indexing, and (ii) with indexing.

2.1.1 Without indexing. The first category includes the following skyline algorithms: Block Nested Loop, Divide-and-Conquer and Bitmap algorithms. It is noticed that these algorithms may generally perform well for medium size datasets and dimensionalities.

Block Nested Loop [1].
This method, which will be further elaborated below, is a variant of the naive approach where all objects are compared to each other. Each object is compared against all objects that are contained in a particular block and, depending on the existing dominance relation, these checked objects are added, removed or are subject to no change.

Divide and Conquer [1].
All variants of this family first calculate the median in any dimension to divide the space into two partitions P_1, P_2. Then, to calculate the skylines S_1, S_2 of P_1, P_2, recursively divide further the partitions P_1, P_2 until only one object is left in each partition. Finally, the global skyline is calculated by merging the partial skylines S_1, S_2 and removing the dominated objects.

Bitmap [14].
Each object is assigned to a binary vector of m bits, where m is the sum of the numbers of distinct values in all dimension. For example, if k_i is the number of distinct values in dimension i then: $m = \sum_{i=1}^{d} k_i$. Suppose that there are k_i distinct values in i-th dimension and that it is sorted in ascending order. Then, the j_i-th smaller value will be represented by k_i bits of which $k_i - j_i + 1$ bits from the left will be 1, while the rest will be 0s. Finally, the skyline is calculated by comparing the binary representation of the vectors.

2.1.2 With indexing. Skyline algorithms that use indices, even if they do not fit in main memory, speed up the process by reducing the number of domination comparisons and pruning dominated objects and areas.

In [1], a B-tree based algorithm is presented for 2-d data. In particular, each dimension is indexed by a B-tree. The algorithm finds a superset of the skyline. This is done by adjusting the first step A_0 of the algorithm of Fagin with data that are read simultaneously from the two B-trees until the first common object is found [4]. At this point we can claim that this object is certainly included into the skyline. The objects that have not appeared in all dimensions are located at further positions in the B-trees, and do not belong into the skyline. Finally, on the retrieved objects, an algorithm without indexing is applied to decide whether these objects belong or not into the skyline.

The first complete algorithm that uses spatial indices is the *Nearest Neighbor Skyline* (NN Skyline) [11]. It is based on the *Nearest Neighbor Search* (NN Search). Searching starts from the origin of the axes and - based on a distance function - it finds the nearest neighbor. The areas dominated by this object are rejected. The rest areas of the site are placed in a list (*to-do list*) and are treated recursively. This process ends when the list has no further areas to examine.

2.2 The Block Nested Loop Algorithm

The *Block Nested Loop* algorithm is one of the simplest forms of skyline calculation. This main memory algorithm uses only one structure to store the skyline objects. Each object is read from the file and it is compared with the objects already existing in the skyline set. The comparison result can be one of the following: (i) the new object is dominated by one or more of the current skyline objects, (ii) the new object dominates one or more objects of the current skyline, and (iii) none of the previous. In the first case, the process continues without making any changes in the current skyline. In the second case, the new object is added in the skyline set, whereas the objects of this set that are dominated by the new object are removed. Finally, in the third case, the new object is added into the skyline set without making any other changes.

2.3 Branch and Bound Skyline

The *Branch and Bound Skyline* algorithm is based on the nearest neighbors search by using an R-tree [13]. However, the same principles can be applied with other data partitioning and indexing

techniques. The intermediate nodes e^* of the R-tree are defined by the MBR (Minimum Bounding Rectangle) of spatial regions, whereas data at the lower level are organized into the tree leaves. L_1 is used as distance metric (i.e. the distance of a point from the origin equals the sum of the values of its coordinates, whereas the distance of a node equals the distance of the lower left point of its MBR). The process starts from the tree root and inserts the root information (*mindist*) into a min-heap. Then, the object with the smallest distance is expanded and, thus, new elements are properly entered into the min-heap.

