Similarity Caching in Large-Scale Image Retrieval

Fabrizio Falchi[a], Claudio Lucchese[a], Salvatore Orlando[b], Raffaele Perego[a], Fausto Rabitti[a]

[a] *ISTI-CNR, Pisa, Italy*
[b] *Università Ca' Foscari Venezia, Italy*

Abstract

Feature-rich data, such as audio-video recordings, digital images, and results of scientific experiments, nowadays constitute the largest fraction of the massive data sets produced daily in the e-society. Content-based similarity search systems working on such data collections are rapidly growing in importance. Unfortunately, similarity search is in general very expensive and hardly scalable.

In this paper we study the case of content-based image retrieval (CBIR) systems, and focus on the problem of increasing the throughput of a large-scale CBIR system that indexes a very large collection of digital images. By analyzing the query log of a real CBIR system available on the Web, we characterize the behavior of users who experience a novel search paradigm, where content-based similarity queries and text-based ones can easily be interleaved. We show that locality and self-similarity is present even in the stream of queries submitted to such a CBIR system. According to these results, we propose an effective way to exploit this locality, by means of a similarity caching system, which stores the results of recently/frequently submitted queries and associated results. Unlike traditional caching, the proposed cache can manage not only exact hits, but also approximate ones that are solved by similarity with respect to the result sets of past queries present in the cache. We evaluate extensively the proposed solution by using the real query stream recorded in the log and a collection of 100 millions of digital photographs. The high hit ratios and small average approximation error figures obtained demonstrate the effectiveness of the approach.

1. Introduction

Content–Based Image Retrieval (CBIR) [14] systems support a search paradigms that is becoming day by day more important. Given an image, used as a query by example, users can ask the CBIR system for identifying a given number of images that are the most similar to the query by comparing their visual contents. More specifically, this content-based similarity search is realized in terms of kNN (k-Nearest Neighbors) queries that exploit an appropriate distance measure, defined over the vectors of visual features extracted from each image.

Indeed, such search paradigm fits not only digital images, but also other data types, such as audio-videos, bio-medical data, financial data, and many other scientific data. They constitute the largest fraction of data produced daily in the e-society. All these dominant data sets can be represented in feature spaces that are inherently multi-dimensional. Providing an effective and efficient way to search such multi-dimensional data sets is very challenging. This is not only due to the curse of dimensionality of the visual feature spaces used to represent images, but also to the colossal size of multimedia data over which such content-based search paradigm could be applied. Just to mention some numbers about the hugeness of such data sets, we can refer to a report by an EMC-sponsored research team of IDC (International Data Corporation). This report states that about 281 exabytes (281 billion gigabytes) of digital data were available in the world in 2007, and that by 2011 the aggregate amount of digital data will reach about 1.8 zettabytes (1,800 exabytes)

Email addresses: fabrizio.falchi@isti.cnr.it (Fabrizio Falchi), claudio.lucchese@isti.cnr.it (Claudio Lucchese), orlando@dsi.unive.it (Salvatore Orlando), raffaele.perego@isti.cnr.it (Raffaele Perego), fausto.rabitti@isti.cnr.it (Fausto Rabitti)

[23]. Already in 2008, more than 80 percent of the size of digital contents was related to images: pictures, surveillance videos, TV streams, and so forth. Moreover, if we consider the Web, we know that about 15 billions of images and 220 millions of new photos per week were uploaded to Facebook in May 2009, and about 4 billions of images and 100 millions of new photos per month were uploaded to Flickr in October 2009.

In this paper we focus on the general issue of making CBIR systems scalable, and tackle this goal by proposing a novel *similarity cache* that intercepts the *kNN* content-based similarity queries directed to a CBIR system. We show that our cache is able to satisfy a large number of these requests, with exact or approximate results. Since a hit on our cache requires a time much shorter than a kNN query on the whole index, the overall query throughput of the CBIR system is remarkably improved.

