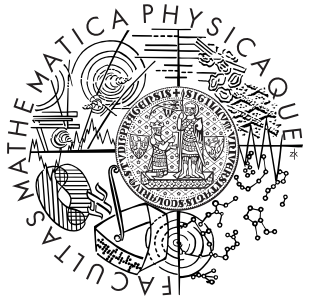
Charles University, Prague, Czech Republic
Faculty of Mathematics and Physics

MASTER THESIS



Bc. Tomáš Grošup

Multi-model Approach For Effective Multimedia Exploration

Department of Software Engineering, MFF UK

Supervisor of the master thesis: RNDr. Jakub Lokoč, Ph.D.

Study programme: Informatics

Specialization: Software Systems

Prague 2014

I would like to thank my supervisor, Jakub Lokoč, for the numerous advices he gave me. I would also like to thank Tomáš Skopal for a lot of feature ideas.

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In Prague date Tomáš Grošup

Název práce: Vícemodelový přístup pro efektivní multimediální exploraci
Autor: Tomáš Grošup
Katedra (ústav): Katedra softwarového inženýrství
Vedoucí diplomové práce: RNDr. Jakub Lokoč, Ph.D.
E-mail vedoucího: lokoc@ksi.mff.cuni.cz

Abstrakt: Tato práce se zabývá explorací multimediálních kolekcí. Detailně rozebírá problematiku explorace a navrhuje nové přístupy, dva založené na datové struktuře M-Index a dva využívající více podobnostních modelů naráz. Tyto přístupy jsou srovnány pomocí pokročilé uživatelské studie. Součástí práce je také analýza nového exploračního systému, návrh jeho architektury, implementace systému i jeho nasazení. Explorační systém byl použit v různých aplikacích, které jsou také popsány a ukázány v této práci.

Klíčová slova: Podobnostní modely, multimediální explorace, dobývání multimedií

Title: Multi-model Approach For Effective Multimedia Exploration
Author: Tomáš Grošup
Department: Department of Software Engineering
Supervisor: RNDr. Jakub Lokoč, Ph.D.
Supervisor's e-mail address: lokoc@ksi.mff.cuni.cz

Abstract: This work is focusing on exploration of multimedia collections. It describes the problematic of exploration and proposes new approaches to it, two based on the data structure M-Index and two utilizing multiple similarity models at once. Those approaches were compared using an extensive user study. Part of this work is also devoted to analysis of a new exploration system, design of its architecture, system implementation and its deployment. This exploration system was used in several applications, which are also shown and described in this thesis.

Keywords: Similarity models, multimedia exploration, multimedia retrieval

Contents

Chapter 1

Multimedia exploration problematic

1.1 Multimedia

The last decade witnessed a massive growth of multimedia contents. Gantz [1] shows that the amount of data stored got bigger ten times between years 2006 and 2011 and this exponential growth continues, mostly driven by the amount of videos, still images and sounds [1]. Jonathan Good [2] estimated 3.5 trillion photos taken till the year 2011 and Yahoo [3] estimates 880 billion photos taken just this year, 2014. This explosion of data was possible mainly because of the availability of capturing devices such as smart phones and compact cameras. They are getting cheaper every year and more than a fifth of people in the world owns one [4].

A lot of this data is stored not only locally on authors' computers or capturing devices, but also uploaded online to web portals for sharing photos or videos, often directly from the capturing device. The most popular galleries are Youtube, Flickr, Instragram and most importantly, the social network Facebook [2]. Facebook employees reported [5] 350 million photos uploaded every day. This makes Facebook the largest image gallery on the Internet.

1.2 Text-based search

All this data and its online accessibility led to a significant research activity in multimedia retrieval. The problem is to satisfy user's search intention with relevant objects from a huge database, preferably without any delay. Traditional web search engines like Google, Yahoo and Bing utilize text-based approaches, where a multimedia object is described/annotated by a set of keywords. Such retrieval approach is very efficient and allows the reuse of well established search architectures utilized for textual data. The keywords are usually extracted from the surrounding HTML page or from a human-entered description. For example, human annotation is often used in photo galleries, where the concept of tags is used to provide search functionality. However, for web-scale collections the automatic annotation (e.g., based on surrounding texts on the web page) is necessary, because the approach with human-entered annotation has several drawbacks. First, it is possible to

annotate only a small fraction of the available data due to limited number of human annotators (or experts in case of domain specific data). Also, the annotation can be subjective and thus the results of the search may not satisfy all users. There are also works focusing on automatic annotation, but they often yield only a limited precision.

To search multimedia data without any textual information, their content has to be analyzed and used. One way to utilize the content is to filter the objects based on some characteristics. Google Images allow this in their advanced search with the option to narrow the results by size, color and other properties [6].

1.3 Similarity Search

In content-based retrieval [7, 8, 9], the content of the data is analyzed and used for searching. To query a content-based retrieval system, user has to provide an object as the query. Both Google Images and Bing Images use the query-by-example paradigm in their content-based retrieval functions, in which a user selected image serves as the query. Since the query object does not have to be present in the searched database, the systems often implement a similarity model to return at least the most similar images.

1.3.1 Similarity model

The similarity model is usually defined as tuple $(\mathbb{F}, \delta)$, where $\mathbb{F}$ represents a feature space in which multimedia objects are mapped using a feature extraction technique, and δ represents a distance function $\mathbb{F} \times \mathbb{F} \to \mathbb{R}$, where the lower the distance, the more similar two objects are. The multimedia collection is then represented as a subset $\mathbb{S}$ of the feature space $\mathbb{F}$, and distance function $\delta(o, q)$ defined for two image descriptors $o, q \in \mathbb{F}$ is used to rank the similarity between two corresponding multimedia objects. The similarity query then usually consists of ranking of all database objects according to a query object and selecting the most relevant (top ranked) database objects. For specific collections like EEG curves, x-ray images or military satellite images, the similarity model has to be designed with the help of domain experts. The experts help with the selection of representative descriptors and suitable similarity measures in order to obtain the most relevant rankings.

Metric space approach

For huge multimedia collections, ranking of all database objects using a similarity model to answer a query is not feasible, and thus an index for efficient similarity searching is necessary. A popular technique for efficient similarity searching is the metric space approach where the distance function satisfies metric axioms, especially the triangle inequality. The triangle inequality is crucial for metric indexing. Following definitions in this chapter are taken from the work of Zezula et al. and from the thesis of Lokoč[10, 11], where you can also find more details and examples.

Definition 1 *Let $\mathbb{F}$ be a feature space, δ a distance function measured on $\mathbb{F}$. $\mathcal{M} = (\mathbb{F}, \delta)$ is called metric space, if distance function δ: $\mathbb{F} \times \mathbb{F} \mapsto \mathbb{R}$ fulfills following postulates:*

(p1) $\forall x, y \in \mathbb{F}$, $\delta(x, y) \geq 0$ — *non-negativity*

(p2) $\forall x, y \in \mathbb{F}$, $\delta(x, y) = \delta(y, x)$ — *symmetry*

(p3) $\forall x \in \mathbb{F}$, $\delta(x, x) = 0$ — *reflexivity*

(p4) $\forall x, y \in \mathbb{F}$, $x \neq y \Rightarrow \delta(x, y) > 0$ — *positiveness*

(p5) $\forall x, y, z \in \mathbb{F}$, $\delta(x, z) \leq \delta(x, y) + \delta(y, z)$ — *triangle inequality*

Although the metric axioms represent restrictions for the similarity model, we can still utilize many general metric indexing techniques that will work in a large number of applications across several domains.

Feature extraction

Feature extraction is a transformation of the raw input data to a point o in the feature space, $o \in \mathbb{F}$. This transformation reduces description complexity of modeled objects (e.g., from matrix of pixels to simple histogram) while it should still preserve relevant features and possibly also reduce noise information in the data. In the domain of image and video retrieval, feature extraction techniques often represent objects using color, shape, texture features or using a combination of these features.

The resulting set of features can take a variety of forms. The following section shows some examples of feature extractions used in image processing. Other examples with their implementation can be found for example in the Lucene Image Retrieval library [12].

MPEG-7

MPEG-7 [13] is a standard for description of multimedia content. There are many different parts describing various aspects of multimedia data. The resulting descriptors are usually fixed-sized vectors describing the content of the whole image. The vectors can be combined together to form a better performing descriptor. Mufin [14] is an example of content-based retrieval system that uses linear combination of several MPEG-7 descriptors.

Scale invariant feature transform

Scale invariant feature transform [15] (SIFT) is an example of descriptor focusing on local features. Instead of a global description of an image, the algorithm detects interesting keypoints in the image and describes them separately resulting in many-vectors-per-image representation. To store the image in a more compact way, bag-of-visual-words (BoVW)

approach [16] is used to represent the image as a frequency histogram of so called codewords. This model can be further extended with compaction techniques such as vector of locally aggregated descriptors (VLAD) [17, 18]. A lot of research was spent on improving the original SIFT algorithm, Wu et al focused on their comparison [19].

Feature Signatures

The feature signatures are popular tool for flexible image description, where the main advantage of using feature signatures is their variable size, which makes them suitable for both simple and complex images [20, 21] .

Definition 2 *Given a feature space $\mathbb{F}$, the feature signature S^o of a multimedia object o is defined as a set of tuples from $\mathbb{F} \times \mathbb{R}^+$ consisting of representatives $r^o \in \mathbb{F}$ and weights $w^o \in \mathbb{R}^+$*

Each representative r^o from a feature signature represents an area with a central point. The feature space we used in this work for feature signatures is a 7-dimensional vector of real numbers comprising the following components:

X. The x-coordinate

Y. The y-coordinate

L. Lightness, L* in *CIE LAB*[1] color space

A. Color position between magenta and green, a* in *CIE LAB* color space

B. Color position between yellow and blue, b* in *CIE LAB* color space

C. Contrast

E. Entropy

Unlike SIFT, feature signatures do not use a frequency histogram based on a global vocabulary (BoVW model), but each feature signature uses its own vocabulary of visual words. Therefore, the feature signatures have to be compared by more expensive adaptive distance functions that compare these local vocabularies [23]. Christian Beecks [24] provides an extensive work on the topic of feature signatures.

[1] *LAB* color space, which was designed to approximate human vision [22]. CIE states for *International Commission on Illumination (Commission internationale de l'éclairage in French).*

Distance functions

Distance functions are used to quantify the closeness of objects from the feature space $\mathbb{F}$ and thus measure their dissimilarity. It is up to the domain experts to choose and eventually parameterize a suitable distance function that should mostly correspond to user expectations. The same descriptors can be compared by many different distances, which can result in different effectiveness and performance.

In this section we will present the distances used in this work.

Minkowsi metrics

The Minkowski metrics (or L_p metrics) are the most popular dissimilarity measures used in various applications. However, the metrics are restricted just to vector spaces, where a distance between two vectors (points) is computed.

Definition 3 *Let $\mathbb{V}$ be an n-dimensional vector space and $x, y \in \mathbb{V}$, then an L_p metric is defined as:*

$$\mathrm{L}_p(x, y) = (\sum_{i=1}^{n} |x_i y_i|^p)^{\frac{1}{p}}$$

Only for $p \geq 1$ holds that L_p is a metric, for $p < 1$ it does not satisfy the triangle inequality. L_1 distance is known as the Manhattan distance, the L_2 metric is the Euclidean distance and the L_∞ is called the Chessboard distance. The time complexity of the distance evaluation is $O(n)$, hence L_p metrics are considered as cheap dissimilarity measures. The L_p metrics are suitable to model a dissimilarity in vector spaces with independent dimensions, such as the MPEG-7 descriptors.

There also exist cases, where it is profitable to prefer more significant coordinates and to suppress the less significant ones. For such cases, the weighted L_p metric can be used as a generalized variant of the original L_p metric [11].

Signature Quadratic Form Distance

The Signature Quadratic Form Distance is a generalization of the Quadratic Form Distance [11] for feature signatures. It is defined as follows:

Definition 4 *Given two feature signatures $S^q = \{\langle r_i^q, w_i^q \rangle\}_{i=1}^n$ and $S^o = \{\langle r_i^o, w_i^o \rangle\}_{i=1}^m$ and a similarity function $f_s : \mathbb{F} \times \mathbb{F} \to \mathbb{R}$ over a feature space $\mathbb{F}$, the signature quadratic form distance SQFD_{f_s} between S^q and S^o is defined as:*

$$\mathrm{SQFD}_{f_s}(S^q, S^o) = \sqrt{(w_q \mid -w_o) \cdot A_{f_s} \cdot (w_q \mid -w_o)^T},$$

where $A_{f_s} \in \mathbb{R}^{(n+m)\times(n+m)}$ is the similarity matrix arising from applying the similarity function f_s to the corresponding feature representatives, i.e., $a_{ij} = f_s(r_i, r_j)$. Furthermore, $w_q = (w_1^q, \ldots, w_n^q)$ and $w_o = (w_1^o, \ldots, w_m^o)$ form weight vectors, and $(w_q \mid -w_o) = (w_1^q, \ldots, w_n^q, -w_1^o, \ldots, -w_m^o)$ denotes the concatenation of weights w_q and $-w_o$.

The similarity function f_s is used to determine similarity values between all pairs of representatives from the feature signatures. In our implementation we use the similarity function $f_s(r_i, r_j) = e^{-\alpha L_2(r_i, r_j)^2}$, where α is a constant for controlling the precision-indexability tradeoff, and L_2 denotes the Euclidean distance.

The time complexity of SQFD is $O(n^2 * T_{f_s})$, where T_{f_s} is the time complexity of the f_s similarity function. The time complexity of the SQFD can be a performance bottleneck of the retrieval, especially when using larger feature signatures. Therefore the number of SQFD computations becomes an important performance measure in indexing techniques [25, 26].

Cosine similarity

Cosine similarity measures the cosine of the angle between two vectors.

Definition 5 *Let $\mathbb{V}$ be an n-dimensional vector space and $A, B \in \mathbb{V}$, then the cosine similarity is defined as:*

$$\sigma_{cos}(A, B) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

The return values range from -1 meaning the exact opposite to +1 meaning an exact match.

To turn this similarity function into the cosine distance, we can subtract the value from 1:

$$D_{cos}(A, B) = 1 - \sigma_{cos}(A, B)$$

Such distance however violates the triangle inequality metric postulate. As an proper metric, the *deviation metric* can be used [27]:

$$D_{dev}(A, B) = \arccos(\sigma_{cos}(A, B))$$

The cosine similarity is especially useful for bag-of-words approaches and is often used in information retrieval [28]. It is also the recommended distance for the bag-of-visual-words used for local features like SIFT, because it effectively normalizes the number of extracted keypoints. In the case of information retrieval, the vectors are never negative. This means that the return value of cosine similarity can never be negative as well and the values range from 0 to 1. When cosine distance is used, the values also range from 0 to 1.

1.3.2 Queries

Besides many others, there are two most popular similarity queries used in similarity search tasks: the range query and the nearest neighbor query ($k-$NN).

Definition 6 *Let q be a query object in domain $\mathbb{F}$ and r be a distance. Range query is defined as $R(\mathrm{q}, \mathrm{r}) = \{o \in X, \delta(o, q) \leq \mathrm{r}\}$, where δ is a metric function on domain $\mathbb{F}$ and $X \subseteq \mathbb{F}$.*

Definition 7 *Let q be a query object in domain $\mathbb{F}$ and k be a number of wanted results. The k nearest neighbor query is defined as $\mathrm{k}NN(\mathrm{q}) = \{R \subseteq X, |R| = \mathrm{k} \wedge \forall x \in R, y \in X - R : \delta(q, x) \leq \delta(q, y)\}$, where δ is a metric function on domain $\mathbb{F}$ and $X \subseteq \mathbb{F}$.*

As we can see in Figure 1.3.2 , both similarity queries are represented by a ball region, where the radius of the ball is dynamically adjusted for the $k-$NN query.

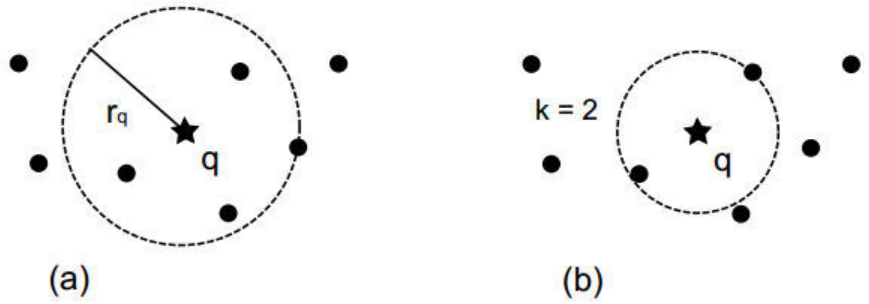


Figure 1.1: (a) Range query ball with radius r_q (b) 2NN query

1.3.3 Metric indexes

When executing a query in large-scale multimedia databases, it is unacceptable to compute the distance function between the query object and every other object in the database. To reduce the number of distance computations, some form of indexing has to be applied. The metric space model can use no other information than the distances between objects. Thanks to the triangle inequality postulate, a previously computed distance can be used to estimate a lower bound and an upper bound of the real distance to the query object. Having the database objects $p, q, x \in S$ and previously computed distances $\mathrm{d}_1 = \delta(p, x)$ and $\mathrm{d}_2 = \delta(p, q)$, we can calculate from the triangle inequality the lower and upper bound of distance $\delta(x, q) \in \langle |d_1 - d_2|, d_1 + d_2 \rangle$. If the lower bound is greater than the radius of current query, we can exclude x from the result set for query q without calculating the exact distance.

A wide family of metric indexes, so called Metric access methods (MAMs) [10, 11], utilizes this approach. The majority of indexes maintains precomputed distances from every database object to a fixed set of pivots, $P \subset S$. Various techniques for selecting pivots and their comparison can be found in the work of Bustos and Amato[29, 30, 31].

LAESA

In the case of LAESA [32], the index consists just of distances from all database objects to pivots. When performing a search, distances from query to pivots are used to eliminate objects using the lower-bound estimate. All remaining objects are directly compared to the query object.

M-Tree

The M-Tree [33] is a hierarchical index structure with good performance in secondary memory. The index is build in a bottom-up way resulting in a balanced hierarchy of nested metric ball regions. During query processing, each ball region is checked for the overlap with the query ball, and if the balls "intersect", the query has to recursively follow this part of the M-Tree potentially ending in leaf nodes, where data objects are stored.

Modifications to this index exist, such as Pm-Tree [34] which enhances the original structure with a global set of pivots. The pivots cut ball-shaped regions by a set of rings to form more compact regions [35, 36].

M-Index

M-Index is a state-of-the-art structure combining practically all existing techniques for metric space pruning and filtering [37]. M-Index creates again a hierarchy of metric clusters, but unlike M-Tree employing ball-regions, the M-Index uses a recursive Voronoi-based partitioning of the metric space using a preselected set of global pivots resulting in the cluster tree structure. The cluster tree and stored distances from objects to pivots are then used for efficient metric space search employing object-pivot distance constraint, double-pivot distance constraint and range-pivot distance constraint [37].

M-Index was also proven to be very efficient in approximate search. In approximate search, there is a tradeoff between accuracy of the search and its performance. A comparison of approximate search techniques was shown by Novák, Batko and Lokoč et al. [37, 38] and will be further described in section 2.2.

1.4 Multimedia Exploration

Both text-based search and query-by-example systems require a query formulation to satisfy user's need. This can only be successful when the user's information needs can be (clearly) specified [39]. Sometimes, the user does not have an example object or does not know how to formulate a query correctly. Even if he does, a search engine based on subjective annotations may return completely irrelevant results from user's point of view. Often the only thing the user has is an idea, a picture of the result in his head. Only once he sees some result he can decide whether it is relevant or not ("I don't know what I'm looking for, but I'll know when I find it" [40]). Multimedia retrieval systems also do not allow the

user to gain insight of a collection, possibly very large one, without starting with a query first.

Multimedia exploration tries to solve these problems by making the user a significant part of the search process. The exploration is intended to maximize the use of the discriminative power of the user and allow him to examine the contents of the collection in an effective way.

1.4.1 Initial view

At the beginning of an exploration process, user is presented with an initial view (e.g., see [41, 42]). The initial view should give a basic overview of the collection, enabling users to start browsing to images of his/her interest. Composition of the initial view is a challenging problem especially in large collections and is commonly referred as the Page-Zero problem [43]. From the initial view, users should be able to reach different parts of the collection by exploration operations like zooming-in, zooming-out, panning [44] or just by selecting objects of his/her interest.

1.4.2 Exploration/Exploitation

Suditu et al. [45] divide exploration queries into two regimes : Exploration and Exploitation.

Exploration is supposed to give an insight at higher level and present the categories to the user. Authors state: "The exploration step aims at driving the search towards different areas of the feature space in order to discover not only relevant images but also informative images"[45]. Exploitation phase should dive more into the collection, converge to specific images and return more objects relevant to the user.

Their proposed system tries to balance the tradeoff between the two regimes adaptively to user's actions and transition between them seamlessly.

1.4.3 Relevance feedback

Some systems try to deduce the ideal similarity model adaptively from the exploration history, utilizing relevance feedback techniques in the process. This process can change the similarity model underneath, for example by changing the weights used in weighted L_p metric and preferring some dimensions over the others [46] or by changing the weights for two different similarity models combined together [43]. The work of Urban et al. [47] describes different ways of collecting relevance feedback in content-based image retrieval and provides basic terminology for systems with build-in relevance feedback techniques. The authors have also utilized the Dempster-Shafer mechanism to combine multiple number of sources to a single ranked list based on the trust in each source calculated from previous steps.

1.4.4 Graphical user interface

Conventional search engines show the search results in form of a ranked list. Such presentation loses information about similarity between retrieved results. Several exploration systems provide the search results in more attractive and interactive graphical user interfaces (GUI), often utilizing some form of similarity based layout [48]. According to Nguyen and Worring [48], such layout should reflect the similarity between multimedia objects and should also offer an overview of the multimedia collection.