The first object found in the min-heap (after expansion of the top entry) is the object located closer to the beginning of the axes. This object belongs into the skyline and is added into the list S that keeps the skyline objects. The algorithm proceeds and checks the subsequent objects and entries of the min-heap. It inserts into the list S all objects that are not dominated by objects that already exist in S. The algorithm visits the data entries in ascending order according to their distance from the beginning of the axes. Each object added to the set S during the execution of the algorithm is guaranteed that will be included in the final skyline.

When dimensionality grows, hierarchical indexing structures do not perform effectively due to the *Curse of Dimensionality* [2]. In the next section the implementation of Branch and Bound Skyline is described along with the use of hashing algorithms. Hash algorithms can replace R-tree, since R-trees and hierarchical indexing do not operate efficiently for high-dimensional data that are derived from image descriptors vectors.

3 SKYLINE WITH HASH INDEXING

There are many hashing algorithms that can be used as indexing methods. We selected four state-of-the-art hashing algorithms: *Locality Sensitive Hashing* (LSH) [8], *Iterative Quantization* (ITQ) [5], *Density Sensitive Hashing* (DSH) [10], and *Spherical Hashing* (SpH) [7]. Each one is tested and evaluated regarding its efficiency and data distribution through the skyline retrieval process.

The aim of our study is to create a unified evaluation that could find the skyline objects in a dataset of image high-dimensional descriptor vectors. The evaluation enables the user to select the proper method and find the skyline along with many other selective parameters.

Two fundamental skyline strategies were implemented: one without indexing (Block Nested Loop) and one with indexing (Branch and Bound Skyline). Branch and Bound Skyline was selected because it is one of the most efficient algorithms as it can be executed in secondary memory and is flexible regarding the data partitioning and indexing methods that can be used (i.e. R-tree, LSH, and so on). Block Nested Loop is a simple algorithm with performance depending mainly on the dataset size and in many cases is slow. However, it is easy to implement and we use it as reference.

In the course of this study, the libraries of *Armadillo*[1] and *Boost*[2] were helpful. Armadillo is a linear algebra library for C++, used to implemented all hashing algorithms examined in this paper. It has similar syntax with *Matlab*[3] and *Octave*[4]. Boost is a collection of general purpose libraries that extends C++ STL.

Algorithm 1: Hash Branch and Bound Skyline.

Input:
  i: input file, K: number of dimensions with highest distinct value cardinality, e: encoding length
Output:
  the skyline items
if $K > 0$ **then**
  value_cardinalities ← find the distinct value cardinality of each dimension
  SORT cardinalities in descending order
  dimension_mapping ← map the top K dimensions of **value_cardinalities** to the actual data
end
hash ← initialize selected hash algorithm with training set and encoding length e
while *not EOF input file* **do**
  if *option K is set* **then**
    v ← read vector based on **dimension_mapping**
  else
    v ← read vector form file
  end
  binary_code ← hash vector v and generate binary code
  data_point ← create object data_point with id, distance and binary_code
  buckets[binary_code] ← data_point
end
SORT each bucket IN buckets BY distance in asc. order
foreach *b IN buckets* **do**
  INSERT only b INTO the heap
end
while *heap size > 0* **do**
  c ← pop the top item of the heap
  if *c is type of bucket* **then**
    expand c and re-insert items to the heap
  else
    if *c is not dominated by any other item in the skyline* **then** INSERT c into skyline
  end
end

First, a pre-processing step (see Figure 2) is executed to derive the number of the distinct value cardinalities for each dimension. Then, it creates a mapping structure which selects only the K dimensions with the highest distinct value cardinalities and reads the input file by using this map. If K is not defined, then all available dimensions will be used in the skyline calculation without any pre-processing step.