Our similarity cache differs from a traditional query result cache (e.g., the one exploited by Web search engines [16]), and is based on the mathematical foundations of *metric spaces* [10]. More specifically, the measurable similarity between the query and the cached objects is exploited for increasing the cache hit rate by allowing approximate cache hits. Besides the traditional *exact* answers, which occur when the same image query has been submitted in the past and its result set has not yet evicted from the cache, our cache can also return *approximate* answers. To this end, we exploit the cached visual features of the images belonging to the result sets of previous queries, and use them to identify the associated images that are closest to the query. Obviously, the effectiveness of our similarity cache approach cannot be evaluated in terms of the usual hit-ratio only. So in our tests we also considered the quality of the approximate results returned by the cache to show the effectiveness of our approach.

It is worth remarking that, given a query q, when the result set returned by the cache is the same as the one retrieved in the past by the back-end for the same query q, we consider it as an exact hit. Obviously, the CBIR back-end can in turn exploit any query answering approach and return either exact or approximate kNN results sets. In the last case also a cache exact hit actually returns an approximate kNN result. Our similarity cache is thus a general approach to improve CBIR throughput since it can improve the response time of any exact or approximate CBIR back-end.

In [18] we investigated an important characteristic of our similarity cache for CBIR systems: its *robustness* with respect to the *near-duplicate* images. A popular image has in fact thousands of slightly different versions in the Web. In case of near-duplicate queries, that are variants of previously cached queries, our similarity cache can return high-quality answers, namely approximate hits, without querying the CBIR back-end.

The experimental setting exploited for testing and evaluating our similarity cache is unique in the panorama of multimedia search. We refer to a real CBIR system, which indexes five MPEG-7 visual descriptors (*Scalable Color, Color Structure, Color Layout, Edge Histogram, Homogeneous Texture*) of a collection of 100 millions of Flickr photos. The set of visual descriptors associated with this image collection is publicly available for research purposes (*CoPhIR*: Content-based Photo Image Retrieval Test-Collection [7] [1]). The CBIR is MUFIN Image Search[2], where query images can be either internal or external images – i.e. images belonging to the indexed CoPhIR collection or uploaded by the user.

We analyzed a log of the queries submitted to MUFIN CBIR system between March 20th and October 28th, 2009, with the aim of devising common patterns of user behaviors that can be exploited to better understand user desiderata, improve the design of a large-scale CBIR system and its caching system, and tune its performance. To the best of our knowledge, no similar analysis was conducted before on such kind of query logs. Even if this query log is not huge if compared to the size of commonly studied query logs of Web Search engines, it can be considered representative of Web user behavior. The log in fact records 65,063 queries issued during 5,004 sessions by 3,390 distinct users. We experimentally show that the distribution of topic popularity existing for text queries [16, 34] is confirmed to some extent also for content-based image queries. In particular a high level of locality and of self-similarity can be found in the results of the queries submitted by the same and different users during their search sessions, so that the distribution of

[1] `http://cophir.isti.cnr.it`
[2] `http://mufin.fi.muni.cz/imgsearch/`

the popularity of such results is highly skewed. Moreover, we mined from the log of the MUFIN system providing the novel content-based similarity search paradigm, the most common behaviors of users, and measured their preferences for using similarity and/or text queries.

Moreover, we exploited the large experimental setting illustrated above, the MUFFIN query log and the *CoPhIR* collection, in order to evaluate carefully our similarity caching strategy. To this end we collected, for each similarity query in the log, the *Top*-200 exact closest results present in the CoPhIR collection, and used them as a sort of *ground truth* to estimate the quality of the approximate results returned by our cache, and thus to evaluate the actual benefits of using the cache in terms of improved query throughput and approximation error. It is worth mentioning that the preliminary conclusions reported in [17, 18], where we exploited a synthetic query log and a smaller image database, were not only confirmed but made remarkably stronger: our caching strategy behaves much better with a larger database and a real-life query log.

The rest of the paper is organized as follows. Section 2 reviews related work, while Section 3 discusses the results of the analysis conducted on the query log recording the content-based similarity queries issued by Web users to the MUFIN Image Search system. In Section 4 the issues related to caching the results of similarity search queries are outlined, and the framework for evaluating the efficiency and effectiveness is presented. Section 5 describes experimental settings, and reports on the results of the experiments conducted. Finally, Section 6 draws some conclusions.