Different approaches to similarity based GUI exist, most of them try to visualize the search results as a graph [49, 42, 41, 50]. Automatic graph drawing can be computationally very expensive [51] and is often only approximated for better performance. Lokoč et al. [41, 42] proposed to leverage the graph drawing to a force-directed layout [52] running in the browser, which saves server resources and provides better interactivity options.

The GUI can also be presented in 3D for better user experience. Experimental systems have been shown by Schoeffmann et al. for video browsing using a 3D carousel [53] and for image browsing on a 3D globe [54].

Other exploration systems have been developed in the last decade and evaluated in extensive surveys [55, 56]. Effective and efficient exploration is requested not only for common users, but also for special cases, like medical images or videos. Specialists searching for a specific frame in a video often spent long time doing mentally challenging work.

To compare different systems for video browsing under the same conditions, the Video Browser Showdown [57] is hold every year since 2012 as part of the Multimedia modeling conference. In this contest, researchers can compare the performance of their systems on a common dataset and on the same set of tasks. The evaluation results [58] describe the tools and show how they performed in direct comparison. Recent success of a novel approach shown by Lokoč et al. [59, 60] shows that even state-of-the-art tools can be outperformed and that the field of browsing tools still has a lot of room for improvement.

1.4.5 Efficiency of the exploration

When exploring through a collection, users expect to see the results almost immediately without noticeable delay. This can be a problem in large scale databases or even in smaller systems using a nontrivially expensive distance function like the SQFD. The sequential scan is not feasible in these scenarios and some form of index has to be used. Previously mentioned systems did not pay a lot of attention to indexing and as Beecks et al. stated [39], index support for multimedia exploration remains a challenge.

Previously mentioned MAMs are designed and optimized for range and $k-$NN queries. This may however not be sufficient for an exploration system that needs to support many other operations. Also, simply returning the most relevant objects disables the option to get insight of the collection and cannot support the exploration regime [45].

Lokoč et al. [61] focused on native exploration of metric indexes. Native exploration uses the inner representation of the structure instead of issuing traditional $k-$NN or range queries. This approach can support traversing the same structure with different strategies

and can be used with zero or almost zero distance computations. Comparison of these approaches will be shown in 2.2.

The ultimate goal is to have a structure capable of both efficient answering of exploration queries and a fast generation of similarity based layout, preferably without any distance computations. A baseline approach could be a generation of complete similarity graph between all objects and cutting of distances above a certain threshold. Exploration would then traverse the graph and return the relevant subgraphs as the result. However, this would require $O(n^2)$ computations of the distance function and additional space, which is not feasible even for smaller databases.

A different problem is the indexing of adaptive similarity models used in relevance feedback techniques. Traditional MAMs consider a fixed distance function and its change would break the functionality of the index. Bustos et al. and Skopal [62, 63] proposed modifications to MAMs that enable indexing not only for one distance function, but for a linear combination of possibly many distance functions. Those weights can then change during querying. This approach cannot deal with other than linear combinations of distances, such as the Dempster-Shafer mechanism mentioned earlier.

Chapter 2

Exploration with M-Index

As part of our work, we have investigated approximate search using the M-Index structure and researched the options of efficient multimedia exploration using the M-Index.

In the following chapter, we will briefly describe the structure of M-Index, show our experimental results for approximate $k-$NN search and present two approaches to multimedia exploration based on M-Index. First approach utilizes fast approximate $k-$NN search, second one exploits the structure of M-Index in the exploration process. Exploration approaches we have developed were compared in a user study called Find the image and the results are presented in section 6.2.

2.1 Structure of M-Index

Two key properties of the M-Index are repetitive partitioning of the metric space and mapping of all objects to a numeric domain. Partitioning divides the metric space into disjoint clusters, which enables pruning during search. Mapping to a numeric domain allows to store objects using well established structures, such as B^+-tree, and efficient retrieval using interval queries. The numeric key for an object consists of two parts. The integral part is based on the partition the object belongs to. The fractional part uses the distance to the closest pivot normalized to $\langle 0, 1)$. In our work, we have used a simplified variant of the M-Index only for the specialized cases of approximate $k-$NN search and data exploration. We were also running the index only in primary memory. For those reasons, we did not use nor implement the mapping to a numeric domain, which is required mainly for data in secondary memory, exact search techniques or distributed storage scenarios. When we refer to M-Index in following sections, we will describe our simplified variant of this data structure.

M-Index uses repetitive Voronoi partitioning to divide all database objects into groups by their relations to a global list of pivots. Relations to pivots are formalized using distance permutations from object to the global list of pivots. Two objects belong to the same group if they share the same prefix of that permutation. Authors of the M-Index propose several variants of it based on the length of that prefix.

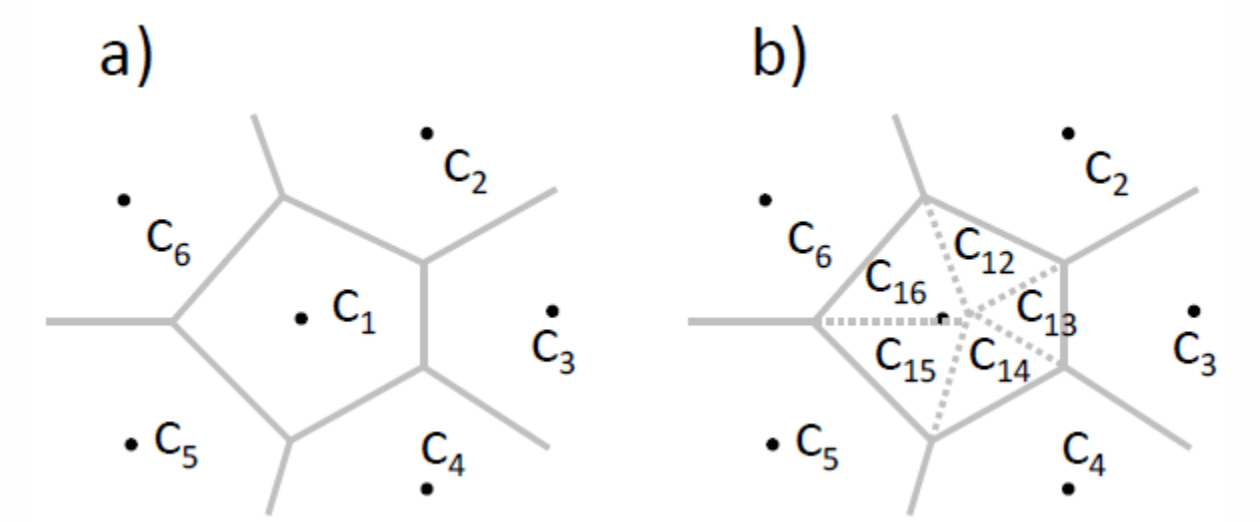


Figure 2.1: Repetitive Voronoi partitioning

Level-1 M-Index uses only the closest pivot to determine an appropriate group for an object. In other words, the distance permutation prefix used has length 1. We can see example of such clustering with 6 global pivots C_1-C_6 in figure 2.1a).

Multi-level M-Index applies further partitioning of the clusters. The M-Index with l levels uses permutation prefix of size l to assign each object to a cluster. With n pivots and l levels, M-Index could partition the space into $n \times (n-1) \times ... \times (n-l+1)$ clusters. However, this number is just theoretical and in practice the number of non-empty clusters is lower than this number.

Dynamic-level M-Index determines the length of the distance permutation prefix dynamically at runtime by the size of the appropriate cluster. With larger levels of multi-level M-Index, the space can be fragmented in a lot of small parts which brings additional overhead for query processing. To deal with this issue, dynamic-level M-Index applies further partitioning of a cluster only after it reaches a maximal capacity of objects. In the example in figure 2.1, partitioning begins with 6 groups in part a). When we insert more objects into the group C_1 and it reaches its maximal capacity, the node has to split into partitions with level 2 (i.e., the second closest pivot is considered for partitioning). Situation after the split is shown in part b) of the figure.

The structure used to store the actual clustering of the metric space in dynamic-level M-Index is called cluster tree. For static M-Index the cluster tree is balanced while for dynamic M-Index the cluster tree is unbalanced. In the root of the cluster tree, clusters determined by prefix of size 1 are stored (C_1-C_6 in figure 2.1a). If any cluster in the cluster tree is split, then a new node is appended to the cluster tree, where the item storing the split cluster is linked with the new node, forming a hierarchy corresponding to the repetitive Voronoi partitioning.

```
InnerNode.Insert(byte[] distancePermutation, data)
{
  var child = this.Children[distancePermutation[0]]
  if(child.ShouldSplit())
  {
    child = child.Split()
    this.Children[distancePermutation[0]] = child
  }
  child.Insert(distancePermutation.Skip(1),data)
}

LeafNode.Insert(byte[] distancePermutation, data)
{
  Store data together with the remaining
  part of its distancePermutation
}
```

Figure 2.2: Cluster tree insert

2.1.1 Algorithms for cluster tree

In our implementation, cluster tree has the form of an unbalanced tree with two types of nodes. Inner nodes do not contain any data and have a fixed-sized array of pointers to its child nodes. On $n-$th level of the tree, the children are divided into the child nodes according to $n-$th closest pivot. Leaf nodes contain the data and are referenced by a parent inner node. If the number of data items goes beyond a given threshold, by default 50 in our implementation, the leaf node is transformed into an inner node with the data divided into new leaf nodes. If the level of the node is already equal to the number of pivots, the algorithm has no more information for splitting the data into child nodes and the leaf buckets begin to grow in size instead.

Insert

Figure 2.2 show a polymorphic algorithm used for inserting data into our implementation of the cluster tree. The algorithm begins by inserting the data and its distance permutation into the root of the cluster tree. With each level, the first item of the permutation is used to determine the appropriate child. The Insert function is then called recursively with the distance permutation without the first item. Recursion ends in a leaf node, where it stores the actual data.

Approximate search

When issuing an approximate $k-$NN search, data has to be processed in an effective manner to achieve a high recall even when the search is terminated early. The structure of the cluster tree is used to return leaf nodes in an order that should approximate their closeness to the query where the only available/stored information are distances to pivots and corresponding distance permutations.

When the leaf nodes are returned to the $k-$NN algorithm, their data objects are filtered by the lower-bound distances computed from their distances to pivots. The filtering threshold (query radius) is computed dynamically as the distance to the $k-$th closest object changes in the current result set. Rest of the objects are compared directly to the query object by computing original distance computations and the result set is updated accordingly. The search terminates when a specified number of distance computations has been spent on direct comparison of objects.

Figure 2.3 describes the algorithm used for ordering of leaf nodes during approximate search. When searching for $k-$NN, we assume that objects closest to the query will probably be in the cluster in which the query would belong. Other candidates reside in the clusters defined by pivots which are closest to the query object.

In this algorithm, we calculate a penalty for each node determining the proximity of the cluster to the query object. Then we visit the nodes in order of this penalty and return them in case of leaf nodes or add their children to a heap ordered by the calculated penalty in case of inner nodes. Note that the penalty is divided by the current level to normalize differences between different levels of the tree. This means that even a leaf node deep in the hierarchy can be processed earlier then some level-1 clusters, if its distance permutation differs only slightly from the distance permutation of the query object.

The algorithm begins with the root node of the cluster tree and returns the leaf nodes lazily. This means that the algorithm proceeds with processing of the heap only when caller of the function requests more results.

This algorithm is based on the original method proposed by the authors of M-Index [37].

2.2 Approximate search using SQFD

In the bachelor thesis of Tomáš Grošup [64], we were observing precision and indexability characteristics of SQFD under changing values of the α parameter. This parameter is used inside the SQFD in the function f_s to determine the similarity between all pairs of representatives in the signatures using the formula $f_s(r_i, r_j) = e^{-\alpha L_2(r_i, r_j)^2}$. For precision, we measured MAP of exact search considering predefined ground truth of the collection. The indexability was determined by intrinsic dimensionality, which has been proven as a good approximation of the indexability of MAMs [65]. It is defined as $\rho = \frac{\mu^2}{2\sigma^2}$, where μ and σ^2 are the mean and the variance of the distance distribution for the entire collection. The lower this value is, the better are the indexability options for the collection. Our conclusion was that selected values of α can gain up to two orders of magnitude better

```
type DistancesToPivots = {Distances : float[], Permutation : byte[]}
ApproxSearch(DistancesToPivots query)
{
heap = {Node = MIndexRoot, Penalty = 0, Level = 0}
while(!heap.IsEmpty)
{
  min = heap.DeleteMin()
  if(min is LeafNode)
    //lazily return next next return value,
    //continue only when caller of the function
    //requests more values
    yield return min.Node
  else
  for(i = 0; i < min.Node.Children.Length; i++)
  {
    penalty = min.Penalty * min.Level
    penalty += |query.Distances[i] -
      query.Distances[query.Permutation[min.Level]]|
    penalty /= min.Level +1
    heap+= {
      Node = min.Node.Children[i],
      Penalty = penalty,
      Level = min.Level +1
        }
  }
}
}
```

Figure 2.3: Ordering of leaf nodes during approximate search

retrieval performance over the most-precise configuration only for a 5% loss of precision. It was concluded that indexability is an important aspect of the distance modeling process.

In our recent paper [38], we focused on retrieval precision of the SQFD under limited constrains. By limited constrains we mean a fixed number of distance computations given as a limit to the retrieval procedure. The number of distance computations is important especially in the case of nontrivially expensive distances, like $O(n^2)$ when using SQFD. We compared different values of the α parameter in approximate search using M-Index and we expected similar behavior as in the exact search scenario where the lower values of α resulted in better trade-offs. However, it turned out that under limited number of distance calculations the model trained for the best effectiveness (but low indexability) has given better results than a model optimized for good trade-off (sufficient effectiveness, good indexability). We have concluded that a high intrinsic dimensionality is not an obstacle for approximate search using M-Index and SQFD, and modeling a SQFD distance space for the best precision is sufficient. Let us note, we have also tested these observations for retrieval using models based on MPEG-7, where instead of changing a distance parameter, we were reducing the number of dimensions by omitting the less important ones. It was again shown that in approximate search with a limited number of distance computations, the best results were achieved when using the most precise model, even if it had high intrinsic dimensionality.

2.2.1 Experimental results

In our experiments, we observed the M-Index under different values of α supplied to SQFD when approximate search with fixed number of distance computations is used. We measured precision of the results for $k-$NN queries issued to two different collections, Profimedia and TWIC. For these experiments, a static 3-level variant of the M-Index was used.

Profimedia dataset [66] contains 21 993 images divided into 100 classes by a semi-automatical procedure and human verification. TWIC dataset [67] consists of of 11 555 images forming 197 classes, where each class represents images obtained by a keyword query to the Google images search engine.

Feature signatures for those collections were generated using a GPU extraction tool [68].

In the first experiment, we have measured the impact of different number of fixed distance computations on precision of the search. As we can see in figure 2.4, best results were always achieved by the highest value of α. This is the value which has best precision in exact search scenario, but also the worst intrinsic dimensionality. We can also see that even for ten distance computations, which really means just reading the first ten objects from the first M-Index bucket obtained by the described approximate $k-$NN search heuristic, the M-Index was able to achieve an interesting precision.

In the second experiment, we have fixed the number of allowed distance computations and focused on percentage of buckets visited during the search. As we can see in figure 2.5, higher values of α do not only achieve better precision, but also need much less buckets to visit because the distances are spent in first few buckets and then the algorithm stops. For lower values of α with a low intrinsic dimensionality, the triangle inequality allows to skip

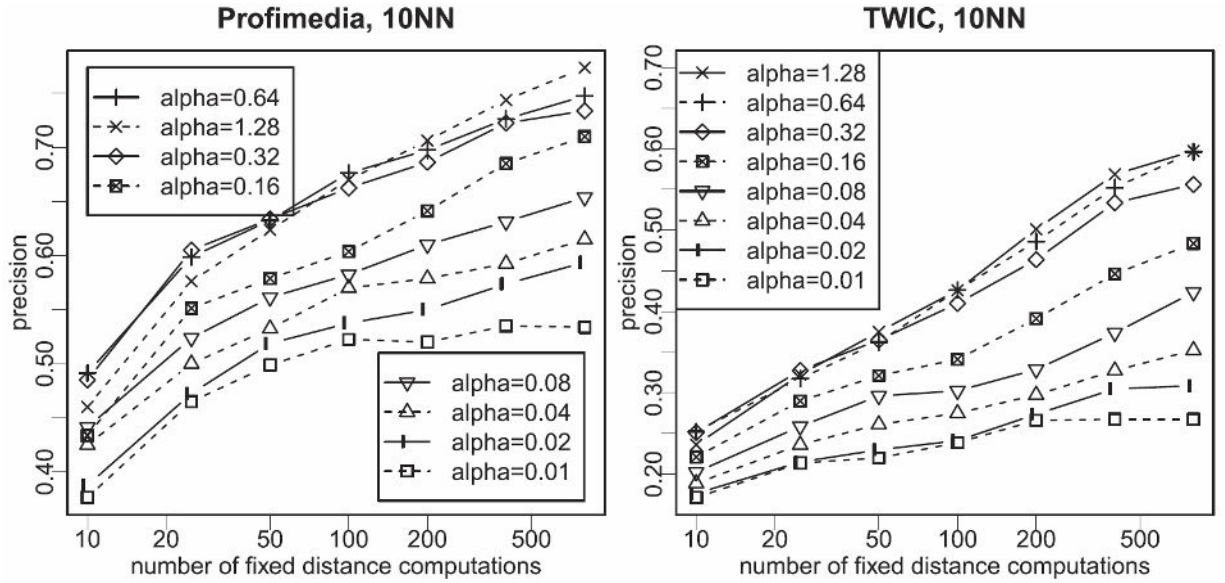


Figure 2.4: Results of the approximate search controlled by the fixed number of allowed distance computations

a lot of distance computation and the search has to go trough a lot of buckets to spent 500 distance computations. Terminating the search early also affects the overall performance of the search by reducing the amount of memory accesses.

To sum up the results, the configuration resulting in the best mean average precision but worst intrinsic dimensionality is highly competitive in approximate search.

2.3 Exploration techniques

We have utilized the dynamic-level variant of M-Index to create two approaches to multimedia exploration.

In the first approach, we have employed the fast approximative search with a fixed limit of allowed distance computations. This approach was used to iteratively query the database with $k - NN$ queries, where the query object is always selected by the exploring user. Thanks to the approximate search, the response can be generated almost in real-time and the result set contains a lot of relevant objects.

In the second approach, we have focused on browsing of the hierarchical structure of the cluster tree used inside the M-Index. It was necessary to extend cluster-tree by a preview function that returns example objects for a given node, since our implementation physically stores data only in leaf nodes and not in inner nodes. With this modification, it is possible to browse the index in a top-down manner from root to leafs. The user drives the direction of exploration trough the index by selecting objects of interest that always correspond to a node. This approach does not use any distance computations at all and generates the result almost immediately.

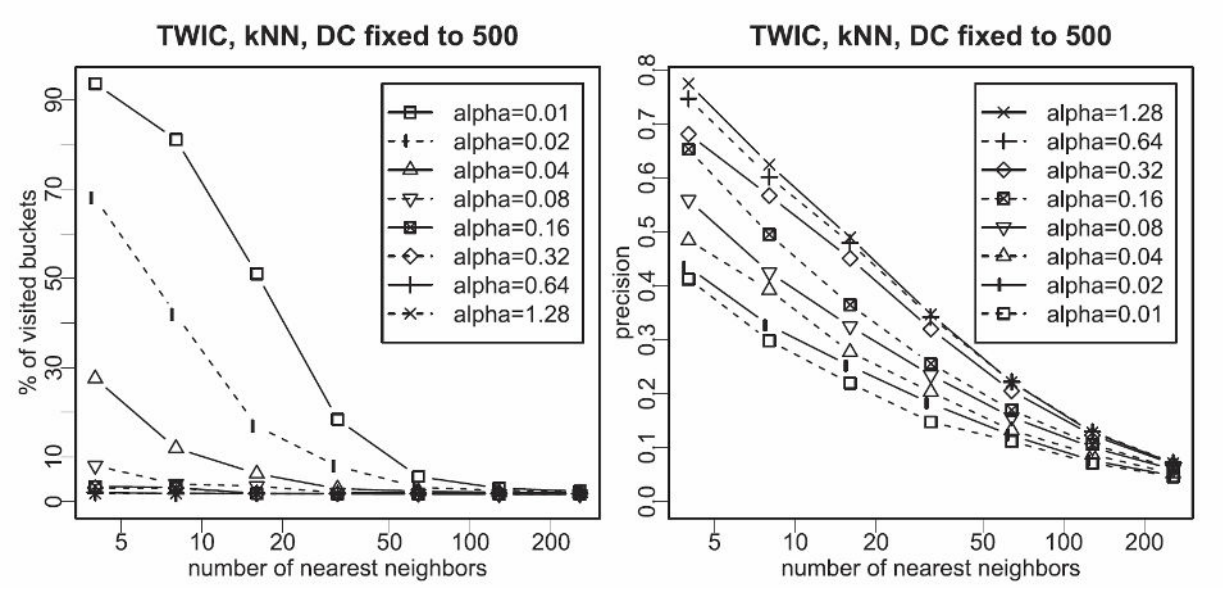


Figure 2.5: Precision and number of visited buckets for growing number of nearest neighbors

2.4 Iterative querying

Iterative querying is an approach to multimedia exploration that delegates exploration to $k - NN$ queries. We have utilized the structure of the cluster tree to generate the initial view for the exploration process and the approximate $k - NN$ search for fast yet precise query execution. We also believe that the approximate search can lead to a more variable result set than an exact search, which can be beneficial for the exploration process. In this section, we will present an algorithm that uses structure of the cluster tree to generate the initial view.