[1] http://arma.sourceforge.net/
[2] http://www.boost.org/
[3] https://www.mathworks.com/products/matlab.html
[4] https://www.gnu.org/software/octave/

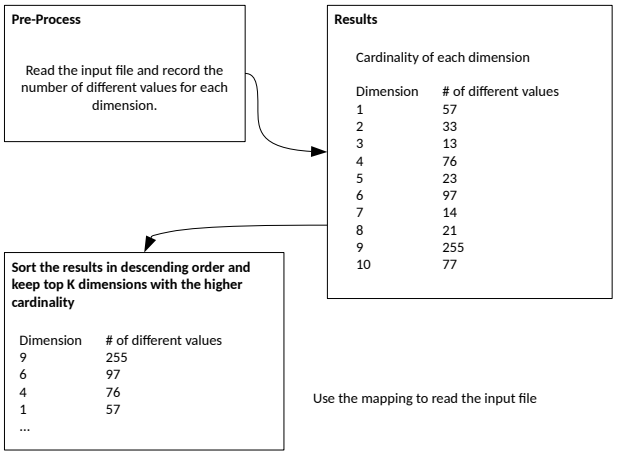

Figure 2: Preprocessing data.

The application starts reading the input file and passes each vector through the selected hash function, which results into a binary code used as a hash key to store objects with the same key in the same bucket. The buckets are sorted based on their distance from the axes origins in ascending order. The next step is to find the skyline. The process starts by inserting all bucket pointers with their corresponding distances into a min-heap. Then, if the top item of the min-heap is a bucket, it is expanded and all contained items are re-inserted into the min-heap. If the top item is a vector, then it is checked if it is dominated by a skyline item. Figure 3 shows the complete process.

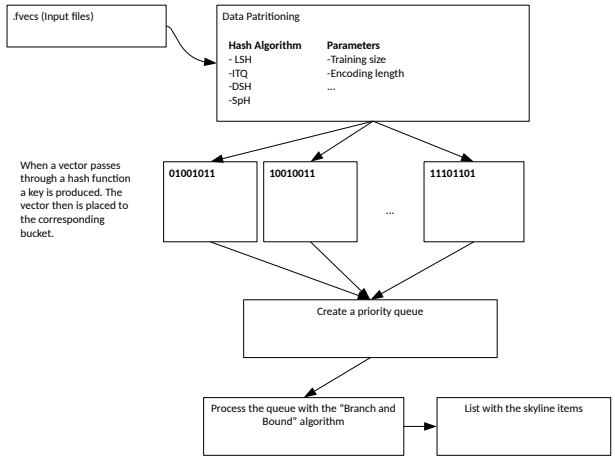

Figure 3: The complete process.

4 EXPERIMENTAL EVALUATION

Two real datasets were used for experiments and evaluation:

- The *ImageCLEF*[5] dataset which contains a collection of 237,424 images. Along with the actual image files, the corresponding descriptors of the images are included in the form of vectors in various formats. From these descriptors we selected CIME, CEDD, TLEP which have 64, 144, 576 dimensions, respectively.
- The ANN_SIFT1M[6] dataset [9] which is a large dataset with 1,000,000 SIFT image descriptor vectors of 128 dimensions without the corresponding actual images.

All experiments were conducted on a desktop PC with the following configuration: CPU: Intel Core i7 - 860 @ 2.8GHz, RAM: 12GByte, Disk: 256GB SSD.

4.1 Time Performance

In the first group of the experiments we test the performance of hashing algorithms. Experiments were conducted for different coding rates (*-e* parameter). In each run the following times were recorded: training time, hashing time and total time. All calculated results are averages from ten runs. In particular, the ANN_SIFT1M dataset was used in full dimensionality (without setting K). Comparing the performance of the hashing algorithms used in Branch and Bound Skyline, in most cases LSH was the faster algorithm. Figure 4 illustrates the hashing time for various encoding lengths.