2. Related Work

There are three main research topics that are related to our work. The first is concerned with WSE query log analysis and the methods for caching query results. Another is concerned with approximate search in metric spaces and the metrics for measuring effectiveness of such methods. Finally, independent of our work, another notion of similarity cache has been recently introduced. The problem as been designed and studied in a completely different context, in particular as a method to make faster contextual advertising systems.

Query result caching. Query logs constitute the most valuable source of information for evaluating the effectiveness of caching systems storing the results of past WSE queries. Many studies confirmed that users share the same query topics according to an inverse power law distribution [39, 33, 34]. This high level of sharing justifies the adoption of a caching system for Web search engines, and several studies analyzed the design and the management of such server-side caches, and reported about their performance [26, 25, 16, 3]. Lempel and Moran proposed *PDC* (Probabilistic Driven Caching), a query result caching policy based on the idea of associating a probability distribution with all the possible queries that can be submitted to a search engine [25]. PDC uses a combination of a SLRU cache (for queries regarding the first page of results), and a heap for storing answers of queries requesting pages next to the first. Priorities are computed on the basis of historical data. *PDC* performance measured on a query log of AltaVista was very promising (up to 53.5% of hit-ratio with a cache of $256,000$ elements and 10 pages prefetched).

Fagni *et al.* showed that combining static and dynamic caching policies together with an adaptive prefetching policy achieves even a higher hit ratio [16]. In their experiments, they observe that devoting a large fraction of entries to static caching along with prefetching obtains the best hit ratio. They also showed the impact of having a static portion of the cache on a multithreaded caching system. Through a simulation of the caching operations they showed that, due to the lower contention, the throughput of the caching system can be doubled by statically fixing a half of the cache entries. This behavior was confirmed also in [3] were the impact of different approaches, such as static vs. dynamic caching, and caching query results vs. caching posting lists was studied.

To the best of our knowledge this is the first paper that analyzes the query log recorded by a CBIR system. Even if the dimension and the characteristics of the query log analyzed do not allow too strong conclusion to be drawn, we can claim that that a good level of locality and self-similarity among the submitted queries can be found even in a stream of content-based image searches. Moreover, caching the results of user queries in this case has the interesting opportunity of exploiting these self-similarities with the aim of retrieving from the cache also high-quality approximate answers.

Approximate search in metric spaces. Similarity search, in particular in metric spaces, has been extensively studied in the literature [40, 32]. Due to its generally expensive cost. Recent literature has focus on distributed data structures [5, 4] and approximate search [1, 15, 20].

Several approximate techniques have been studied to improve efficiency at the price of obtaining approximate result-sets. A general justification for the use of approximation is given by the fact that similarity measures are indeed an approximation of user perception.

As suggested in [19], approaches to approximate similarity search can be broadly classified into two categories: approaches that exploit *transformations of the metric space*, and approaches that *reduce the subset of data to be examined*. In the transformation approaches, approximation is achieved by changing the object representation and/or distance function with the objective of reducing search costs. The second category of approaches is based on reducing the amount of data examined. To this aim, two basic strategies are used: *early termination*, and *relaxed branching* strategies, where the first stops the similarity search algorithm before its "precise" end, while the second avoids accessing data regions that are not likely to contain close objects. A survey of approximate similarity search techniques up to 2006 can be found in [40].

Recently, some approximate systems belonging to the family of permutation-based indexes have been proposed [1, 11, 15, 20] . MI-File [1], PP-Index [15] and the Lucene based approach proposed in [20] have also be used to efficiently index the same collection of 100M images for approximate CBIR using a central server. Finally, in [27] a novel indexing and searching mechanism called M-Index that employs most of the principles of metric space partitioning, pruning and filtering is presented. It enables approximate search indexing the data in well-established structures such as the B+-tree or even in a distributed storage. Metric Index has been used to build the MUFIN image search system [4, 27]. MUFIN is the most scalable academic CBIR prototype supporting a comprehensive content-based similarity search service on visual descriptors.

In this paper we study how a cache, storing content-based similarity queries and associated results, can be used to return approximate answers with acceptable quality guarantee, even when an exact image match is not found in the cache. Thus, our proposal introduces a novel class of approximation techniques, based on reusing the results of previously submitted queries.