The initial view is computed by a recursive strategy that asks each node to return a number of representatives. The requested number is computed using the ratio of number of all node's descendants and all objects in total. Once the recursive function hits a leaf node, the first objects from that node are returned. A pseudocode for this strategy is described in figure 2.6. For the initial view, the GetExamples function is called for the root node with 50 samples to take.

2.5 Iterative browsing

Iterative browsing uses the structure of the index to execute all exploration queries, not just for generation of the initial view. Instead of issuing a $k - NN$ search during exploration, strategy for recursive traversing of the cluster tree described in figure 2.6 is used again. The data structure holds a map from returned objects in previous result sets to nodes they were returned from. When an exploration query is issued, this map is looked up to find a

```
struct Samples = (Node, list of objects)

GetExamples(InnerNode innerNode, int samplesToTake){
if(innerNode.CountOfNonEmptyChildNodes > samplesToTake)
  return (innerNode, first objects from samplesToTake children)

totalItemsLeft = innerNode.TotalNumberOfDescendantItems
result = {}
foreach(MIndexNode child in innerNode.Children)
{
  childTotal = child.TotalNumberOfDescendantItems
  canTake = samplesToTake * childTotal / totalItemsLeft
  samplesToTake -= canTake
  totalItemsLeft -= childTotal
  if(child is LeafNode)
    result+= (child, first canTake objects)
  else if(child is InnerNode)
    result+= GetExamples(child,canTake)
}
return result
}
```

Figure 2.6: M-Index initial view

node that was used in the previous exploration step. The response is build again using the GetExamples function as in the case of initial view. Instead of calling that function with the root node, the node found in the map is used. Once the exploration reaches leaf nodes, an approximate $k - NN$ search with zero distance computations limit is issued. However, this usually causes the exploration to cycle between a few nodes with no option to traverse to a different part of the cluster tree.

With this approach, all objects of the database can be reached by a limited number of operations from the initial view. The number of operations is limited by the height of the cluster tree, if both number of pivots and maximal size of a leaf node are lower than the number of objects displayed at each step of the exploration. Reachability is a qualitative criterion for exploration structures and is defined as the ratio of objects that can be reached by any number of operations and all objects in the database. Iterative browsing of the cluster tree guarantees 100% reachability. It also guarantees realtime responsiveness, since it does not use any distance computations at query time at all.

Chapter 3

Multi-model approach to multimedia exploration

This chapter is devoted to multi-model exploration of multimedia collections. We describe the concept of multi-model exploration and present two new approaches we have developed.

Although these two approaches could be used for any similarity models, we have described them on two specific similarity models for demonstration purposes. This allows us to highlight their differences and show scenarios, in which our approaches are beneficial. Similarity models we have used are also described in this chapter.

3.1 Multiple similarity models

As we have shown in section 1.3.1, many approaches to model similarity between multimedia objects exist. As you can see on the examples like SIFT or Feature Signatures, the models can use completely different techniques to create a feature space and to compute the similarity using a distance function. To obtain even better retrieval results, multiple similarity models can be combined together to form a new similarity model. This approach is called multi-model and is already used in content-based multimedia retrieval. One common example is the usage of MPEG-7 descriptors, which are rarely used solo and more often several of them are used in a weighted combination to produce better results [69]. It has been also shown [67] that various combinations of basic MPEG-7 descriptors using L_1 metric and Feature Signatures using SQFD outperform the mean average precision (MAP) of any of the models used alone in content-based image search.

3.1.1 Multiple modalities

The term multi-model should not be confused with multi-modal which refers to using multiple modalities to perform search. Each modality is a different source of data with a different form. The multi-modal approach is used in video retrieval a lot, where search systems combine audio features, visual features from video frames, transcript from speech

recognition or other information [70]. It has been utilized for image retrieval as well by combining visual features and textual annotation and was shown to be successful for both generic image search [71] and specialized search for online shopping [72]. In this work, we have not focused on the multi-modal approach as we have focused primarily on the content-based exploration of data without any external annotation.

3.2 Similarity models used

We have decided to combine two modern similarity models for images, both of them achieving good results in content-based image retrieval and yet using completely different mathematical models underneath. The first similarity model uses feature signatures and the SQFD to compute distances between signatures. We will refer to it as $FS + SQFD$. The second similarity model uses SURF detector to extract local features from an image, VLAD algorithm to aggregate the features into a single normalized descriptor and cosine distance function to compute dissimilarity between VLAD descriptors. We will refer to this model as $SURF + VLAD$

We tried two ways of combining these two similarity models together – distance combination and mixing of results.

3.3 Distance combination

This approach uses a combination of distances from the similarity models $FS + SQFD$ and $SURF + VLAD$ to form a new similarity model. At the beginning of the exploration, a random subset of the database is returned as the initial view. When the user explores, 50 nearest neighbors in the new similarity model are returned.

3.3.1 Weighting scheme

First we started with a simple non-weighted combination. This was, however, shown to be ineffective and produced confusing results, since the distance values returned from the individual models have different meanings. The distribution of those distances is also different and simple rescaling to a normalized value would not help it. We chose to utilize a dynamic weighting scheme determined by progress of the user to set weights for the individual models. For the beginning, the weights are selected according to observed values for the distances among nearest 50 objects. In our case, it was detected that $FS + SQFD$ produces three times smaller distance values than $SURF + VLAD$ on average. We set the weighting scheme $3 \times (FS + SQFD) + 1 \times (SURF + VLAD)$ as the default value for the new similarity model. During the exploration, new weights are generated.

When an object is returned to the user, it is passed together with metadata information M that holds the original distances to the query object in the basic similarity models we are combining together. $M_{SURF+VLAD}$ holds the distance from the returned object to the query object in the $SURF + VLAD$ similarity model. $M_{FS+SQFD}$ holds the distance

between the same objects in the $FS + SQFD$ similarity model. When user proceeds with exploration, this information is parsed out of the metadata of the query object. The ratio of the original distances determines the new weighting scheme for the current exploration step. To smooth the differences in edge cases, a small constant value is added to both of the distances, so the ratio never reaches extreme values and both underlying similarity models always participate in the combined similarity model at least partially. The formula for the weighting scheme is as follows:

$$\alpha = \frac{0.05 + M_{SURF+VLAD}}{0.05 + M_{FS+SQFD}}$$

$$new\,weighting\,scheme = \alpha \times (FS + SQFD) + 1 \times (SURF + VLAD)$$

3.3.2 Relevance feedback

This dynamic approach can be seen as a relevance feedback technique that detects user's search intention by inclining more to one or the other similarity model. When user focuses on objects with similar color distribution, the $FS + SQFD$ returns smaller values for distances, as the signatures are likely to be similar as well. The $SURF + VLAD$ model, on the other hand, does not focus on color at all and will likely return high distance values if there are not common keypoints between compared images. This will produce a high value of α (for example ten or more) and incline to the $FS + SQFD$ model more, since even small differences in this model will outweigh the differences in the $SURF + VLAD$ model. The opposite case can occur when user explores via an object that shares almost no color information with the previous query object, but exhibits the same structure. This can be for example two bottles of wine with different color and different background, which are not similar in the $FS + SQFD$ model, but share a lot of common keypoints detected by the SURF algorithm. This will result in a value of α smaller than 1 and the $FS + SQFD$ model will be suppressed for the current exploration step.

This relevance feedback technique does not require to remember exploration history and does not need to track user's steps. The exploration function does therefore not hold any kind of mutable state and is functionally pure. This means that for the same query object with the same metadata the same results will always be returned. This property is important for horizontally scaled scenarios, in which more computers response to requests. With pure functions, it does not matter which one of the machines answers the request and the distribution of requests between machines can be controlled e.g. by a load balancer.

3.3.3 SURF drawback

We have observed that for images without any complex structure, the SURF algorithm detects no keypoints and this results in a VLAD vector full of zeros. When it is compared against other VLAD vector using the cosine distance, division by zero occurs and NaN result is returned. In those cases, we replace the NaN results with a constant value 2.0

(the maximal value for the cosine distance), and let the $FS + SQFD$ model define the similarity alone.

Figure 3.1: Image with no keypoints detected by the SURF algorithm

3.3.4 Indexing

In our prototype, we used no indexing structure for this approach and search was performed via a simple sequential scan over the database. When the Profimedia collection is searched, the exploration query takes 450 ms on average, which is almost unnoticeable for the user. For larger collections, an index structure could be utilized. To utilize any of the MAMs, the combined distance used in our approach would have to fulfill metric postulates. The SQFD is already a metric. Cosine distance does violate the triangle inequality, but could be changed to deviation metric mentioned in section 1.3.1 while preserving the same ordering of objects as cosine distance does.

A linear combination of distances preserves the metric postulates if the weights are non-negative, which is true for our dynamic weighting scheme. To index a dynamically changing distance function, one of the proposed modifications to MAMs for dynamic distances could be utilized [63].

3.3.5 Similarity based layout

To generate similarity based layout, we used a component [41, 42] that computes complete similarity graph and passes it to the particle physics model which distributes images on the screen. To generate the similarity graph, a distance function has to be passed in to compute the values of edges between objects.

When we tried to pass in the combined distance function, the screen did not form clusters which could be visually recognized from a high-level point of view. As it turned out, the $FS + SQFD$ similarity model fits much better to generation of the similarity graph. It clusters images with similar color distribution together and those clusters can be quickly observed by human vision. The $SURF + VLAD$ model, on the other hand, focuses a lot on details inside the images and the clusters do not make any sense in a high-level point of view. The similarity inside those clusters can be seen first after noticing the common details between them.

For those reasons, we ended up with generating the similarity graph based only on the $FS + SQFD$ similarity model.

3.4 Mixed results

The second approach does not form a new similarity model. Instead, it issues queries to the underlying $FS + SQFD$ and $SURF + VLAD$ models and mixes the returned results together. This way it effectively deals with the mismatch of meaning of the values returned by distance functions, and it does not need any weighting scheme to adjust the distances.

Figure 3.2: Query object

When the exploration starts, the same random subset of the database as in the case of distance combination approach is returned. After the first exploration step, both of the similarity models are asked to fill half of the exploration result. In figure 3.3, you can see a result of an exploration step via the query object displayed in figure 3.2. The objects returned by the $FS + SQFD$ similarity model are displayed with a red border for demonstration purposes. As you can see, they exhibit the same color distribution as the query object did. The object from the $SURF + VLAD$ similarity model, displayed without any border, have different colors than the query object and yet we can see a lot of relevant objects among them. Mixing of results extended the results by new objects that would never be displayed with the use of a single similarity model.

3.4.1 Relevance feedback

However, the mixing of results comes at a cost. Once the user wants that kind of similarity he is interested in, the mixed results may be not useful to him and may just take space for the objects he finds relevant. To deal with this issue, we introduced a relevance feedback technique that dynamically changes the number of objects returned by a single model.

By default, the $FS+SQFD$ similarity model returns 25 nearest neighbors to the query object. The $SURF + VLAD$ similarity model is used to fill the result set with nearest neighbors which are not already present up to total number of 50 objects. For each image in the result set, we store metadata M with the positions within a list of all database objects ordered by the distance to the current query object for both similarity models. To do so,

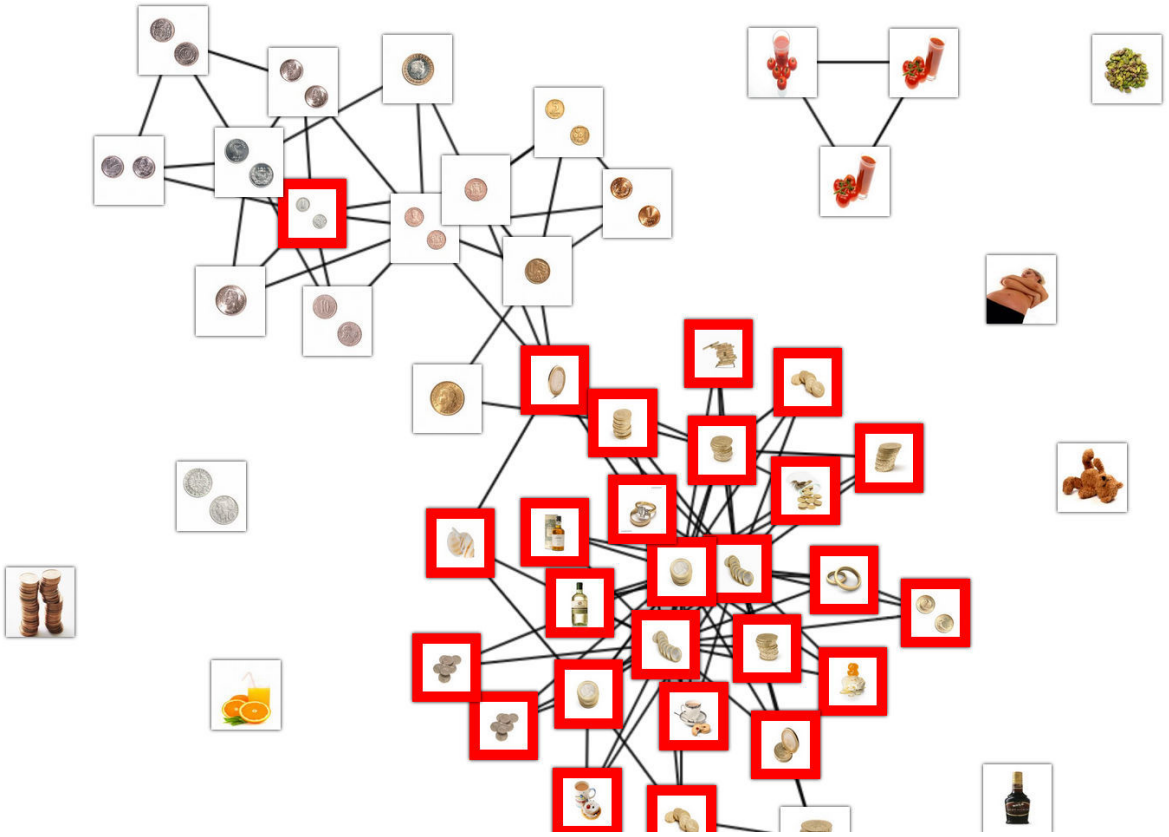


Figure 3.3: Exploration with mixed results

we have to compute the distances to all database objects using a sequential scan and order the results. $M_{SURF+VLAD}$ contains the position of the returned object within the list of all database objects sorted by the distance to current query object in the $SURF+VLAD$ similarity model. $M_{FS+SQFD}$ is computed the same way, just from the list sorted by the distances to current query object in the $FS+SQFD$ similarity model instead.

When an object with the metadata M is used as a query object, number of nearest neighbors for the $FS+SQFD$ model is computed using this metadata. The better the position of the query object in $FS+SQFD$ model was compared to the position in $SURF+VLAD$, the more objects can $FS+SQFD$ model return in this exploration step and vice versa. The $SURF+VLAD$ similarity model is always used to fill the rest of the result set. Since the Profimedia collection has a size of 21 993 images, ratios between positions within individual similarity models can be quite huge. To soften this ratio, we used fourth root of the ratio instead. To always mix at least some results from both similarity models, the ratio is kept within the range $< \frac{1}{9}; 9 >$. In the case of 50 images per screen, this means the minimum of 5 objects per similarity model and the maximum of 45 objects per similarity model. The formula for calculating the ratio ρ from metadata M and to calculate the

number of nearest neighbors $k_{FS+SQFD}$ for the $FS + SQFD$ model is as follows:

$$\rho = \min\left(\max\left(\sqrt[4]{\frac{M_{SURF+VLAD}}{M_{FS+SQFD}}}, \frac{1}{9}\right), 9\right)$$

$$k_{FS+SQFD} = \frac{\rho \times 50}{\rho + 1}$$

We can observe that for the initial value of $\rho = 1$, the $k_{FS+SQFD}$ is computed as $\frac{1\times50}{1+1} = 25$ and both similarity models have to fill half of the result set. As the exploration continues, the ratio changes for each step. When user selects an object similar to the current query only in the $FS + SQFD$ model and not similar in the $SURF + VLAD$ model at all, the ratio immediately changes in favor of the $FS + SQFD$ model. User is still given the option to explore via objects returned by the $SURF + VLAD$ model, since it is guaranteed that it participates in the result set (at least 10% of displayed objects).

3.4.2 Indexing

In our prototype, we also used sequential scan of all objects in the database to perform search. The query time is a little higher than in the distance combination approach, about 700 ms on average. For metadata generation, we have to hold two ordered lists of all distances in memory and also two lookup tables from object to position within the result set, which increases memory requirements and affects the overall performance.

The indexing of the individual $k-$NN searches could employ any of the known indexing techniques after we turn cosine distance to deviation metric to fulfill the triangle inequality. However, for metadata generation, we would need a structure that is able to return at least approximate position of an object within results, without computing all distances as in sequential search. A hierarchical index comes to mind, such as M-Tree or PM-Tree. Having the space divided into clusters, the position could be approximated by counting sizes of clusters that have all objects definitely closer to the query object just by calculating the distance to the center of the cluster. In addition, the distances to centers of clusters could be already computed in the first phase of the $k-$NN search. Such algorithm would however require modification to one of the indexes and is outside of scope of this work.

3.4.3 Similarity based layout

For the generation of similarity based layout, basically the same reasoning as for the distance combination approach applies. In advance to the first approach, we also have the positions stored as the metadata. This makes, however, no sense in the generation of a similarity graph, since none of the feature spaces has a natural ordering of objects. Closeness of positions to the query object does not give any guarantee about similarity between two objects.

Therefore, we also ended up with using just the $FS + SQFD$ similarity model to generate the similarity graph.

3.5 Experiments

We have compared our two multi-model approaches together with other exploration strategies we have built. The testing scenario and the evaluation results can be found in section 6.2.

Chapter 4

Motivation to create an exploration framework

To develop and test new approaches to multimedia exploration, we needed a codebase with all the necessary parts to create an exploration system. We needed something to turn our research ideas into visual applications without a lot of plumbing code and use those applications to measure the impact of these ideas.

4.1 Functional requirements

Developing such system usually requires several steps, and each of them can be complicated itself. For example, a system might have to deal with the following steps:

- Obtaining metadata for a collection, or downloading it online.
- Download of the raw multimedia data, such as images or videos.
- Storing the raw data.
- One or more feature extraction functions.
- Distance evaluation by one or more distance functions.
- Building a data structure for extracted features and efficient insertion in this structure.
- Building the Page zero.
- Generation of similarity based layout.
- Generating the GUI with controls.
- Communication and data transmission from back-end to front-end.

- Data conversions between storage format, communication format and display format.
- Processing of different types of exploration queries.
- Efficient utilization of data structures for exploration queries.
- Communication from front-end to back-end and utilization of the relevance feedback.
- Managing concurrent access to shared resources.

As we can see, this requires a tremendous amount of programming work. And typically, a researcher is only focusing on a small subset of these steps, often even just on one. But he still needs to be able to build an end-to-end system to prove and validate the ideas in practice, because exploration is an interactive process with focus on the user. Even if we can automatically measure performance, data size, system throughput and lots of other metrics, we cannot automatically measure the main problem - whether user's search intention was satisfied - without having a user actually using the system.

4.2 Extensibility

What we wanted to create was a prepared framework for multimedia exploration. Developers could plug in just the one part they are interested in and keep the rest on the framework, which would already contain a small set of existing implementations for any of the basic concepts. After plugging in their part, they would have an application they can use to present the effect of their part and easily measure and collect data they are interested in.

An example situation is a developer coming up with a new index which seems to be perfect for exploration queries. He does not want to build a whole new system, so he plugs in the index by implementing a specified interface and switches it on as the default index implementation. After that, he can launch the application and let human testers evaluate the influence of this new implementation in a predefined scenario.

A scenario we used in our comparison of data structures for image exploration [61] was using ground truth provided by the collection we were using [66]. This ground truth has metadata for each object saying to which image class it belongs, what are the default query objects for search tasks and how big each image class is. This information can be compared against results of an exploration session and determine how many images of the given class the user has found, how many image classes he visited in total etc. A more detailed description of our testing scenario will be given in section 6.2.

4.3 Graphical user interface

Since exploration is a user-centered interactive process, the GUI is a very important aspect of exploration systems. The framework should provide a GUI by default and represent

explored data in an intuitive way, preferably utilizing the similarity based layout, which is a popular choice in demo applications for multimedia exploration[48, 39].

4.4 Non-functional requirements

Besides the core functional parts we wanted the framework to provide, there were also some non-functional requirements. They would affect mainly the programmer using the framework and could help him with some common operations. There are also some concepts which should be kept in mind when developing the framework according to our opinion:

- Caching of data used across several requests and not retrieving it every time.
- Persistent caching of data that is costly to compute, such as extracted features from images downloaded from a web page. These results should survive the restart of the application so they do not have to be computed ever again.
- Support for concurrent updates of core structures.
- Utilization of multi-core environment in CPU-intensive parts of the framework.
- Monitoring of operations happening in the back-end.
- Performance tracing for different kinds of operations.
- Prepared infrastructure for automatic integration testing.

4.5 Related work

We investigated existing systems to see if we could find anything that at least partially fits our needs. A lot of multimedia exploration systems have been proposed in the past [56, 55], but they do not offer the kind of extensibility and interchangeability of components we needed. Also, they might not be publicly available at all. Most often, they are just singe-purpose demos and not general systems, like our demo showing image exploration using online feature extraction and re-ranking [41, 42]. Also, they often skip a few aspects that are important, such as the possibility to work with larger databases and support for modern indexing methods.

For some parts of the system, an existing specialized library might be useful. For image processing and feature extraction, the OpenCV library could be used [73]. It is an open source computer vision library that contains hundreds of algorithms that can be applied to content-based image retrieval as well. Another interesting library for content-based image retrieval is the Lucene Image Retrieval library [12], that can not only extract various image features, but also indexes the results using a Lucene [74] index.

For metric space indexing, the Siret Object Library (SOL) could be utilized. It is a framework for efficient metric and non-metric similarity search developed by the Siret Research Group (SRG).