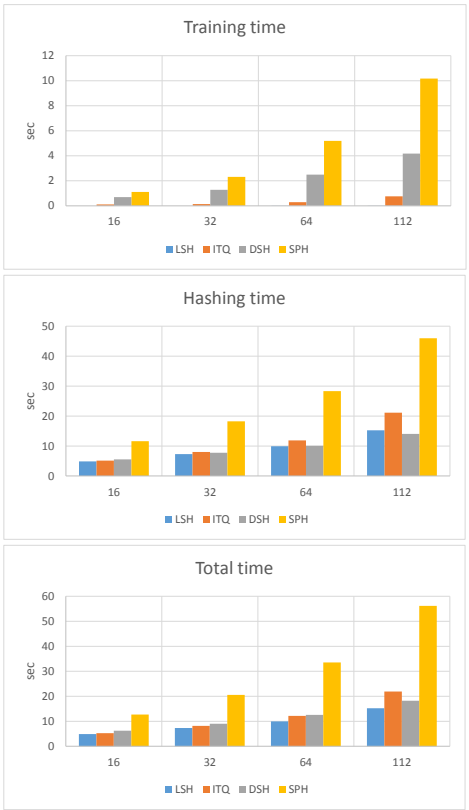

Figure 4: Training, Hashing and Total Time.

[5] http://imageclef.org/wikidata

[6] http://corpus-texmex.irisa.fr

Figure 5: Data distribution for 4,8,12 bit encoding length.

4.2 Data Distribution

In the second group of experiments we recorded the data distribution through the hashing algorithms for various encoding lengths (more specifically, for $e = 4, 8, 12, 16$). In particular, we can monitor how the data objects are hashed and distributed in the 2^e hash-buckets. The results show that SpH spreads the data more evenly.

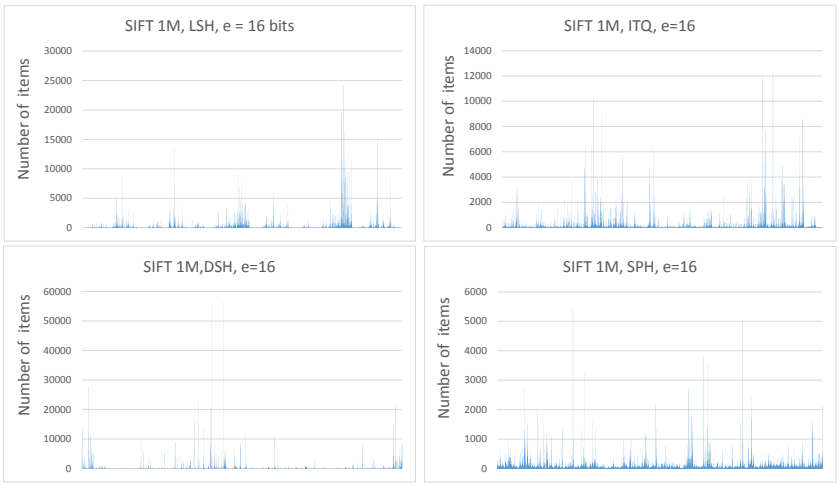

Figure 6: Data distribution for 16 bit encoding length.

Figures 5 and 6 depict the number of items in each bucket for encoding length 4, 8, 12 and 16 bits. All four hashing algorithms were evaluated for the ANN_SIFT1M dataset.

4.3 Evaluation of the Skyline Calculation Performance

In the third group of experiments we measure the execution times of the whole process for different values of K (mapping to the top-K dimensions with the highest distinct value cardinality). Both datasets were used for this evaluation of Block Nested Loop and Branch and Bound Skyline. Specifically, we used the following datasets: SIFT-1M (128 dimensions), ImageCLEF-237K with CEDD descriptor (144 dimensions), ImageCLEF-237K with CIME descriptor (64 dimensions), ImageCLEF-237K with TLEP descriptor (576 dimensions).

SIFT-1M	K=15	K=16	K=17	K=18	K=19
LSH	6.04	8.37	16.48	34.24	45.90
ITQ	6.05	8.36	17.03	32.43	47.79
DSH	6.03	8.33	15.81	32.39	47.48
SPH	7.49	9.74	17.11	34.10	49.55
BNL	33.04	79.33	337.14	633.65	947.82

Table 1: Calculation time (sec) for the SIFT descriptor.

CEDD	K=15	K=16	K=17	K=18	K=55
LSH	1.35	1.26	1.34	1.35	12.26
ITQ	1.31	1.32	1.68	1.61	12.02
DSH	1.24	1.50	1.76	1.51	13.48
SPH	2.21	2.34	2.52	2.39	13.67
BNL	0.54	0.53	0.61	0.66	247.53

Table 2: Calculation time (sec) for the CEDD descriptor.

Tables 1-4 present the calculation time recorded for each image descriptor. Branch and Bound Skyline was faster than Block Nested Loop in general, except for CIME and CEDD image descriptors of the ImageCLEF dataset in which Block Nested Loop was faster for some encoding lengths. This happens because objects with small distance from the axes origin are very early inserted into the skyline. Therefore, almost all subsequent inserted objects are dominated from the existing ones. The main difference between the two skyline algorithms is the training/initialization times that the hashing process requires. Regarding the hashing methods, all four methods have about the same performance. SpH is the slowest of the four, whereas most of the times LSH is the fastest. Except for the execution time, the number of the objects in the skyline was recorded for every dataset and for every K (see Table 5).

CIME	K=15	K=16	K=17	K=18	K=64
LSH	1.24	1.27	1.39	1.32	2.17
ITQ	1.25	1.39	1.40	1.37	2.23
DSH	1.32	1.33	1.45	1.48	2.33
SPH	1.81	1.86	1.99	1.94	3.10
BNL	0.43	0.41	0.52	0.51	1.08

Table 3: Calculation time (sec) for the CIME descriptor.

TLEP	K=15	K=16	K=17	K=18
LSH	2.79	2.99	3.54	3.54
ITQ	2.93	3.04	3.68	3.62
DSH	2.83	3.12	3.66	3.70
SPH	3.65	3.75	4.48	4.46
BNL	19.30	19.52	61.44	62.19

Table 4: Calculation time (sec) for the TLEP descriptor.

	SIFT-1M	CEDD	CIME	TLEP
K=15	556	15	12	2646
K=16	3024	16	12	2658
K=17	5577	17	12	2776
K=18	10255	18	13	2778
K=19	13731	-	-	-
K=22	474555	-	-	-
K=55	-	7458	-	-
K=64	-	-	38	-

Table 5: Number of skyline items.

#	Image ID	Distance																																			
1	4401	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	31063	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
3	49869	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	53442	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
5	229255	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	201996	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	201998	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	195376	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
9	38909	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
10	42426	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
11	7773	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
12	28901	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
13	23420	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	114651	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	20650	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	58914	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	66501	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	104742	2	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	101102	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	72751	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	33696	2	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	19648	3	0	0	0	0	0	0	0	0	0	0	0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	63189	4	0	0	0	3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	241433	5	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	1	0	0	0	0
25	112374	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	211443	7	0	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	214726	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0
28	207058	7	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
29	56111	7	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
30	205105	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	136104	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0
32	67766	10	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
33	77616	11	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	3	0	0	0	0
34	11401	11	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	3	0	0	0	0
35	209798	16	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	7	4	2	0	0	0
36	140748	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	5	3	6	0	0	0
37	144565	17	0	0	4	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	5	3	0	0	0	0
38	80472	18	0	0	4	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	6	3	2	0	0	0

Figure 7: Skyline items from the visual evaluation experiment.

4.4 Visual Evaluation of the Skyline Images

In this last group of experiments we visually tested the differences in the actual retrieved images in the skyline and the degree of the intrinsic differences of their descriptor vectors. For this evaluation we used the ImageCLEF dataset and CEDD descriptors, which combine color and texture information of the images. We used this dataset since, along with the descriptor vectors of the images, the actual corresponding images are also available. Due to the lack of space, we present in this paper one of the most representative experiments for this evaluation, which was carried out with the following parameters: "`skyline.exe -i cedd.bin -k 36 -e 4 -h sph -mem 1`". The outcome was a skyline with 38 images, each of which had a dominant color. Some of the retrieved skyline images are displayed in Figure 8, whereas a list with all resulted images with their corresponding CEDD descriptor vectors is presented in Figure 7.