An open source collection of metric space indexes can be also found in the results section of the Laboratory of Data Intensive Systems and Applications (DISA). The implementation of M-Index presented there is used to power the large-scale image search system MUFIN [14] which can search for visually similar images in a collection of 100 million images.

Chapter 5

Implementation of exploration framework

After collection of the functional and non-functional requirements, we decided to build a new framework for multimedia exploration. Author of this thesis was the software architect of the framework and designed interfaces of the core components and the main data flows. Besides that, he also implemented the infrastructure needed to support all the non-functional requirements and integrated them into the rest of the framework. Implementation of the individual components was done within a small team of software developers, each responsible for a specified region of the framework. Besides the author of this thesis, the team consisted of Přemysl Čech, Jakub Kinšt, Miroslav Macík and Lukáš Navrátil.

5.1 Framework design

The framework is deployed as a single web server together with several dynamically loaded libraries. It was developed in the C# 5.0 language and the web layer was build using the ASP.NET MVC and ASP.NET WebAPI frameworks.

It consists of many loosely coupled components. It is written in object-oriented manner and utilizes some concepts from the functional paradigm as well, especially with the use of C#'s LINQ feature[1]. When creating the system, we tried to develop the parts according to SOLID principles [75]. SOLID is a set of 5 principles for a good object-oriented class design first summarized by Robert C. Martin:

- Single Responsibility Principle: A class should have one, and only one, reason to change.
- Open Closed Principle: You should be able to extend a classes behavior, without modifying it.

[1]Language-Integrated Query (LINQ) is a feature that allows to write queries directly in the C# language.

- Liskov Substitution Principle: Derived classes must be substitutable for their base classes.
- Interface Segregation Principle: Make fine grained interfaces that are client specific.
- Dependency Inversion Principle: Depend on abstractions, not on concretions.

With the dependency inversion principle in mind, we designed our components to require their lower-level dependencies as constructor arguments, and not to create them directly. This makes it possible to create highly reusable components in which specific parts can be changed without touching the code of the component [76]. This is commonly known as the Dependency Injection pattern.

The application infrastructure is then responsible for resolving fully instantiated graphs of cooperating objects. To resolve those graphs of objects at runtime, we use an Inversion of Control (IoC) Container [77], in our case the Autofac library. The container does three important things during application lifetime for each object:

- Register: Before any object is created by the container, its dependencies must be registered. The container allows both manual and automatic registration of classes, interfaces and even generic templates. If more implementations for the same interface exist, the interface is registered manually, e.g. from the current configuration setting. During registration, the lifetime of objects is also specified. In our work, we used three basic types of lifetime: One instance for entire application, one instance per web request and one instance per each dependency.
- Resolve: When an instance of a class is requested by the container, it begins to recursively resolve its dependencies and call their constructors with the correct arguments. Dependencies are looked up in the current collection of registrations and dependent on the lifetime of the registration either reused or created from scratch. The container also allows to resolve factory methods that can be used to create the objects by supplying a subset of the constructor arguments and letting the container to resolve the rest.
- Release: At the end of each object's lifetime, the container is responsible for correct releasing of the object. This is done by invoking the $.Dispose()$ method if the object implements the $IDisposable$ interface. This is crucial for objects holding references to shared resources or objects not managed by the .NET runtime, such as file handles or database connections.

5.2 Basic components

Figure 5.1 shows a high-level flow of messages between individual abstract components. It is not bound to any specific implementation, it is just an outline for the common flow and a demonstration how interfaces are meant to cooperate.

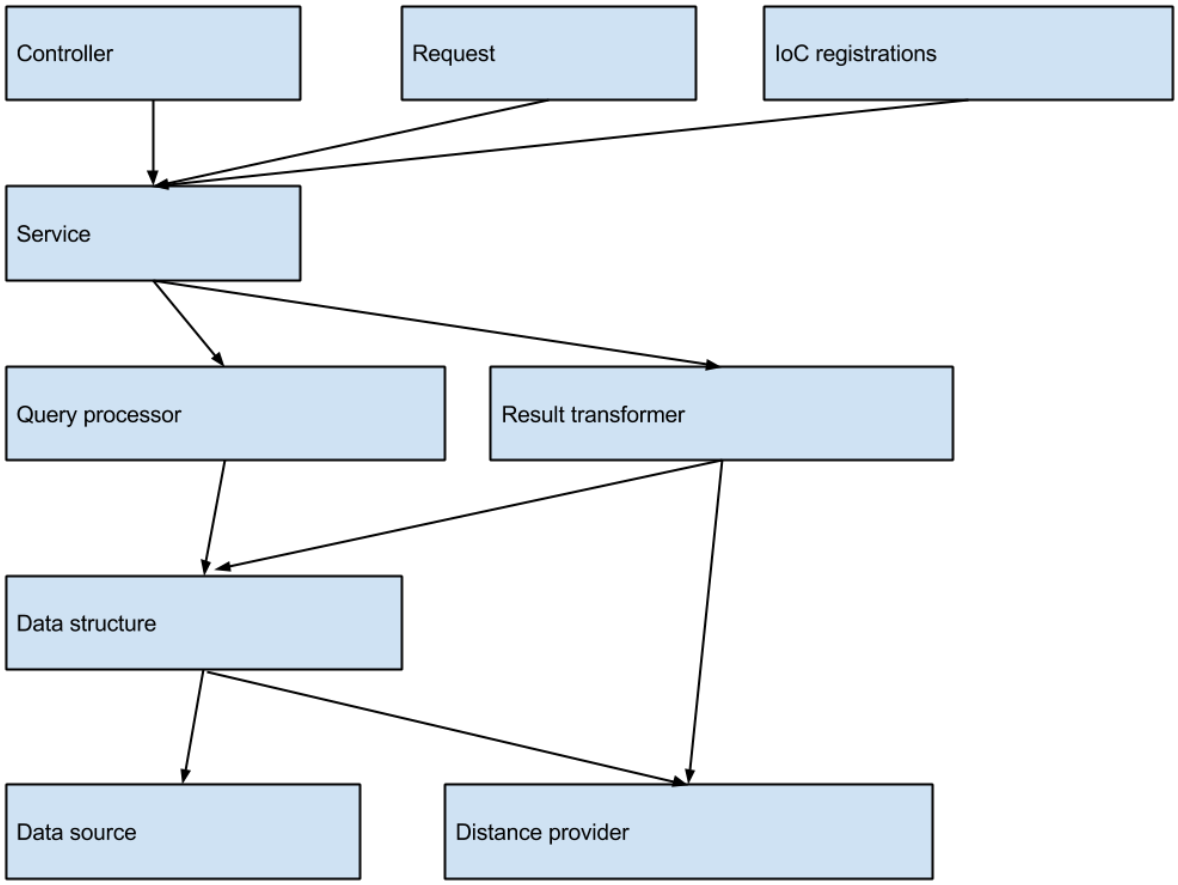


Figure 5.1: Flow of messages between abstract components

Components are ordered from the high-level ones starting at the top to the low-level ones at the bottom. An arrow means having the component at lower-level of abstraction as a dependency and calling some of its methods.

- Processing starts with an incoming request. It is passed to a controller and current configuration settings are parsed out.
- Current settings are looked up in global Inversion Of Control registrations and marked as default for the processing of current request.
- Controller uses the IoC container to resolve an instance of Service component with the correct dependencies and calls its processing method.
- Service takes the exploration request and passes it to query processor.
- Query processor invokes exploration queries on a data structure and combines the individual results to form an unified exploration result.

- The data structure is either picked up from cache or created on demand using a data source.
- The data source might return already precomputed results or start with additional operations such as downloading and image processing.
- Data structure uses the currently registered distance provider to answer the exploration query.
- Service passes the exploration result to a result transformer to form a view-model for the current application.

This basic description shows only abstract concepts, which may have several implementations with different flows of data. An application might for example require additional processing to handle its requirements, and add some components into the process.

A more complex example is the case of our exploration portal publicly available online at http://herkules.ms.mff.cuni.cz/ which downloads images from 3rd party services and processes them on the fly, while delivering response in a few seconds. Figure 5.2 is similar to the first one, but instead of showing abstract concepts, specific implementations of classes are shown. Documentation for public APIs, request and response objects and their dependencies can be found in the online API help available at http://herkules.ms.mff.cuni.cz/Help.

- Exploration begins with passing a request, *ExplorationQuery*, to the controller.
- The query contains a configuration token which is used as a key in the database of stored configuration settings.
- The settings is looked up and all of the contained components are looked up and registered using Autofac IoC container.
- Autofac IoC container is used with the current registrations to resolve an instance of $MultiQueryService$, and the controller passes the current query object to it.
- $MultiQueryService$ uses $MultiQueryProcessor$ to execute the query.
- All query objects are executed in parallel in case of multi-query or panning.
- $MIndex$ is either looked up in a cache, or created from scratch.
- To create a new $MIndex$, $WebDataSource$ is asked for data.
- $WebDataSource$ asks $ImageResultsSynchronizer$ to return the first batch of images
- It executes ten queries to Bing to load paginated metadata for current text query (e.g. “Jaguar”).

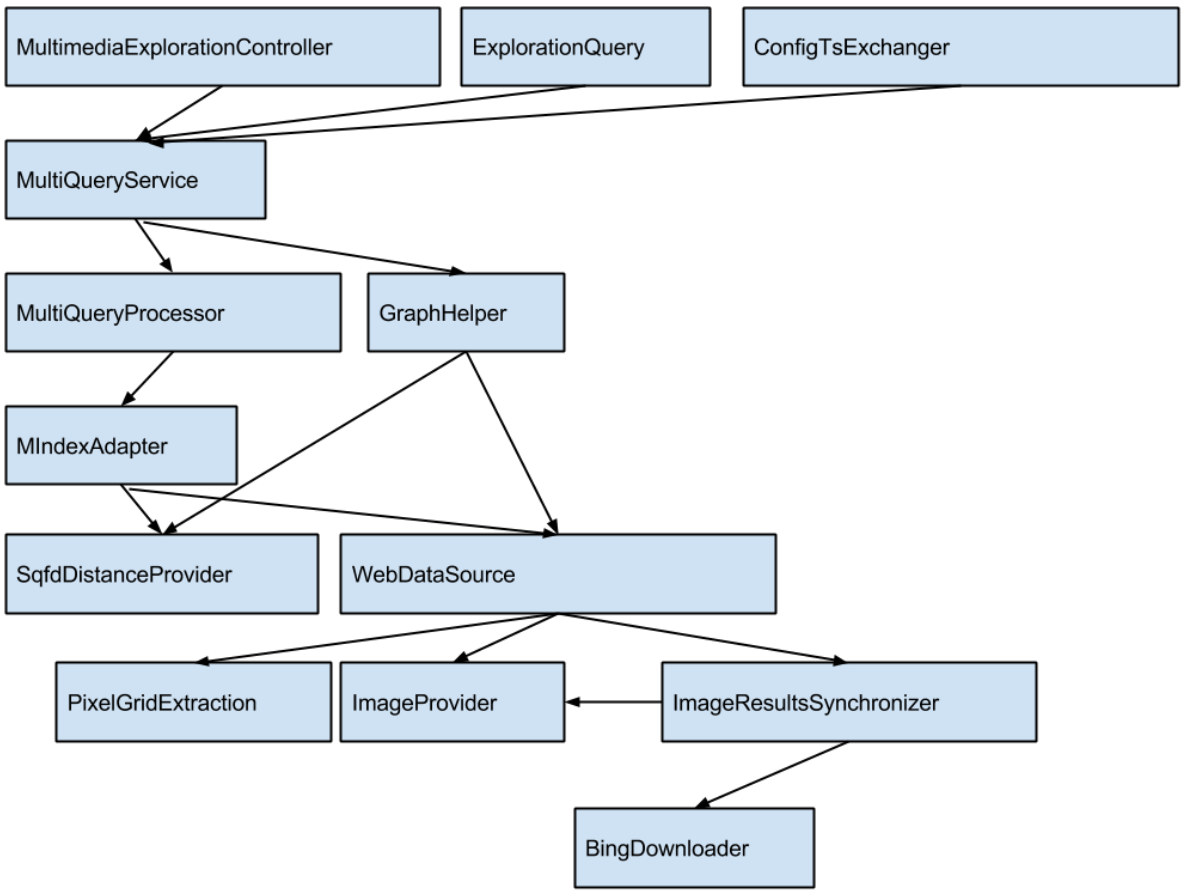


Figure 5.2: Specific flow for the exploration portal

- The first query that completes is passed to *ImageProvider* to load up 50 bitmaps according to Bing metadata. It is either found on the hard-drive or downloaded from the web.
- The images are passed to *PixelGridExtraction* component, which creates feature signatures using image thumbnails. These signatures are accessible as properties of the $WebDataSource$.
- The rest of Bing queries execute on background threads within the same pipeline and enlarge the $WebDataSource$ once completed.
- $MIndex$ chooses a set of pivots randomly and uses the feature signatures to compute distances to pivots using $SqfdDistanceProvider$ and stores the data in its internal structure.
- $MIndex$ answers the query for a query object.
- *MultiQueryProcessor* combines individual results and orders items by the position within each returned set. If panning was used, items from the previous screen are excluded. 50 items with the best score are returned.
- *MultiQueryService* uses *GraphHelper* to prepare the result for our implementation of similarity based layout, the particle-physics model.
- Edges for similarity graph are computed using $SqfdDistanceProvider$ and object metadata for images is loaded from $WebDataSource$.
- The result of exploration is returned and converted to JSON format using ASP.NET WebApi response formatter.

5.3 Existing implementations

Together with the core infrastructure and abstract interfaces we also provided several implementations for each of the components.

5.3.1 Image collections

The users can choose from one of three basic sources of images.

Profimedia

Profimedia [66] is a static collection consisting of 21 993 images. It used mainly for testing purposes and evaluation of performance and effectiveness of individual approaches.

Bing

Bing is a dynamically downloaded collection from text-based image search engine Bing Images http://www.bing.com/images. The user inputs a textual search query and first 500 images returned for that query are provided for exploration. The textual search also allows typical modifiers for information retrieval systems, such as negation ('apple -mac') or URL restriction ('bmw site:ebay.com').

Facebook

Facebook is an option to explore own images uploaded to the social network Facebook https://www.facebook.com/. When user logs in to the web portal using his Facebook account, permissions to his albums are asked for. The portal then dynamically downloads all the albums and their images and allows to explore trough them.

5.3.2 Data source

Each of the image collections has three options to convert the images into data and compute similarity between them.

Feature Signatures

This option extracts feature signatures from the images and computes the similarity between them using the Signature Quadratic Form Distance. For the Profimedia collection, the signatures were precomputed using a GPU extractor [68] and are loaded from a file. For dynamic collections, the signatures are computed online from thumbnails of the images.

Bag-of-Words

The bag-of-words [16] approach utilizes a shared vocabulary of visual words to represent each image by a vector of occurrences of each of the individual words.

During the vocabulary training phase, the algorithm first creates descriptors for the images and then uses all of the extracted features to create a shared vocabulary of descriptors. The features are clustered using k-means clustering algorithm into a fixed-size vocabulary, by default 10000 features.

After the vocabulary is created, each feature from each image is assigned to a nearest neighbor in the vocabulary. This is done using the M-Index structure to speed up the process. The image is then represented by a vector of the same size as the vocabulary is, having the number of occurrences of the corresponding visual word in each dimension.

Distance between two vectors is computed using the cosine distance function.

VLAD

VLAD descriptor [17], vector of locally aggregated descriptors, is a state-of-the-art extension to the bag-of-words model. Instead of storing the number of occurrences for each

dimension, a sum of differences between a feature and vocabulary word is computed. The model uses a smaller vocabulary, by default 32, and also computes the distance between two vectors using the cosine distance function. Since the VLAD descriptor can contain negative values, the range of values returned by the cosine distance is from 0 to 2. In the standard bag-of-words approach, number of occurrences cannot be negative, so the vector cannot contain negative numbers. Thus the range of values returned by the cosine distance is from 0 to 1 in the case of standard bag-of-words.

5.3.3 Feature extraction

For bag-of-words and VLAD data sources, three different feature extractors are offered. Each of them extracts a set of features of each image. SIFT and SURF are state-of-the-art extraction techniques, PixelGrid is our modification to feature signature extraction. Although feature signatures are meant to be compared with an adaptive distance function such as SQFD, it is also possible to use them in bag-of-words techniques like we did.

SIFT

SIFT is an algorithm to detect and describe local features of an image. It detects interesting keypoints in an image and then describes each such point with a 128-dimensional vector of floating point numbers. The results are robust to scaling, rotation and illumination changes [15].

We used the EmguCV library for both keypoints detection and feature extraction.

SURF

Speeded Up Robust Features(SURF) is a local feature detector inspired by the SIFT algorithm. It uses several numerical approximations and is several times faster than the standard SIFT algorithm. It is also claimed to be more robust to image transformations than the SIFT algorithm [78].

We also used the EmguCV library for both keypoints detection and feature extraction.

PixelGrid

The pixel grid is a simplified variant of feature signature extraction mentioned earlier. The original algorithm locates clusters within the image by random Gaussian sampling and k-means clustering. Then the centroids are used for extraction of low level features. This algorithm is fine in typical scenarios where the extraction happens once during a preprocessing phase and the feature signatures are stored in a persistent storage. For the collections obtained online (Bing, Facebook) a more performant extraction is necessary

Our modification works with two extremely small thumbnails of the input image. The minification also serves as a noise reduction step. One of the thumbnails has a little higher resolution and is used to extract features from the middle of the image. The other one has a lower resolution and extracts features from the edges of the image, which is

Figure 5.3: Clustering feature signature extraction

usually just background. This modification skips the most CPU-intensive part, k-means clustering, entirely. Instead of clustering, each pixel from the two small thumbnails is used to describe the image. To simplify the process even more, we skipped the calculation of contrast and entropy. Figure 5.3 shows the visual representation of extracted feature signatures for three images using the original clustering algorithm, figure 5.4 below shows the same representation of extracted feature signature using the simplified, non-clustering modification of the algorithm.

5.3.4 Exploration structures

An exploration structure has to provide an initial view and executes exploration queries. They can be used in combination with any of the previously mentioned components.

Sequential $k-$NN

This is a baseline structure that only executes $k-$NN queries when exploring and presents a random subset of the database as its initial view. During exploration, it always computes distances to all objects in the database and does not utilize any kind of indexing.

Pm-Tree $k-$NN

This structure uses the Pm-Tree metric index. Exploration queries are translated into exact search $k-$NN queries. Initial view is formed by the root of the index structure and its direct children.

M-Index $k-$NN

This structure uses the iterative querying strategy for M-Index described in section 2.4.

Figure 5.4: Non-clustering feature signature extraction

M-Index traversing

This structure uses the iterative browsing strategy for M-Index described in section 2.5.

Pm-Tree traversing strategies and Static M-Tree

The framework provides three other strategies for exploring Pm-Tree and one strategy for exploration of static M-Tree. Their description is outside of scope of this work and their details can be found in the thesis of Přemysl Čech [79].

5.4 Implementation details

Implementation details for the core components and other parts of the framework as well can be found in the Programmer's documentation, which can be found on the attached CD-ROM. It presents the key classes, describes extensibility points and shows several short code examples. It describes how we dealt with generic in-memory and persistent caching, performance monitoring, GUI creation and other functional and non-functional requirements. The documentation was written by the entire team of developers who participated on creation of the framework and contains also description of modules that aren't mentioned in this thesis at all.

The code of the framework is also available on the attached CD-ROM or in an online SVN repository at http://subversion.assembla.com/svn/multimedia-exploration/. The repository can be publicly read, write-rights need to be requested at tomasgrosup@gmail.com. The documentation also describes how to build the code and how to publish it to a web server.

Chapter 6

Applications of multimedia exploration

6.1 Multimedia exploration framework

One of the main contributions of this work is the exploration framework itself. We have analyzed its requirements, implemented its core and developed several implementations of its components. We also developed two plug-ins for multi-model exploration as plug-ins for this framework. In the following sections, we will show the applications we have build on this framework and describe them shortly.

6.2 Find the image

Find the image is an artificial search scenario designed for testing and comparison of our exploration techniques. The task is to use our web-based exploration application to find as much images from a predetermined class as possible. This predetermined class should correspond to a search intention that cannot be easily transformed to a text-based query or to a query-by-example. The user is presented with an initial view of the Profimedia collection and has a limit of ten exploration operations to find the images. The image class is presented to the user as displayed object in the top-left corner.

Image classes for this task were selected from the ground truth of the Profimedia collection which contains 100 classes of images. We narrowed the classes only for those classes, which do not appear in the initial view in any of our exploration strategies. Thanks to this filtering, we also have an interesting sub-task included: how many clicks are spent before first image of the query class is found.

The classes that were used as query classes are displayed in figure 6.1.