Fundamental differences were detected among the retrieved images through a visual inspection/checking. These images are inevitably very different, due to the fact that they belong to the skyline. This result of existence of very different actual images in the skyline of the high-dimensional space that their descriptor vectors define is significant. It is significant, as it shows that without any similarity/distance ranking function, any distance-based indexing, or any complex indexing algorithm, we can select different images very fast even in large image databases, by taking advantage of the intrinsic differences that are included into the descriptor vector representations of the images.

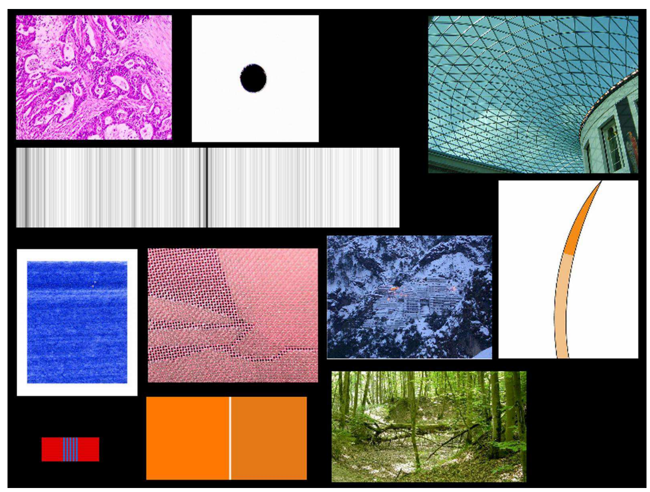

Figure 8: Images from the visual evaluation.

5 CONCLUSIONS

This paper presents a first ever study, to the best of our knowledge, on how the intrinsic dissimilarities of images included in their descriptor vectors affect the actual image differences, through skylines in the high-dimensional spaces defined from their descriptor vectors. The reason of using skylines was that fast retrieval of different images is facilitated, even in large databases, without defining any similarity/distance ranking function, any distance-based

indexing, or any complex indexing algorithm, taking advantage of the intrinsic differences that are included into the source descriptor vectors of the images. We assume that the used descriptor vectors are representative for describing the images (like: CIME, CEDD, TLEP, SIFT, SURF, etc., or combinations of them).

To retrieve efficiently the skyline in such high-dimensional spaces, a key point of this study was the combination of basic skyline methods with four hashing state-of-the-art algorithms to create efficient indexing and scalability support in large image databases. Finding the skyline in high dimensionality data results that a large part of the dataset, or even the whole dataset, is included in the skyline, which is an undesirable fact. Therefore, to reduce the dimensionality we selected the dimensions with the higher distinct value cardinality as they hold the most included information in the descriptor vectors, a method that was suggested as a prior step to processing a dataset for content based image retrieval [17].

We compared and evaluated the results of different hashing algorithms and skylines by using two real image datasets, measuring the performance and the effectiveness of the skyline retrieval process. We concluded that calculating methods of skyline through indexing structures combined with the use of hash algorithms are much faster than methods of skyline without index or methods with hierarchical indexing structures for high dimensional datasets and large groups of objects.

Moreover, we performed a visual evaluation of the images found in the skyline to observe the actual differences that the images have, and their relationship to the type of descriptors. We concluded that the images found in the skyline have actual different characteristics, which depend on the type of the descriptor. This is a strong indication that the skyline objects can be used as base items for future grouping or clustering of the whole image database, which is a concrete byproduct application of this study, e.g. to initialize a clustering algorithm like k-means or as a cut-off parameter in a hierarchical clustering algorithm.

Finally, an extension of this work could be the evaluation of other segmentation methods in conjunction with the Branch and Bound Skyline algorithm, the implementation of additional methods that return only a part of the skyline or search for the skyline in a part only of the whole dataset, and further exploration of the relationship of the skyline objects with clustering methods.

Another future research topic would be to compare the resulting skyline set with results of algorithms for furthest neighbor searching [12], the opposite problem of nearest neighbor searching. Although from the efficiency point of view, skyline processing is prevailing, it would be interesting to have a deeper knowledge about the effectiveness of the proposed method in selecting dissimilar images from a large image database. In addition, another problem relevant to ours is that of diversification [3]; comparison of the two approaches could provide interesting results.