Figure 6.1: Find the image query classes

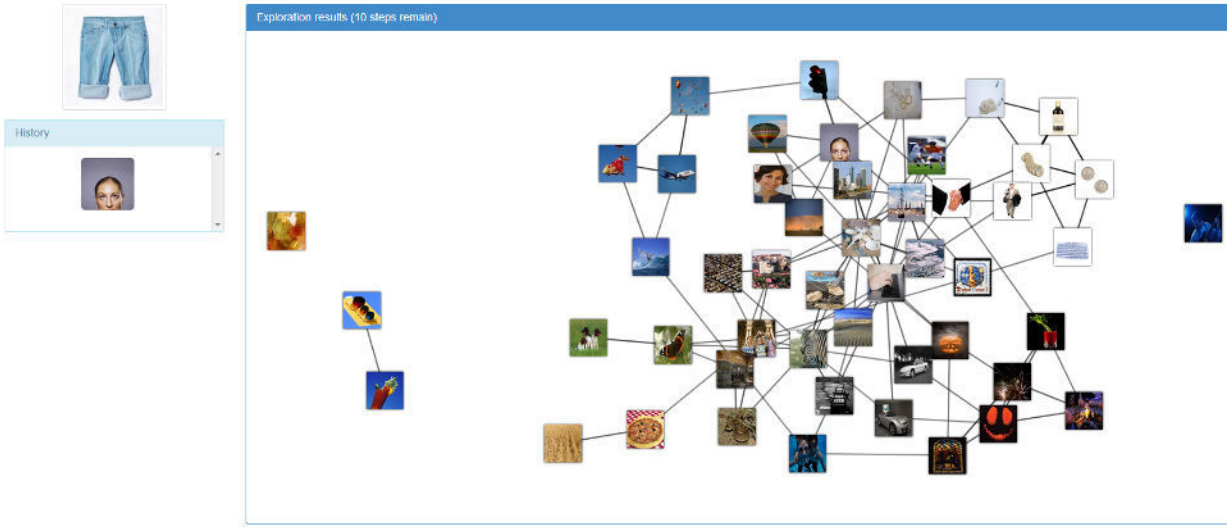


Figure 6.2: Initial view

6.2.1 Web application

The web application used for the test scenario is available online at siret.cz/MaMExploration. It is the same application we used for our comparison of two exploration strategies, iterative querying with M-Index and iterative browsing using PM-Tree [61]. We presented those results on the CBMI 2014 international conference and demonstrated the application to the participants of the conference.

At the beginning of the exploration, an exploration strategy is picked at random. It is used for one search task and it is randomly picked again with each refresh of the page. Also a query class is picked at random and presented to the user in the top-left corner of the window, as you can see in figure 6.19. The rest of the screen is covered mainly by the initial view, which was generated by the randomly picked exploration strategy and presented to the user using our layout based on the particle physics model.

User has options to move with the images via mouse dragging, zoom in or out using the mouse wheel or enlarge an image by a single-click. When user double-clicks an image, it is passed to the underlying exploration strategy as an exploration query. A new result set is generated and displayed to the user, as you can see in figure 6.3.

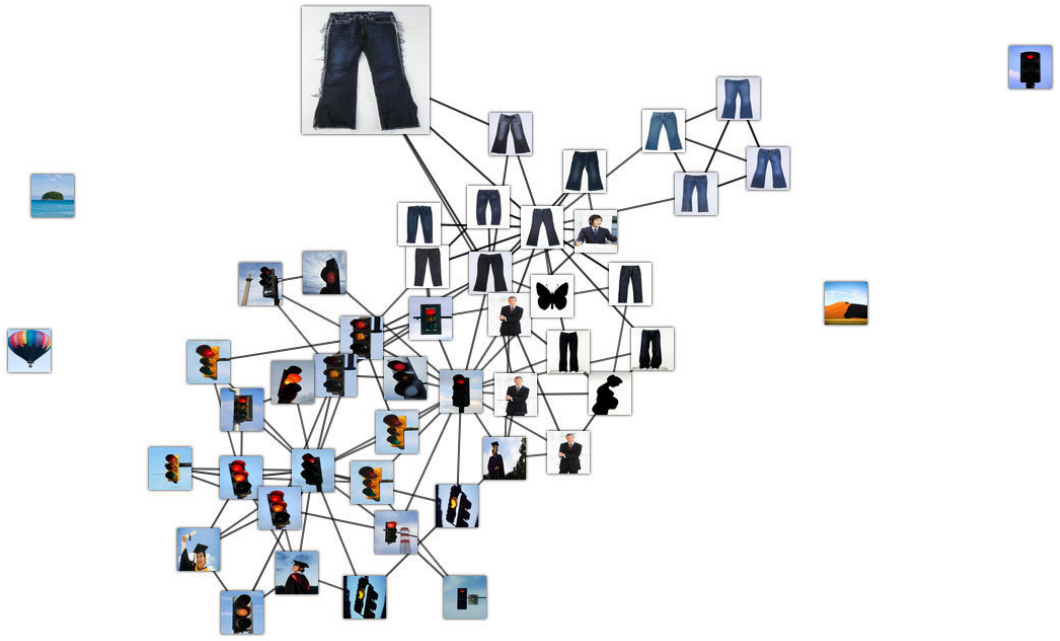


Figure 6.3: Exploration detail

Whenever user finds an image within the same class as the query image, status bar at the top of the page is updated. It displays current percentual progress, the number of found images and the total number of all images in the desired class. An information bar below informs the user how many exploration steps he has left. You can see both these bars in figure 6.4. All found images are displayed in the bottom of the page, as in the example in figure 6.5. This example also shows a very successful exploitation step, which has led to an increase of found images by more than 30. As our experimental results show, such exploitation was possible only with some exploration strategies. Other strategies, even if they have found some representatives of the query class, are not capable of finding a larger number of images in the limited number of ten operations.

User can view also the previous exploration queries he issued and he can issue them again using history displayed on the left side of the screen, where each item can be visited again by clicking on it. Visiting of the history is not counted as an exploration operation to the total number of ten clicks, and can be used to emulate breadth-first search of the explored collection. To do so, the user can always go back to the initial view and issue new query from there until he finds a satisfactory result set. An example of the exploration history is in figure 6.6

6.2.2 Experimental results

During the exploration, our server is collecting results and exploration history of each session. We analyzed the results and compared our exploration strategies using multiple

Figure 6.4: Exploration progress

Figure 6.5: Exploitation step

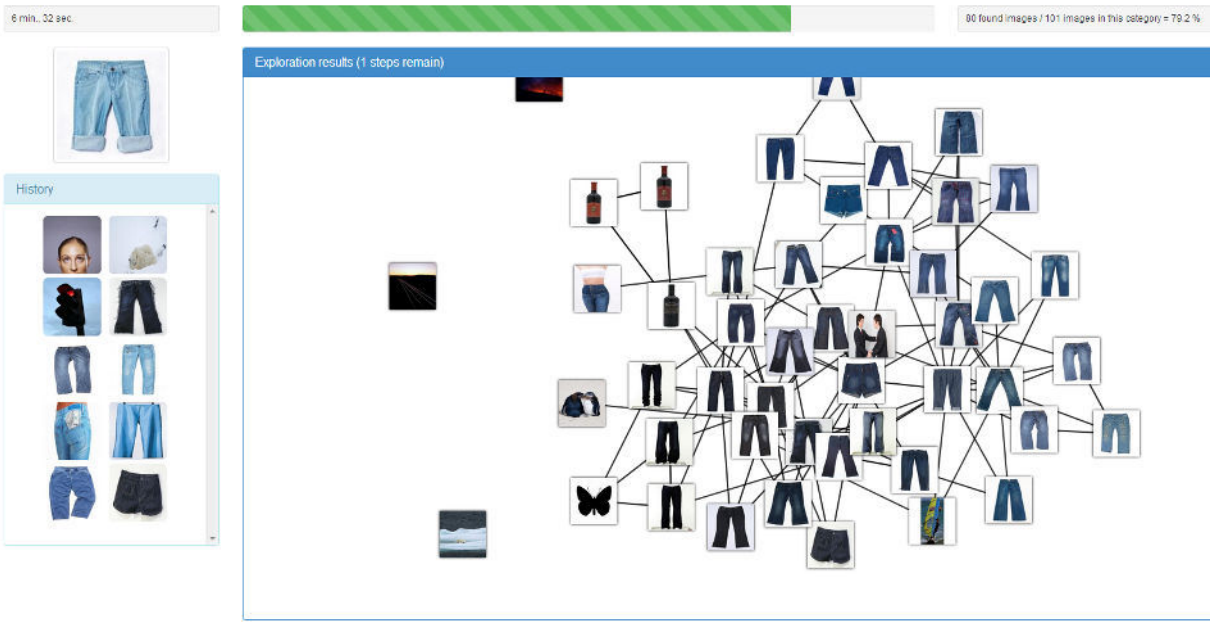


Figure 6.6: Exploration history

criteria.

This experiment and the processing of the results was done together with Přemysl Čech, who also uses the results for his master thesis. Each of us developed several exploration strategies and we compared them directly to each other. While this work focuses on the results of M-Index and multi-model approaches, his work focuses on various techniques for native exploration of tree-based structures, such as the M-Tree and PM-Tree. The graphs presented in this section are a common work and some of them are displayed both in this thesis and in the master thesis of Přemysl Čech [79].

The exploration strategies we were comparing are M-index based techniques from chapters 2.4 and 2.5 and two multi-model exploration strategies described in chapter 3. All single-model strategies were using only the $FS + SQFD$ similarity model, multi-model strategies were using both $FS + SQFD$ and $SURF + VLAD$ similarity models.

Unsuccessful search

In the first figure 6.7, we show the percentage of unsuccessful search. Unsuccessful search means that given a query class, the user was unable to find at least one image from that class during ten explorations steps. The M-Index Traversing strategy has roughly three times less percent of unsuccessful search than the other strategies we are comparing. This can be caused mainly by its construction and exploration algorithms which guarantee 100% reachability. The clusters created by the partitioning in M-Index are presented evenly to the user and the exploration continues in this approach. The multi-model approaches and M-Index k-NN all use a k-NN search underneath and had basically the same percent of

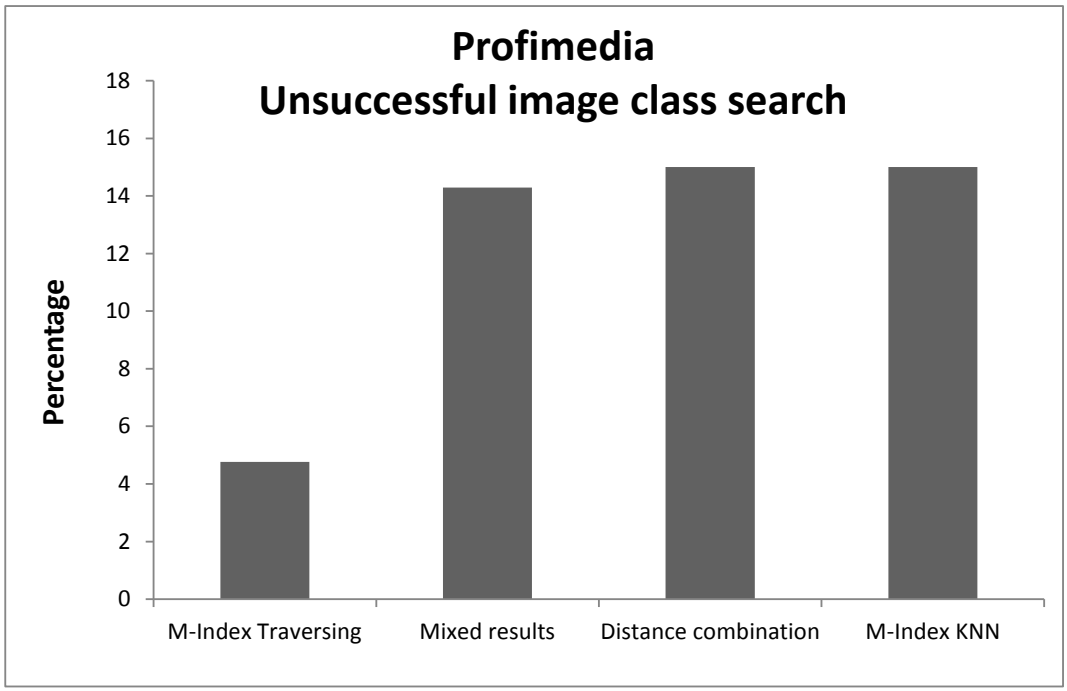


Figure 6.7: Unsuccessful search

unsuccessful search. When the query image isn't at least partially similar to any of the images presented in the initial view, users often have to guess and try more images from the initial view before they find a match.

Clicks to first found image

The figure 6.8 shows the number of clicks needed to find at least one matching image. The results are the average value for all of the successful cases. We have excluded all cases of unsuccessful search from this graph, and from some others as well. If they are excluded, the header of the graph has "Only successful search" in it.

As you can see, all strategies needed between 2 and 4 clicks to find a first match on average. An interesting difference can be found between Distance combination and Mixed results. Mixed results strategy, which displays results for queries to two similarity models at the same time, finds a matching image faster. Combination of multiple models can increase recall and help user to find an object of interest more quickly. It is also interesting to see that these results have no correlation to the percentage of unsuccessful search.

Found images

Number of found images matching the query class is the main criterion of this task. The results are displayed in figure 6.9. They are the average values of all cases. The best scores were achieved by our two multi-model approaches, Mixed results and Distance combination.

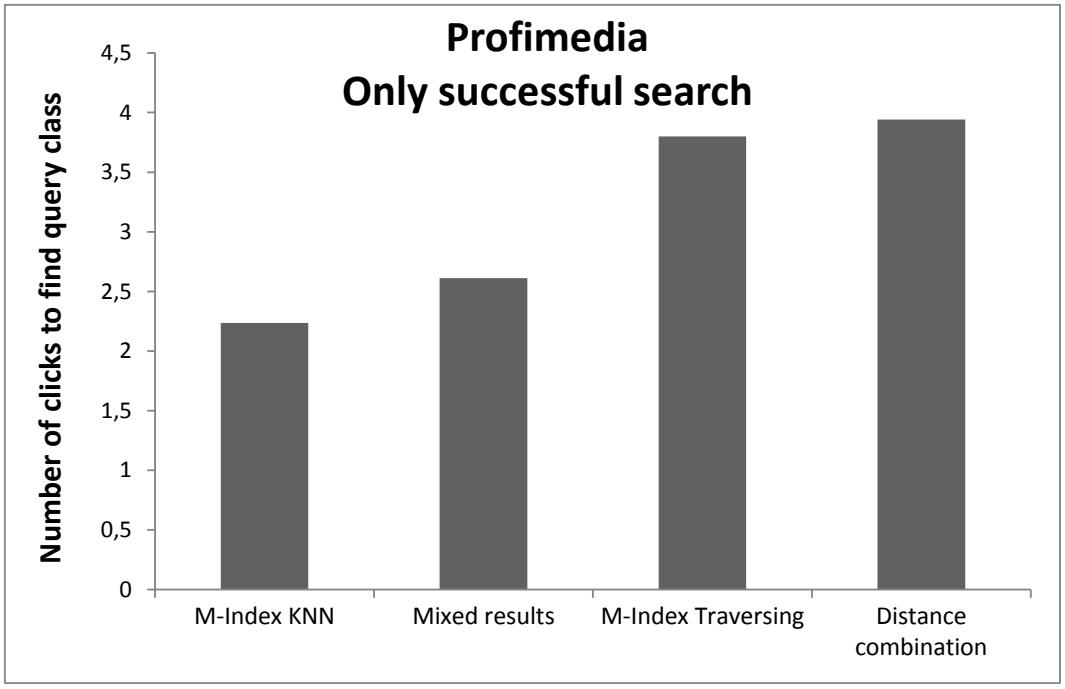


Figure 6.8: Clicks to find first image

By achieving better score than any of the single-model approaches, they validate the idea of multi-model to multimedia exploration and we will focus on it more in our future work.

M-Index Traversing achieved a very low score, even when it had one of the best results for unsuccessful search percentage. By exploring only through space partitioned by M-Index, this strategy has no option of exploitation and retrieving more results if user already finds a match.

Figure 6.10 shows also the percentage of found images matching the query class, but divided into individual exploration steps. As we can see, the results correspond to the results in figure 6.9 with the exception of the Distance combination strategy. We believe that the rapid gain after 5th click comes from the utilized relevance feedback technique. Once the user finds a first few objects, he starts on using them as exploration queries an the combined similarity model adapts exactly to the query class he is trying to find. Thanks to this approach, Distance combination strategy achieves better results than Mixed results strategy, which had the best results up to the 6th click. Mixing of results shows a better recall and stably increases the number of found images with each click. M-Index traversing strategy could not get any more results after first 5 clicks and is clearly missing an exploitation step that would help it to retrieve more results similar to the objects user is interested in.

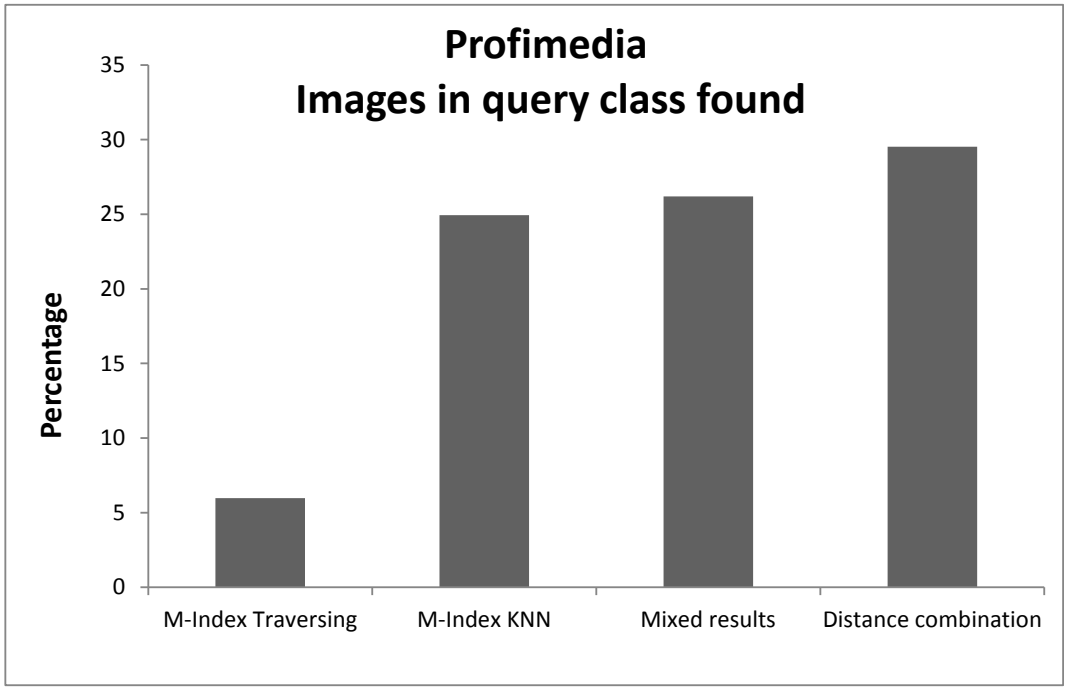


Figure 6.9: Found images

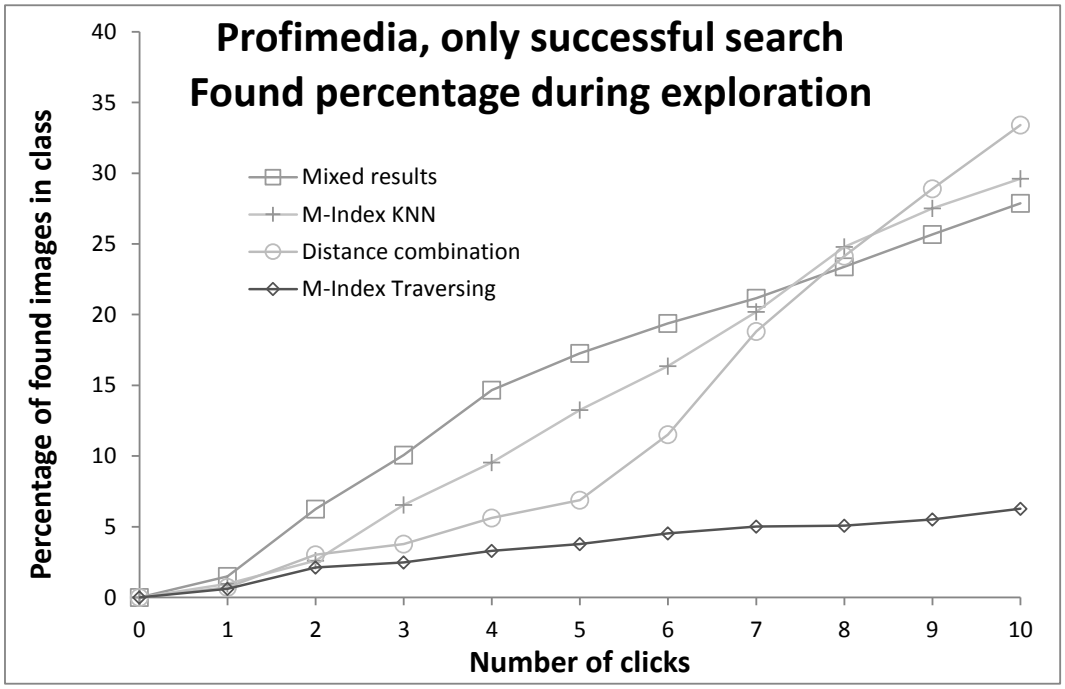


Figure 6.10: Found images per click

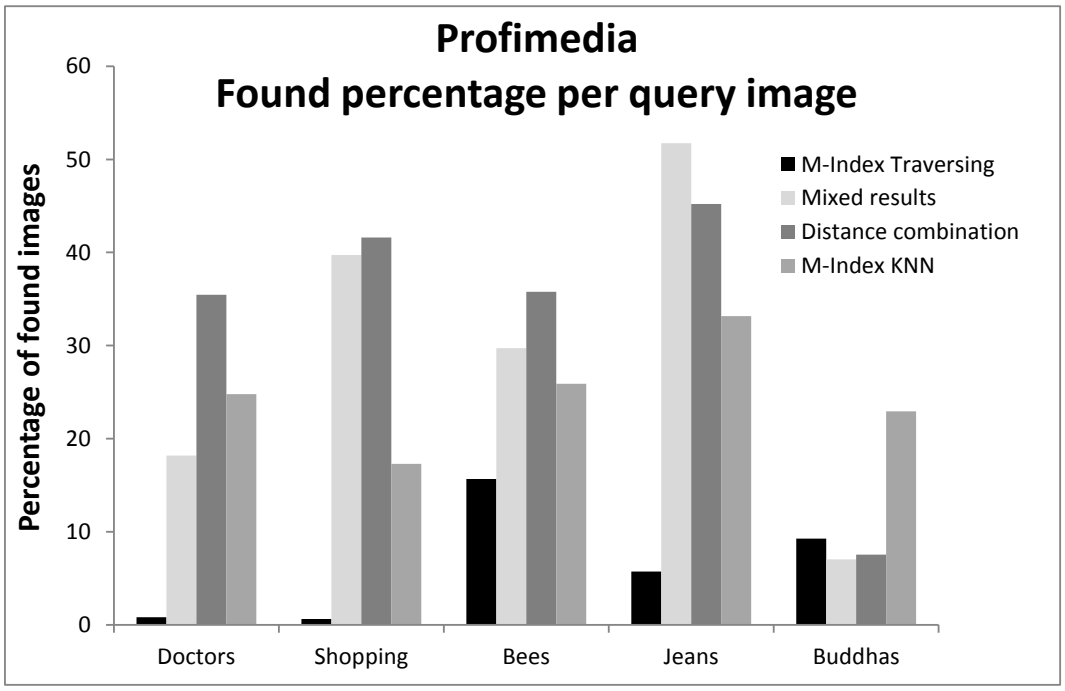


Figure 6.11: Number of found images per query

Differences between queries

We also observed the results for individual query classes we presented in figure 6.1. The results for individual query classes in figure 6.11 show how much is an exploration strategy affected by the query class. The poorest performance was observed for images of Buddha. It is a hard class to determine using content-based retrieval, since the Profimedia collection also contains images of sphinx, pyramids, deserts and other objects exhibiting very similar color distribution and sharing common keypoints.

Except for Buddhas, multi-model approaches had the best score in all query classes. M-Index $k-$NN achieved very good results in all classes except for Shopping. Shopping contains various images from a supermarket, with different people and different products. Such images are hard to match using only the $FS + SQFD$ similarity model, since their color distribution can vary a lot.

Images viewed during exploration

Secondary metrics we observed were the numbers of viewed images and classes during exploration, not considering what the current query class was. As you can see in figure 6.12, the average number of viewed images varied between 200 and 400 images per exploration session.With the initial view and ten exploration operations, a theoretical maximum is 550 images per session. The lowest number of view images was achieved by exploration strategies based on traversing an index structure. The users were presented with a lot of

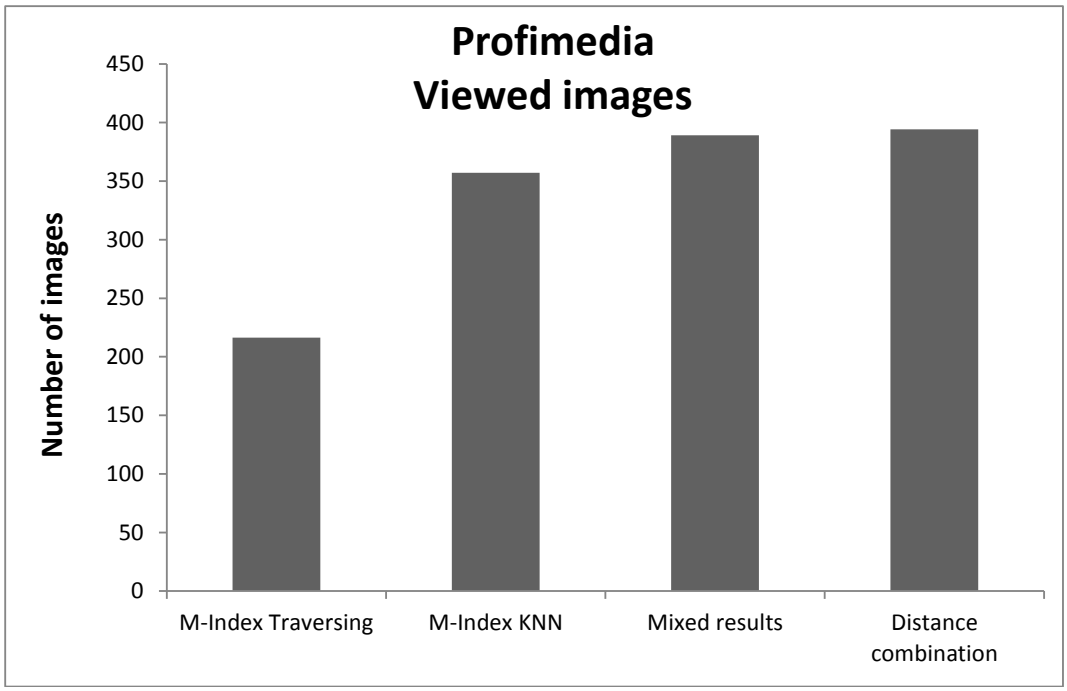


Figure 6.12: Number of images viewed

duplicate images and may have get stuck in a small part of the index. The best results were achieved by the multi-model approaches. Their usage of relevance feedback also minimizes the changes of a duplicate result set presented to the user, since the query parameters are changing with each exploration step.

Number of viewed classes presented in figure 6.13 is displayed separately for each exploration step. Multi-model strategies, which use a random initial view, had on average nine more classes in the initial view than the other strategies which are relying on capabilities of an index structure to cluster objects and display the relevant representatives.

Increasing number of viewed classes after the first matching image is found basically means presenting non-relevant objects to the user. It was happening to every exploration strategy in a small amount. If those non-relevant objects take only a small portion of the screen, it shouldn't be harmful to the exploration process. New classes will give more exploration options to the user, which may be beneficial in some cases.

6.2.3 Comparison of all strategies

In this section, we directly compare results of proposed strategies together with the strategies from the thesis of Přemysl Čech.

As you can see in figures 6.14 and 6.15, multi-model approaches were achieving the best results together with the M-Index $k-$NN approach. M-Index $k-$NN had the best score of all single-model approaches. It was also a lot better than similar approaches using PM-Tree and $k-$NN search, which may be affected by the approximate search used in the

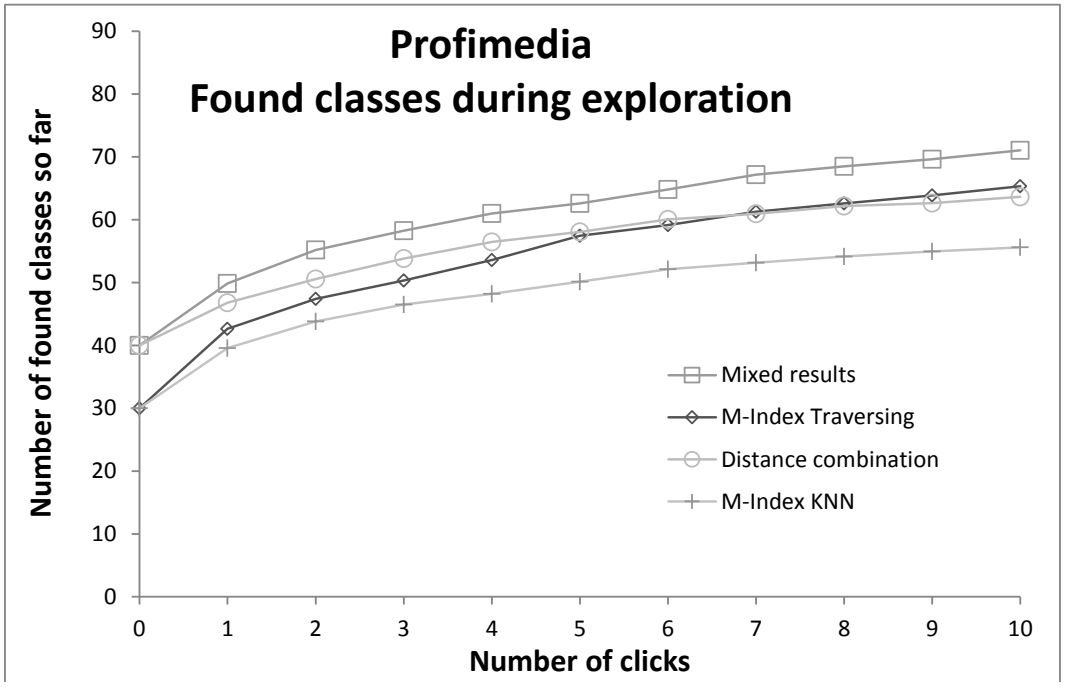


Figure 6.13: Number of classes viewed

M-Index $k-$NN strategy. By approximating the search, we can gain more variable results that can be beneficial to the exploration process.

We also think that exploration approaches utilizing a PM-Tree were handicapped by our selection of query classes. The Pm-Tree should perform better for the classes contained in the root of the tree. However, we filtered away all the classes appearing in the initial view and thus we did not run our experiments on any of the classes contained in the root. We plan to extend our experiments to support this opinion in the future.

6.2.4 Discussion

We have observed that the multi-model approach to image exploration increases the effectiveness of the exploration process. The utilization of relevance feedback techniques boosts the exploration process as well, especially in the exploitation phase. We will further analyze different relevance feedback techniques and implement them in our exploration strategies.

In the future, we plan to focus on efficiency of multi-model search. To do so, we will investigate different indexing options and native exploration of the corresponding indexing structures.

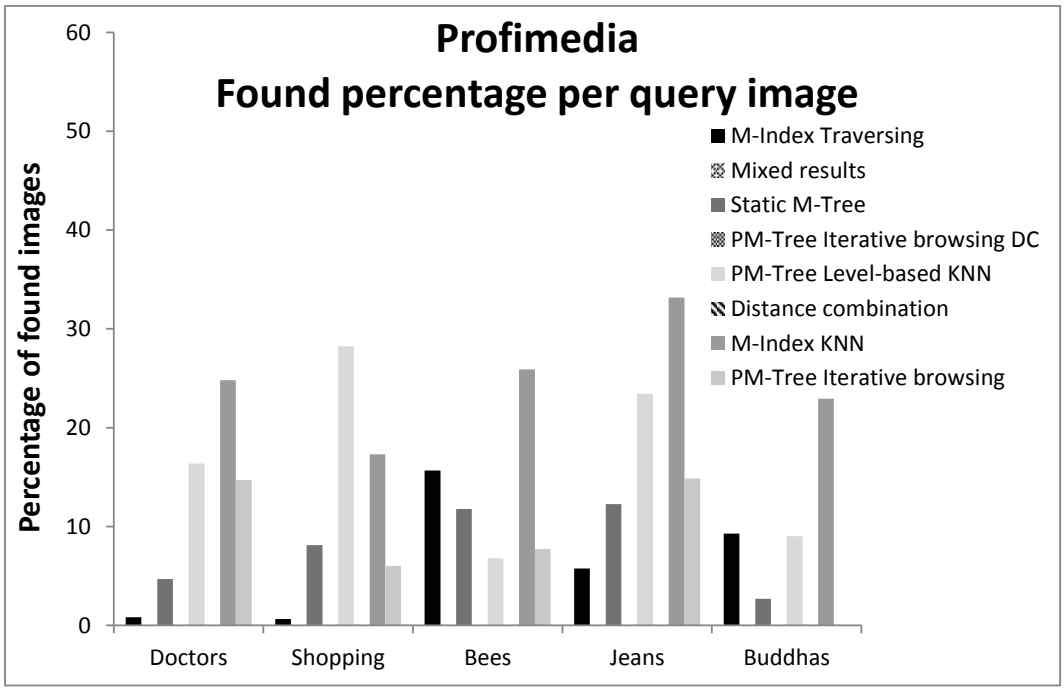


Figure 6.14: Found images per query for all strategies

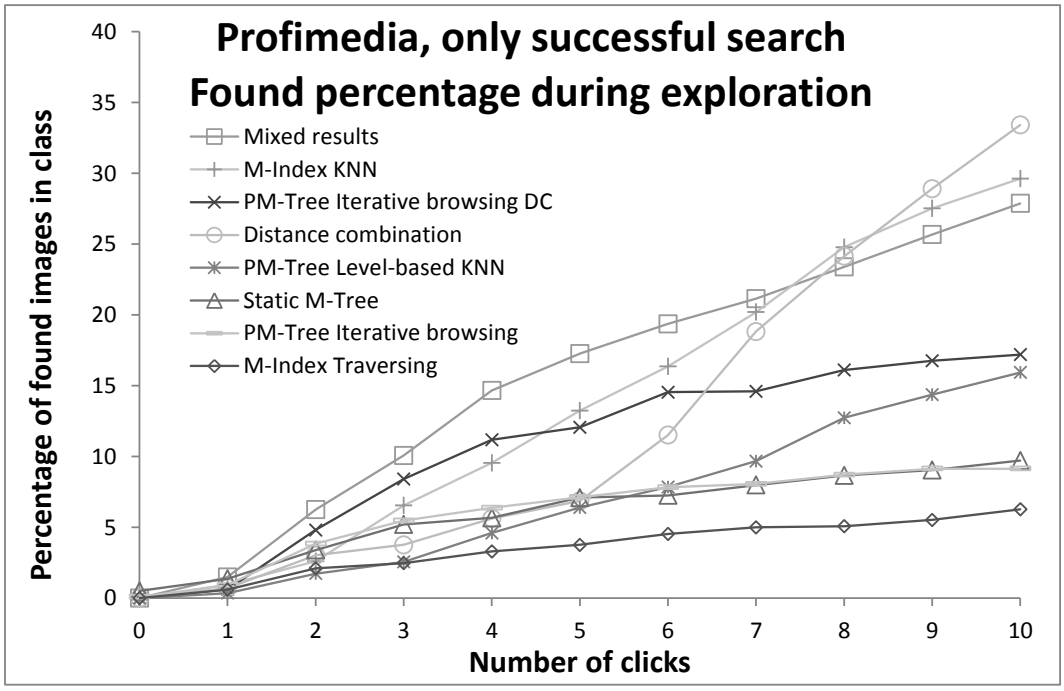


Figure 6.15: Found images for all strategies

6.3 Image exploration portal

Exploration portal is an application that demonstrates most of the features implemented in our multimedia exploration framework. It is an online exploration system in which users can explore images from three different image sources: Bing, Facebook and Profimedia.

The application has a modern user interface that leaves most of the screen for our similarity based layout presenting the results. The layout is based in the particle physics model and supports several operations, such as exploration via an image, multi-queries or moving in different directions.

The exploration can be driven by any of the similarity models implemented in our framework. To change the configuration of the backend for upcoming requests, a variety of options can be changed in a settings window. Those options can be also changed during exploration, which brings new options to comparison of various techniques.

The portal is available online at http://herkules.ms.mff.cuni.cz/. User's guide describing the controls and options of the portal can be found online in a public Google document linked from the homepage of the portal or on the attached CD-ROM. This guide also contains a lot of screen-shots from the portal which show the exploration process in action.

Portal's functions can also be accessed via a web API, which is described in generated API documentation.

6.4 Similarity analytics

6.4.1 Motivation

Our multimedia exploration framework is not designed just for images or videos and can be applied to a broader range of data. With our model, any kind of data that can be visualized and compared to each other using a distance function can be used for multimedia exploration. We began to develop a tool called similarity analytics, which should help experts from various domains to explore their data and get a better high-level understanding of it. It should also help when modeling a distance function or when tuning its parameters, by directly visualizing impact of the change to the distance function. In addition to that, the tool should be able to quickly propose a distance function that learns from annotation of the data, and let the domain expert manually tune eventual details. By strong focus on visualization, users should quickly see whether the current distance function corresponds to their understanding of relations between the objects or not.

This tool is meant to replace current manual and often hard to use methods, in which experts are only running measurements against data collections with some kind of ground truth and reading the results in the form of a text file containing different values, such as mean average precision, recall and other metrics. We believe that visualization and exploration techniques could not only improve the process of modeling a distance, but also can be used for some other tasks, such as correcting the ground truth or observing completely new patterns in the data.

6.4.2 Network events

We began to develop this tool for a research division of the CISCO company. The research team is developing a technology that can monitor behavior of the entire computer network at real-time and detect potential security threads. To do so, they utilize behavior analysis and custom machine learning and artificial intelligence algorithms. By monitoring the network, they are also learning how the network works and learn suspicious activities from that.

Research of this technology first started as a research project at Czech Technical University (ČVUT) and a company called Cognitive Security was established as a result of this research. In February 2013, CISCO announced the acquisition of Cognitive Security and integrated it as a part of their security division.

With our tool, we are trying to help the research team with modeling a distance function used for classification of network events. Each event contains information that was captured during monitoring of the network. An event consists of one or many flows of data between two network nodes in a limited time-frame.

By classifying the event, the system can decide whether it is a known network traffic, known cyber-thread or an unknown type of network traffic.

Examples of known classifications can be seen in figure 6.17, details for an event that are displayed in our tool are shown in figure 6.16.

6.4.3 Distance between events

For distance computation, each event is converted to several histograms. This conversion was already done by domain experts from CISCO and our tool receives the data already with the computed histograms. Each histogram corresponds to a property computed from the event's flows, such as number of bytes sent, length of the domain name or elapsed time between request and response. The property is transformed to a number for each flow and assigned to a bin in the histogram.

To compute distance between two events, a linear combination of L_1 metrics over the individual histograms is used right now. We plan to propose more distances over histograms in the future. What is unknown are the weights of the linear combination used to form the final distance. By default, each histogram is assigned the same weight 1.0 for computation of the distance.

6.4.4 Event exploration

The tool should help to find a better linear combination than the default setting. The users start by selecting a combination of data files they want to explore, as you can see in figure 6.18. Any subset of data files can be used together, for example a mix of annotated and unannotated data. The user can also upload a new data file, which has the format of a .zip archive containing .json files describing the events.