ACKNOWLEDGEMENT

All hashing algorithms implemented in this study were based in the following Github repository: `https://github.com/willard-yuan/hashing-baseline-for-image-retrieval`,
which is a collection of hashing algorithms in Matlab for use in content based image retrieval applications.

REFERENCES

[1] S. Borzsony, D. Kossmann, and K. Stocker. 2001. The Skyline Operator. In *Proceedings 17th International Conference on Data Engineering (ICDE)*. 421–430.
[2] Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luis Marroquín. 2001. Searching in Metric Spaces. *Comput. Surveys* 33, 3 (2001), 273–321.
[3] Marina Drosou and Evaggelia Pitoura. 2014. Diverse Set Selection Over Dynamic Data. *IEEE Transactions on Knowledge and Data Engineering* 26, 5 (2014), 1102–1116.
[4] Ronald Fagin. 1999. Combining Fuzzy Information from Multiple Systems. *Journal of Computer & System Sciences* 58, 1 (1999), 83–99.
[5] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin. 2013. Iterative Quantization: A Procrustean Approach to Learning Binary Codes for Large-Scale Image Retrieval. *IEEE Transactions on Pattern Analysis & Machine Intelligence* 35, 12 (2013), 2916–2929.
[6] Kristen Grauman and Rob Fergus. 2013. *Learning Binary Hash Codes for Large-Scale Image Search.* Springer, Berlin, Heidelberg, 49–87.
[7] Jae Pil Heo, Youngwoon Lee, Junfeng He, Shih Fu Chang, and Sung Eui Yoon. 2015. Spherical Hashing: Binary Code Embedding with Hyperspheres. *IEEE Transactions on Pattern Analysis & Machine Intelligence* 37, 11 (2015), 2304–2316.
[8] Piotr Indyk and Rajeev Motwani. 1998. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings 30th Annual ACM Symposium on Theory of Computing (STOC)*. 604–613.
[9] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis & Machine Intelligence* 33, 1 (2011), 117–128.
[10] Zhongming Jin, Cheng Li, Yue Lin, and Deng Cai. 2014. Density Sensitive Hashing. *IEEE Transactions on Cybernetics* 44, 8 (2014), 1362–1371.
[11] Donald Kossmann, Frank Ramsak, and Steffen Rost. 2002. Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. In *Proceedings 28th International Conference on Very Large Data Bases (VLDB)*. 275–286.
[12] R. T. Liu, C. Chang, Z. Q. Man, and Z. Wang. 2015. The Farthest Neighbor Queries Based on R-trees. In *Proceedings International Conference on Machine Learning & Cybernetics (ICMLC)*. 235–238.
[13] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. 2003. An Optimal and Progressive Algorithm for Skyline Queries. In *Proceedings ACM International Conference on Management of Data (SIGMOD)*. 467–478.
[14] Kian-Lee Tan, Pin-Kwang Eng, and Beng Chin Ooi. 2001. Efficient Progressive Skyline Computation. In *Proceedings 27th International Conference on Very Large Data Bases (VLDB)*. 301–310.
[15] Eleftherios Tiakas, Apostolos N. Papadopoulos, and Yannis Manolopoulos. 2013. On Estimating the Maximum Domination Value and the Skyline Cardinality of Multidimensional Data Sets. *International Journal of Knowledge-Based Organizations* 3, 4 (2013), 61–83.
[16] Eleftherios Tiakas, Apostolos N. Papadopoulos, and Yannis Manolopoulos. 2016. Skyline Queries: An Introduction. In *Proceedings 6th International Conference on Information, Intelligence, Systems & Applications (IISA)*. 1–6.
[17] Eleftherios Tiakas, Dimitrios Rafailidis, Anastasios Dimou, and Petros Daras. 2013. MSIDX: Multi-Sort Indexing for Efficient Content-Based Image Search and Retrieval. *IEEE Transactions on Multimedia* 15, 6 (2013), 1415–1430.