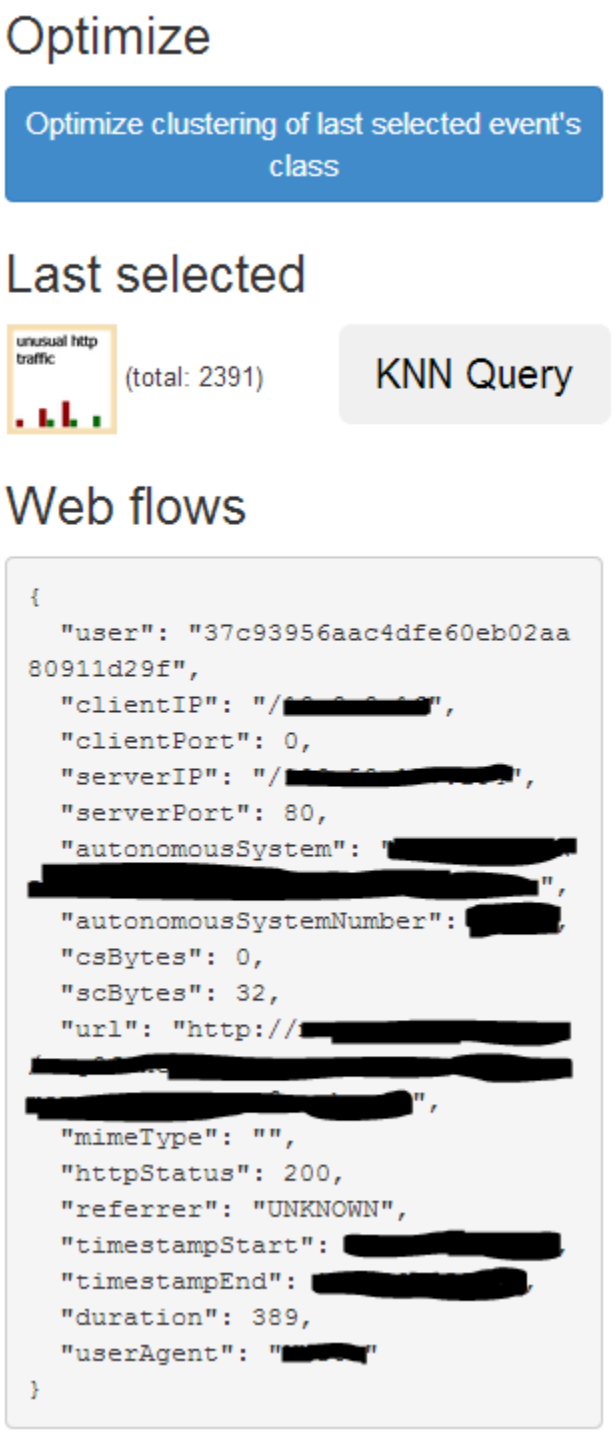


Figure 6.16: Event details

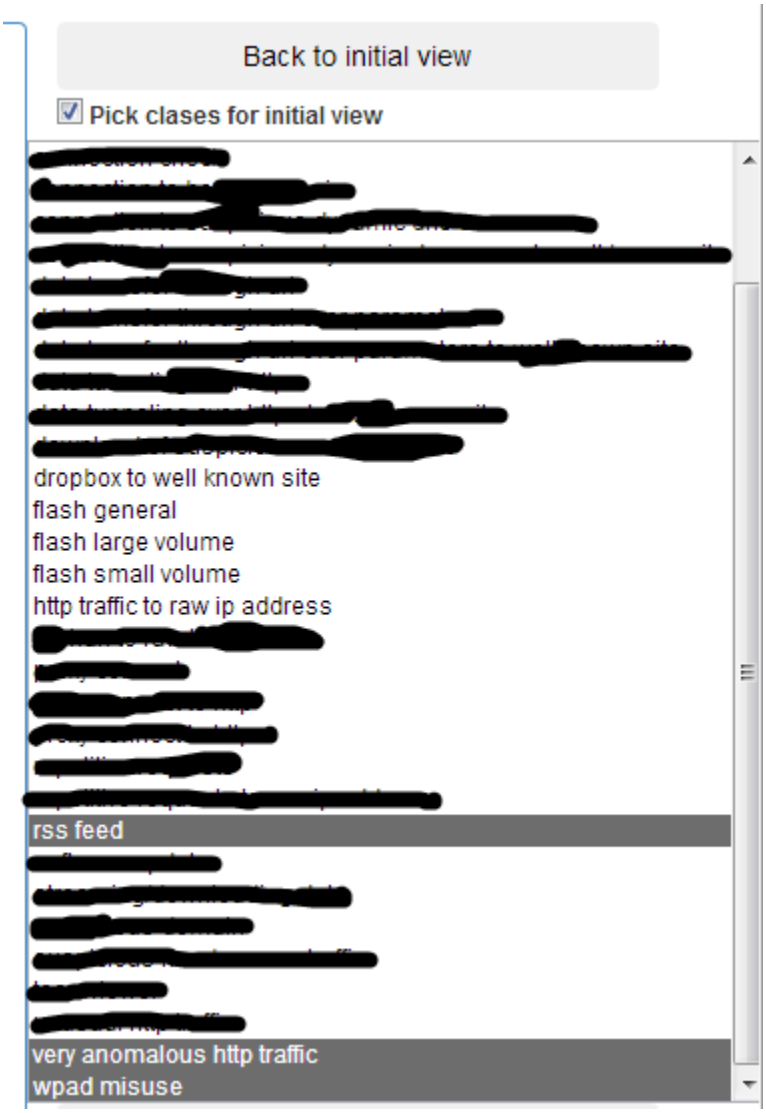


Figure 6.17: Filtering classes for initial view

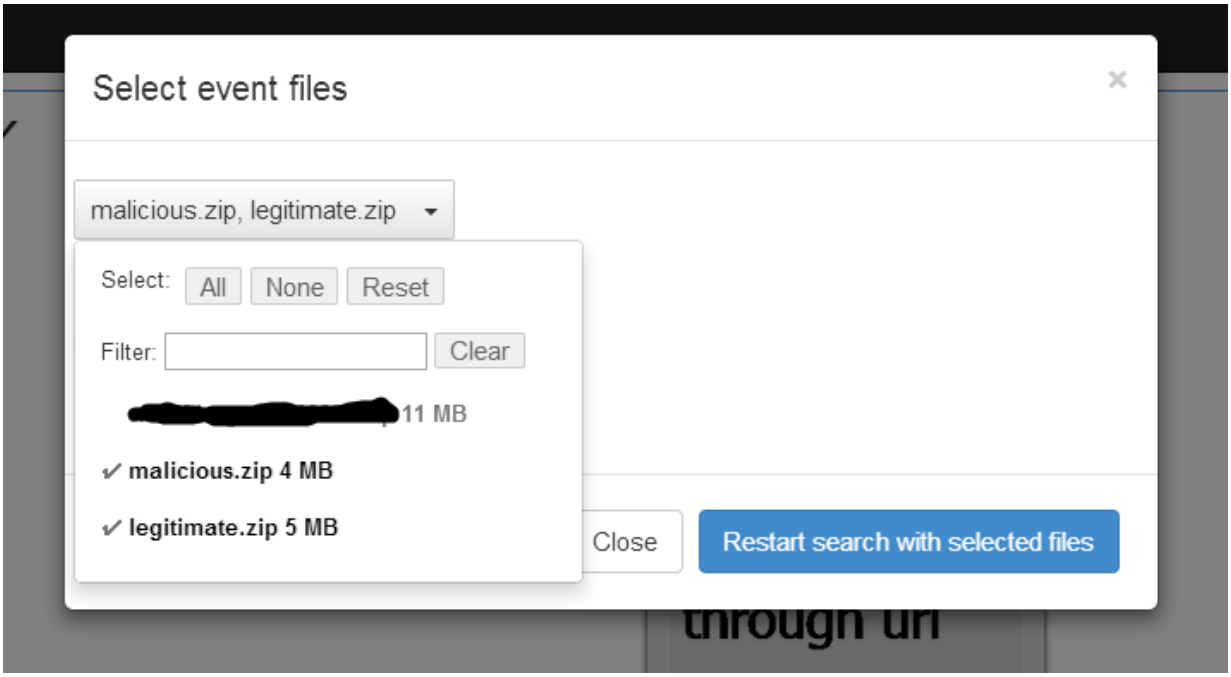


Figure 6.18: Custom data exploration

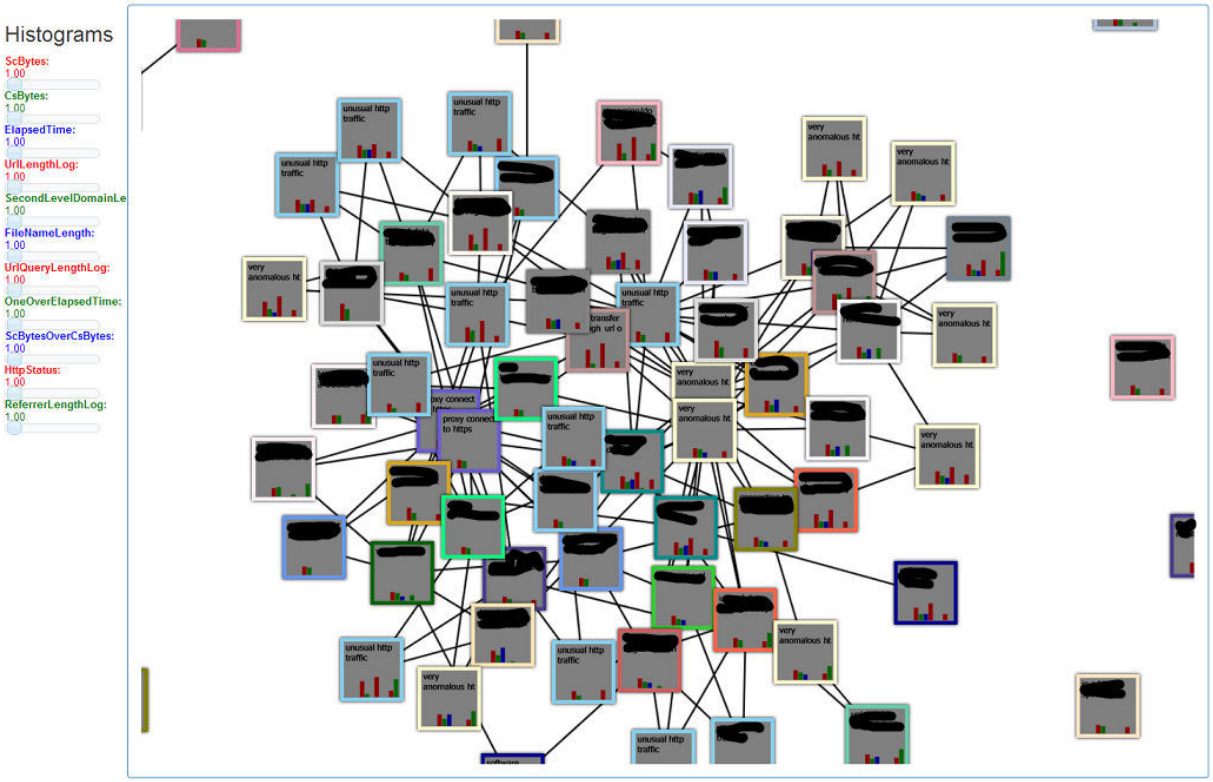


Figure 6.19: Initial view

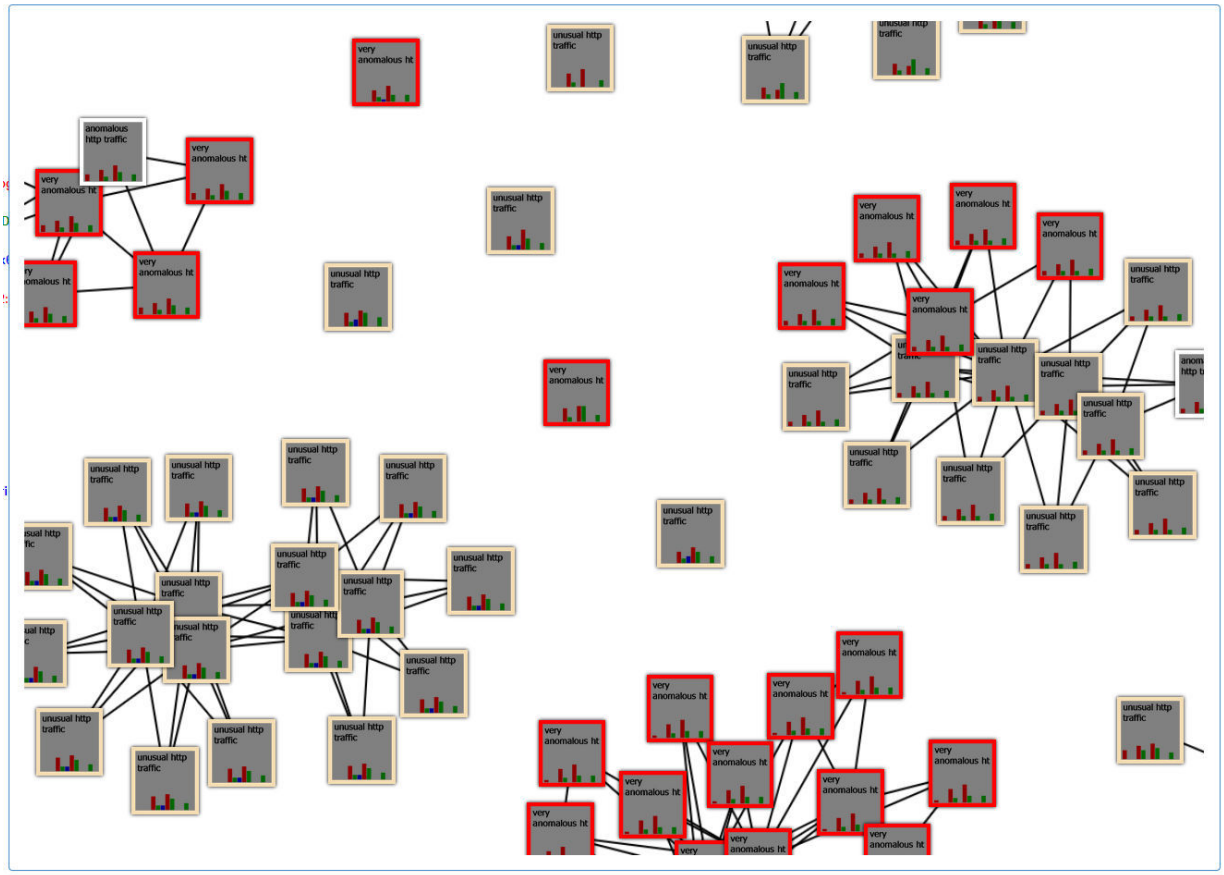


Figure 6.20: Exploration result

After that, an initial view containing at least one event for each classification and a random subset of the database is presented in a layout based on the particle physics model. You can see an example of the initial view in figure 6.19. On the left side of the screen, individual histograms are shown to the user together with current weights for the linear combination. As you can see, it begins with every value set to 1.0. The center of the screen is used for exploration of the events.

To explore the events, double-click on an event is used. This triggers a $k-$NN query in the entire dataset and applies the current weights for the linear combination of histogram distances. Results of the exploration step are shown in figure 6.20.

6.4.5 Visualization

Each event is visualized by a border representing the event's classification, a textual description of that classification and a small bitmap giving a high-level view of the data used for distance computation. As you can see in figure 6.21, the bitmap has the form of a small histogram. This can be enlarged by a mouse-click. Each bin corresponds to a single

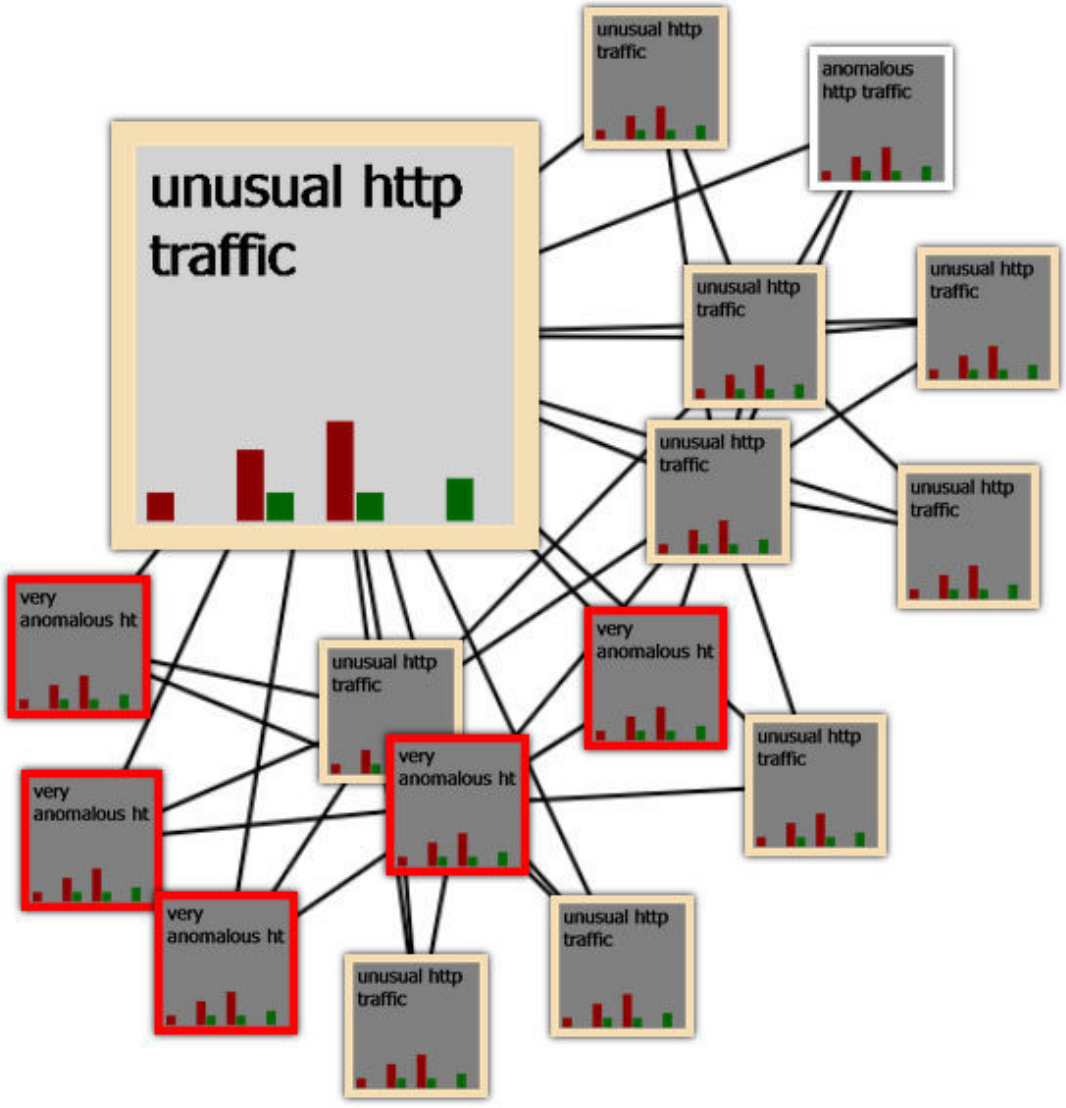


Figure 6.21: Histogram visualization

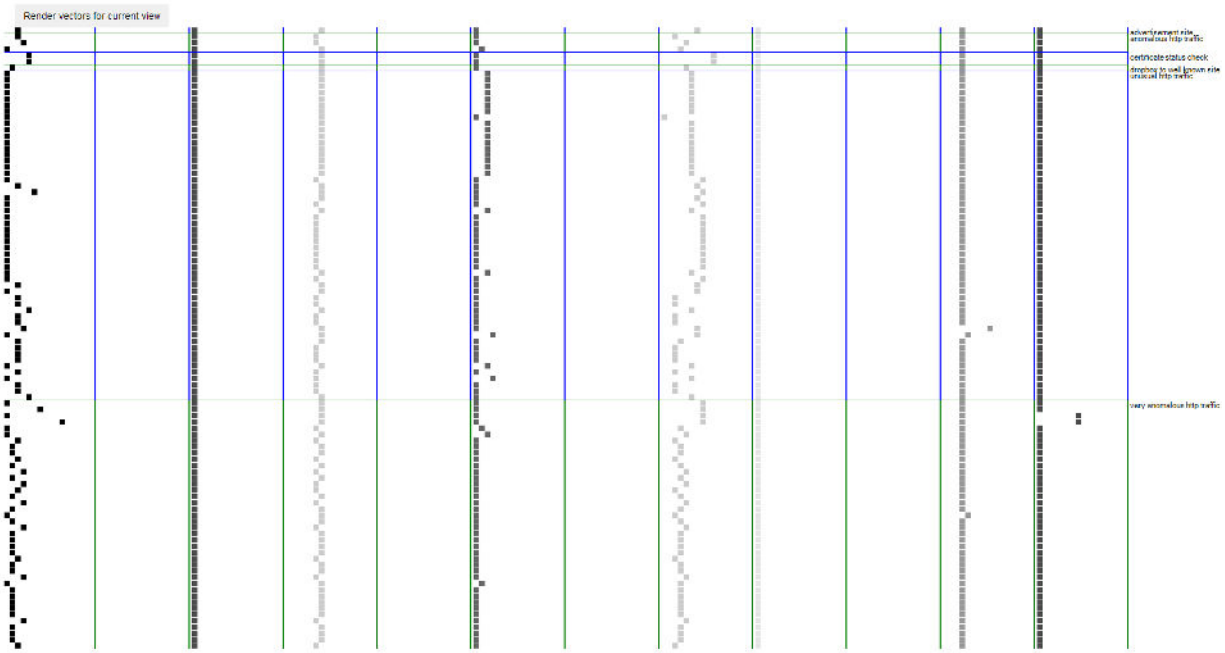


Figure 6.22: Vectorized data overview

histogram from the event's data. The height of the bin is determined by the largest value within that histogram. Bins are ordered in the same way as the histograms on the left side of the screen and such representation gives a basic understanding of why two events are or aren't connected with an edge, meaning the distance between them is or isn't below a threshold value. You can see that connected events have similar representation of the histogram. You can also see that events with same classification are connected together, but events with other classification are mixed with them as well.

For a detail view over the data used for distance computation, vectors of histogram values can be rendered for all of the objects presented in the current view. Each event is represented by a row of values, where each column represents a single bin within a histogram. Individual histograms are in the same order as presented on the left side of the screen and are divided by vertical lines. Horizontal lines are dividing individual classifications, which are written on the right side of the rendered image. Cells of the image correspond to bin values of the histogram, taking the linear combination for distance combination into account by multiplying the value. Resulting values are normalized into the range from 0 to 255 and each value is assigned a different drawing. Values of 0 stay white and as you can see, this is the vast majority of values in the image. This means that the histograms are very sparse and usually contain just a single value. Small values between 1 and 7 are represented by a red circle with lightness decreasing linearly from 1 to 7 . Rest of the values is represented by a gray square with different shades of gray. Lightness of the square also decreases linearly from 8 to 255. You can see the rendered bitmap in figure 6.22 and a zoomed-in view in figure 6.23, which also shows the classifications visible

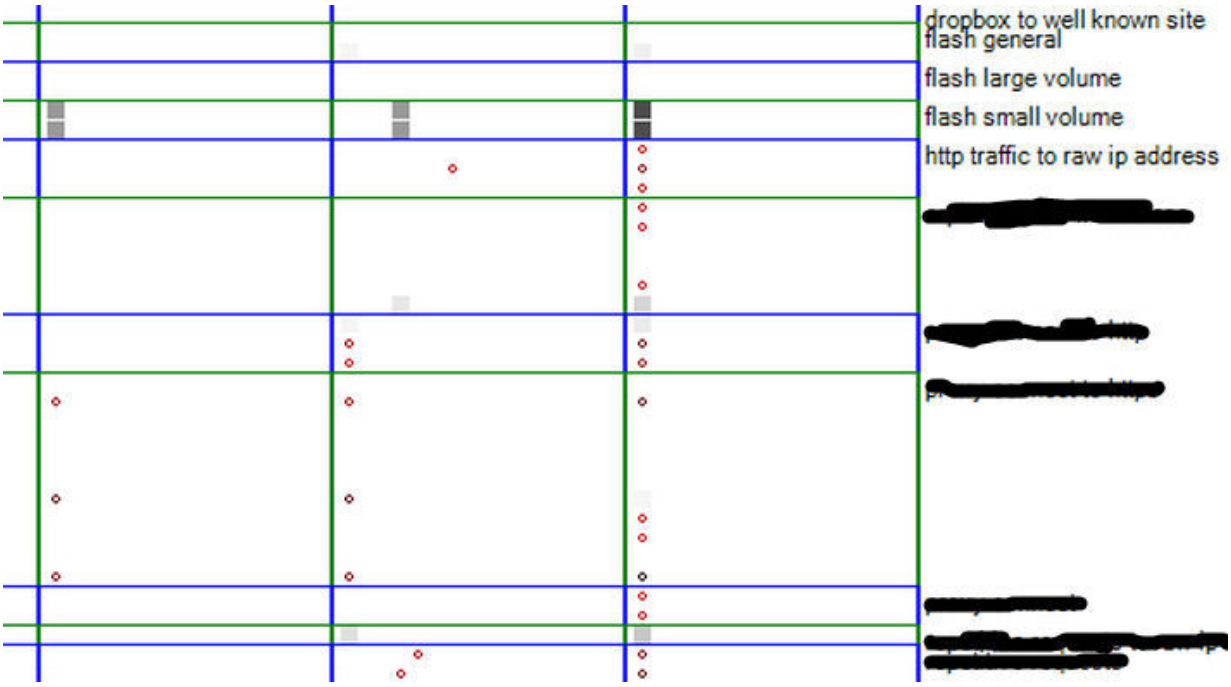


Figure 6.23: Small differences in vectorized data

on the right side of the bitmap.

6.4.6 Distance tuning

If the user is not satisfied with the results of a $k-$NN query or with the clustering of the events on the screen, he can manually change the weights used for the linear combination of histogram distances. Each histogram has a slider which allows to set a weight between 0 and ten to adjust the linear combination. After the weights are changed, the screen is automatically updated with the new distance function. The user can observe how change of the weights affects clustering of events. He can also continue with exploration and new weights are used to find nearest neighbors in the entire dataset. The response is shown withing a second and user can quickly evaluate impact of his change.

It is also possible to go back to initial view while keeping the current weights. This enables to see how those weights affect the clustering of all types of classifications at once. To evaluate the weights only on a small subset of the database, an option to pick classifications for the initial view exists. To do so, user has to select the desired classifications in the top right section of the window, as can be seen in figure 6.17. After going back to the initial view, only events with one of the selected classifications are present. This can be used to see if two classifications share events that are similar or not. An example of a view filtered to contain only classifications “rss feed”, “wpad misuse” and “very anomalous http traffic” is in figure 6.24.

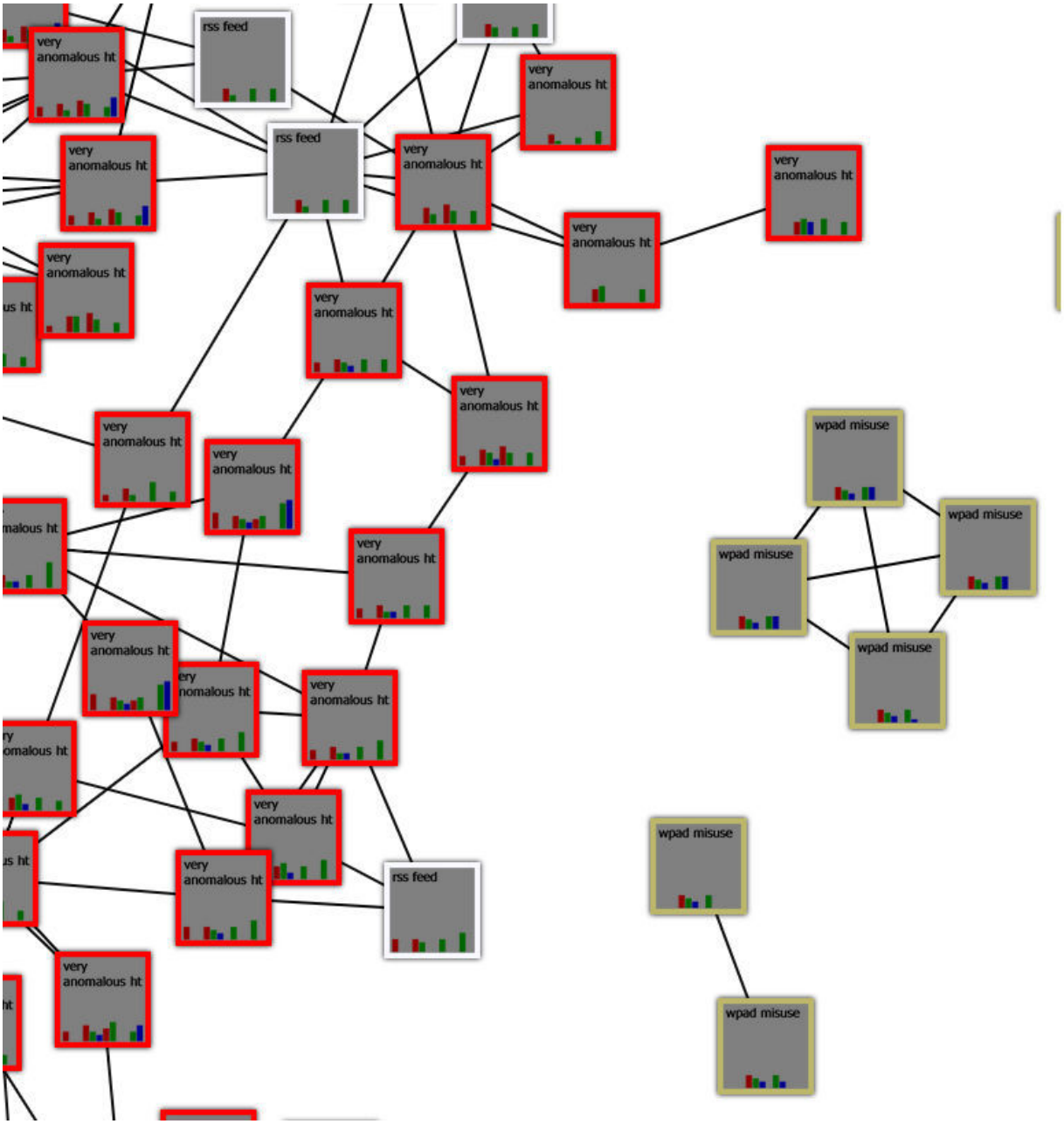


Figure 6.24: Filtered initial view

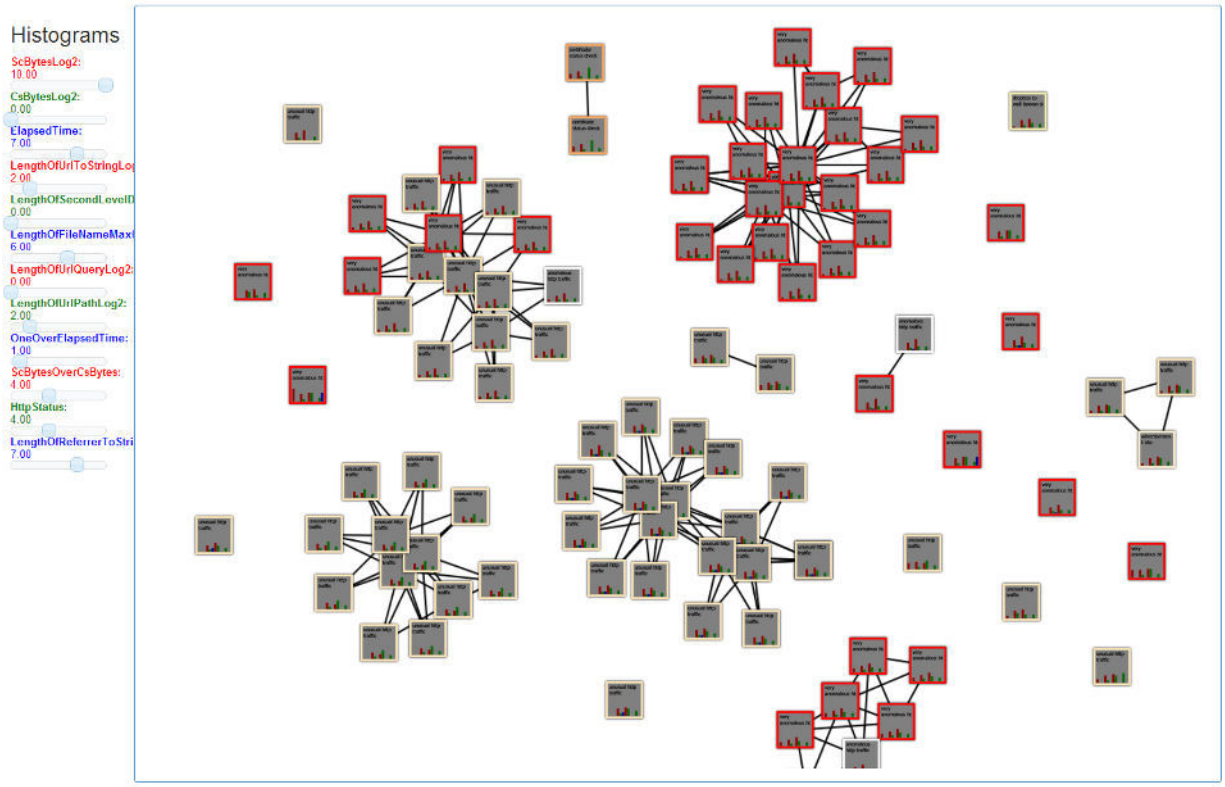


Figure 6.25: Automatic optimization

6.4.7 Automatic optimization

To simplify the process of finding a good combination of weights manually, we added an option to optimize the weights automatically. Given a result set and a selected event, the button "Optimize clustering of last selected event's class" can be pressed. It tries to find such configuration that gives the best clustering of the classification selected by the user. To do so, the application randomly generates weights configurations and creates similarity graph for the currently visible events. To pick the best configuration, a fitness function to evaluate the configuration is passed in. The current implementation uses a function that counts edges between events of the selected classification and subtracts count of edges going from an event of the selected classification to an event of a different one.

This operation gives quickly a sketch of the weights, which can be manually refined later. A result of this operation can be seen in figure 6.25. The algorithm tried to optimize the clustering for all events with the classification "unusual http traffic" (yellow background). Several clusters have formed and some of them contain only a single classification. The clusters can be reviewed by the domain expert and potentially be used to describe a new type of behavior. As figure 6.26 shows, the algorithm tried to generate a configuration in 192 iterations and the optimization took 4.002 ms.

Graph creation	20
Iterations	192
KNN query	21
Total time	4,002

Figure 6.26: Optimization information

6.4.8 Discussion

We have developed the similarity analytics tool and used it for exploration of network events. In our future work, we plan to focus more on this tool and improve the tool and its user interface intuitiveness. We would like to use it on bigger data collections and add new features that would enable the experts to annotate unknown events and to create new classifications. We would also like to use the GUI to correct classifications for a cluster of events by propagating the classification to all connected events. The application would then allow to download the corrected data.

Big data

The tool uses sequential scan over the entire dataset to find nearest neighbors to an event. It also holds all the data in memory. If the data gets too big, a different approach will be needed. Traditional MAMs cannot be used, since the tool allows to change the resulting distance during the process. Additions to MAMs that allow to index dynamic distances are an option, but their efficiency quickly drops when more than 3 models are combined together [63]. Currently, we are combining 11 distances and utilization of such index wouldn't be very big. If needed, we would like to come up with a new approach for dynamic distance, possibly based on approximate search with M-Index.

Conclusion

We have investigated multi-model approach to multimedia exploration and implemented two different strategies. We have also developed two strategies for M-Index exploration and compared them with our multi-model approaches in a user study called Find the image. All strategies were built in a new multimedia exploration framework we have designed and implemented.

Our comparative work focusing on approximative search using M-Index was published and presented at international conference SISAP 2013. Results of the user study Find the image and the demo application itself were presented at international conference CBMI 2014.

Our main contribution lies also in the new exploration framework we have developed. In this work, we have analyzed its requirements, described its implementation and deployed several applications using it. We have applied it not only to exploration of images, but also to exploration of network events in a tool called similarity analytics.

Future work

In the future, we plan to focus mainly on the similarity analytics tool and continue in the started cooperation with the CISCO company. We would also like to improve our multi-model exploration strategies and utilize indexing methods to make them available for large-scale exploration.

Attachments

The following documents can be found on the attached CD-ROM:

- PDF version of this thesis
- Source code of the exploration framework containing multi-model plug-ins, find the image application and similarity analytics tool
- Programmer's guide to the exploration framework
- User's guide to the exploration portal
- Files used for evaluation of find the image results

Nomenclature

BoVW	Bag of visual words
GUI	Graphical user interface
IoC	Inversion of Control
k-NN	k nearest neighbors
MAMs	Metric access methods
MAP	Mean average precision
SIFT	Scale invariant feature transform
SOL	Siret Object Library
SQFD	Signature quadratic form distance
SURF	Speeded up robust features
VLAD	Vector of locally aggregated descriptors

Bibliography

[1] Gantz, J.: The Diverse and Exploding Digital Universe, 2008

[2] http://blog.1000memories.com/94-number-of-photos-ever-taken-digital-and-analog-in-shoebox

[3] http://news.yahoo.com/number-photos-taken-2014-approach-1-trillion-thanks-013002154.html

[4] http://www.businessinsider.com/smartphone-and-tablet-penetration-2013-10

[5] Facebook, Ericsson and Qualcomm: A Focus on Efficiency, 2013

[6] https://www.google.com/advanced_image_search

[7] Deb, J. Multimedia Systems and Content-Based Image Retrieval. Information Science Publ.,2004

[8] Blanken, H. M., de Vries, A. P., Blok, H. E., and Feng, L. : Multimedia Retrieval. Springer,2007

[9] Datta, R., Joshi, D., Li, J., and Wang, J. Z.. Image Retrieval: Ideas, Influences, and Trends of the New Age. ACM Computing Surveys 40, 2, 1–60, 2008

[10] P. Zezula, G. Amato, V. Dohnal, and M. Batko. Similarity Search: The Metric Space Approach. Springer-Verlag New York, Inc., 2005.

[11] Jakub Lokoč. Tree-based Indexing Methods for Similarity Search in Metric and Non-metric Spaces, Doctoral Thesis 2010

[12] M. Lux. Content Based Image Retrieval with LIRE, ACM Multimedia 2011

[13] Sikora, T. : The MPEG-7 visual standard for content description-an overview. IEEE Transactions on Circuits and Systems for Video Technology 2001

[14] Batko, M.,Kohoutkova, P, Zezula, P.:Combining metric features in large collections, SISAP 2008

[15] Lowe, David G. Object recognition from local scale-invariant features, ICCV 1999

[16] Gabriella Csurka , Christopher R. Dance , Lixin Fan , Jutta Willamowski , Cédric Bray: Visual categorization with bags of keypoints, ECCV 2004

[17] R. Arandjelovic and A. Zisserman. All about vlad, CVPR 2013

[18] Jégou, H., Perronnin, F., Douze, M., Sánchez, J., Pérez, P., & Schmid, C. (2012). Aggregating local image descriptors into compact codes. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 34(9), 1704-1716

[19] Jian Wu , Zhiming Cui , Victor S. Sheng , Pengpeng Zhao , Dongliang Su , Shengrong Gong: A Comparative Study of SIFT and its Variants, MEASUREMENT SCIENCE REVIEW 2013

[20] Christian Beecks, Anca Maria Ivanescu, Steffen Kirchhoff, Thomas Seidl: Modeling multimedia contents through probabilistic feature signatures, ACM Multimedia 2011

[21] M. Kruliš, J. Lokoč, Ch. Beecks, T. Skopal, T. Seidl. Processing Signature Quadratic Form Distance on Many-Core GPU Architecture, CIKM 2011

[22] Margulis, Dan. Photoshop Lab Color: The Canyon Conundrum and Other Adventures in the Most Powerful Colorspace, 2006.

[23] Christian Beecks : Large-Scale Multimedia Exploration with Adaptive Similarity Measures, http://campustv.uni-klu.ac.at/?q=node/259

[24] Christian Beecks : Distance-based Similarity Models for Content-based Multimedia Retrieval , Doctoral Thesis, RWTH Aachen University, 2013

[25] C. Beecks, J. Lokoč, T. Seidl, and T. Skopal. Indexing the signature quadratic form distance for efficient content-based multimedia retrieval, ICMR 2011

[26] J. Lokoč, M. Hetland, T. Skopal, and C. Beecks. Ptolemaic indexing of the signature quadratic form distance, SISAP 2011

[27] T. Skopal, P. Moravec: Modified LSI model for efficient search by metric access methods, ECIR 2005

[28] C. D. Manning, P. Raghavan, and H. Schutze. Introduction to Information Retrieval. Cambridge University Press, 2008

[29] Bustos, B., Navarro, G., and Chavez, E. .Pivot Selection Techniques for Proximity Searching in Metric Spaces, Pattern Recogn. Lett. 24, 14, 2357–2366, 2003

[30] Bustos, B., Pedreira, O., and Brisaboa, N. A Dynamic Pivot Selection Technique for Similarity Search, SISAP 2008

[31] Giuseppe Amato, Fabrizio Falchi and Andrea Esuli: Pivot selection strategies for permutation-based similarity search, SISAP 2013

[32] Mico, L., Oncina, J., Vidal, E. An Algorithm for Finding Nearest Neighbours in Constant Average Time with a Linear Space Complexity, ICPR 1992

[33] Ciaccia, Paolo; Patella, Marco; Zezula, Pavel. M-tree An Efficient Access Method for Similarity Search in Metric Spaces, VLDB 1997

[34] Skopal T., Pokorný J., Snášel V.: PM-tree: Pivoting Metric Tree for Similarity Search in Multimedia Databases, ADBIS 2004

[35] Jakub Lokoc, Premysl Cech, Jiri Novák, Tomás Skopal: Cut-Region: A Compact Building Block for Hierarchical Metric Indexing, SISAP 2012

[36] Jakub Lokoc, Juraj Mosko, Premysl Cech, Tomás Skopal: On indexing metric spaces using cut-regions, Inf. Syst. 43: 1-19 ,2014

[37] David Novak, Michal Batko: Metric Index: An Ecient and Scalable Solution for Similarity Search, SISAP 2009

[38] J. Lokoč, T. Grošup, T. Skopal: On Scalable Approximate Search with the Signature Quadratic Form Distance, SISAP 2013

[39] Beecks C., Skopal T., Schöffmann K., Seidl T.: Towards Large-Scale Multimedia Exploration, DbRank 2011

[40] A. H. M. ter Hofstede, H. A. Proper, and T. van der Weide: Query formulation as an information retrieval problem. The Computer Journal, 39(4):255–274, 1996.

[41] J. Lokoč, T. Grošup, T. Skopal: Image Exploration using Online Feature Extraction and Reranking, ACM ICMR 2012

[42] J. Lokoč, T. Grošup, T. Skopal: SIR: The Smart Image Retrieval Engine, SISAP 2012

[43] Bart Thomee, Mark J. Huiskes, Erwin Bakker and Michael S. Lew: An Exploration-Based Interface for Interactive Image Retrieval, ISPA 2009

[44] J. Moško, T. Skopal, T. Bartoš and J. Lokoč: Real-Time Exploration of Multimedia Collections, ADC 2014

[45] Nicolae Suditu,François Fleuret: Iterative Relevance Feedback with Adaptive Exploration/Exploitation Trade-off , CIKM 2012

[46] Sebastian Stober , Andreas Nürnberger: Similarity adaptation in an exploratory retrieval scenario, AMR 2010

[47] Jana Urban, Joemon M. Jose and C.J. van Rijsbergen : AN ADAPTIVE APPROACH TOWARDS CONTENT-BASED IMAGE RETRIEVAL, Multimedia Tools and Applications 2006

[48] G. P. Nguyen and M. Worring: Interactive access to large image collections using similarity-based visualization, Journal of Visual Languages and Computing 2008

[49] Yushi Jing et al.: Google Image Swirl: A Large-Scale Content-Based Image Browsing System, ICME 2010

[50] Beecks C., Uysal M., Driessen P., Seidl T.: Content-Based Exploration of Multimedia Databases, CBMI 2013

[51] Susan Sim. Automatic Graph Drawing Algorithms, Information Visualization, 1996

[52] T. M. J. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. Software: Practice and Experience, 21(11):1129–1164, 1991.

[53] Klaus Schoeffmann, Manfred del Fabro: Hierarchical video browsing with a 3D carousel, ACM Multimedia 2011

[54] Klaus Schoeffmann, Marco A. Hudelist, Manfred del Fabro, Gerald Schaefer: Mobile image browsing on a 3D globe, ICMR 2012

[55] K. Schoeffmann, F. Hopfgartner, O. Marques, L. Boeszoermenyi, and J. M. Jose: Video browsing interfaces and applications: a review, SPIE 2010

[56] D. Heesch: A survey of browsing models for content based image retrieval, MTAP, 2008

[57] http://www.videobrowsershowdown.org/

[58] Klaus Schoeffmann et al., The Video Browser Showdown: a live evaluation of interactive video search tools, IJMIR 2014

[59] J. Lokoč, A. Blažek, T. Skopal: Signature-based Video Browser, VBS at MMM 2014

[60] J. Lokoč, A. Blažek, T. Skopal: On Effective Known Item Video Search Using Feature Signatures, ACM ICMR 2014

[61] J. Lokoč, T. Grošup, P. Čech, T. Skopal: Towards Efficient Multimedia Exploration Using The Metric Space Approach, CBMI 2014

[62] Benjamin Bustos, Daniel A. Keim, Tobias Schreck: A pivot-based index structure for combination of feature vectors, SAC 2005

[63] B. Bustos, T. Skopal: Dynamic Similarity Search in Multi-Metric Spaces, MIR 2006

[64] Tomáš Grošup: Similarity-based image search on the web, Bachelor thesis 2012

[65] E. Chavez, G. Navarro, R. Baeza-Yates, and J. L. Marroqun: Searching in metric spaces, ACM Comput. Surv. 2001

[66] P. Budikova, M. Batko, and P. Zezula: Evaluation platform for content-based image retrieval systems, TPDL 2011

[67] J. Lokoč, D. Novák, M. Batko, T. Skopal: Visual Image Search: Feature Signatures or/and Global Descriptors, SISAP 201

[68] M. Kruliš, J. Lokoč, T. Skopal: Efficient Extraction of Feature Signatures Using Multi-GPU Architecture, MMM 2013

[69] M. Batko, F. Falchi, C. Lucchese, D. Novak, R. Perego, F. Rabitti, J. Sedmidubsky, P.Zezula: Building a web-scale image similarity search system, Multimedia Tools and Applications 2010

[70] A. Amir et al. : Multimodal video search techniques: late fusion of speech-based retrieval and visual content-based retrieval, ICASSP 2004

[71] S. Westman, A. Lustila, P. Oittinen: Search strategies in multimodal image retrieval, IIiX 2008

[72] Renfei Li, Daling Wang, Yifei Zhang, Shi Feng, Ge Yu: An Approach of Text-Based and Image-Based Multi-modal Search for Online Shopping, WAIM 2012

[73] http://opencv.org/about.html

[74] http://lucene.apache.org/

[75] Robert C. Martin: The Principles of OOD

[76] Martin Fowler: InversionOfControl

[77] Martin Fowler: Inversion of Control Containers and the Dependency Injection pattern

[78] H. Bay, A. Ess, T. Tuytelaars, L. V. Gool: SURF: Speeded Up Robust Features, CVIU 2008

[79] Přemysl Čech: Using Metric Indexes For Effective and Efficient Multimedia Exploration, Master thesis 2014