Similarity cache. Recently, and independently of our work, the authors of [12] discussed a concept of similarity caching, which appears to be very close to our work. In a later work, they also applied the idea of similarity caching to Web advertising [28]. The context of application is however very different, since the authors aim to exploit similarity caching to make faster contextual advertising systems. In addition, there are several other differences with respect to our work, concerning the way the cache is maintained, and how the cache is queried to determine whether a cache hit or miss occurs.

In case of a miss, the feature vector of the query q (the authors call this query a *key*, that may correspond to a page visited by a user) is used to perform a *kNN* query on a disk-based database aimed at retrieving a much larger number of results than those actually needed. The query thus returns a large set of k relevant advertisement candidates close to q. The authors call this set of candidates the *value* associated with the *key*, and store the pair *(key, value)* in the cache. In the experiments they fixed $k = 1000$, while the final advertisement "selection" involves choosing a few relevant advertisements among the k neighbors of q stored in the cache.

The main novelty of this similarity caching proposal, which to some extent is common to ours, is the cache lookup strategy. It may also be approximate, by exploiting query similarity. Due the large set of advertisement candidates stored in the cache for each past query, given an incoming query q, the approximate cache lookup only aims at finding the cached query p that is the most similar to q. A similarity threshold is set to decide when p can be considered a good approximation, and thus corresponds to a cache hit. The retrieved data is the value associated with p, i.e. the k neighbor advertisements, over which the final selection is carried out.

Storing a large number of result for each cached query may improve the quality of "approximate hits", but affects significantly the memory footprint of the algorithm. The cache lookup method we propose here is more complex, since it considers not only one, but the h cached queries close to q. This allows us to store small result sets, providing a fine grained coverage of the database, potentially capable of answering a more

diverse set of queries. Also, we exploit the metric properties of the similarity measures to establish some guarantees about the quality of the results retrieved from the cache.

An interesting point in [28] is the way the cache content is managed. In particular, cache lines are evicted not only as a consequence of a cache miss. Indeed, the history of the queries answered by each cache line is stored. On the basis of this history, a cache line can be replaced also as a consequence of an approximate cache hit, with the aim of improving the quality of future approximate cache hits. On the other hand, in the design of our metric cache we adapted the classic LRU cache replacement policy: we mark as the most recently used the cache line that contains the past query that mostly contributes to an approximate answer to a given query.

D-cache, a main-memory structure that keeps track of distances computed for answering the queries received so far within the runtime session was proposed in [35]. Our proposed cache is a higher-level concept that can be combined with any solution employed in a content-based retrieval system. Nevertheless, the two approaches can coexist in the same system by putting our caching module in front of a D-cache-enhanced metric access structure.

3. The CBIR system Query Log

Although there exist several CBIR prototypes[3], no log of content-based similarity queries is publicly available. Recently, both Google and Microsoft Bing introduced some limited possibilities for content-based search within their popular image search services. However, no study on the use of such services is available yet.

We had access to the query log recorded by the MUFIN image search system [27, 4]. MUFIN is the most scalable academic CBIR prototype, and supports a comprehensive content-based similarity search service on the MPEG-7 visual descriptors of the CoPhIR collection. Indeed, CoPhIR is the largest publicly available collection of metadata coming from high-quality images. It contains five MPEG-7 visual descriptors (*Scalable Color, Color Structure, Color Layout, Edge Histogram, Homogeneous Texture*), and other textual information (title, tags, comments, etc.) associated with more than 100 million of photos stored on the Flickr photo-sharing website.

Besides content-based similarity, MUFIN Image Search also permits textual queries, which are simply forwarded to Flickr for retrieving a set of relevant photos according to the associated tag metadata. As regards similarity queries, any image in a MUFIN result page can be used to issue a new query by example by simply clicking on it. A user can start search from the home page of MUFIN, which visualizes a set of randomly chosen images from the CoPhIR dataset, or can use a Firefox browser plugin which allows any image displayed on a web page to be used to submit a similarity query over the CoPhIR collection. This is an interesting use of the system: while browsing the Web, a user likes a visualized image and searches the MUFIN index for similar ones.

The MUFIN system is quite popular in the Czech Republic, because its launch as an image search service was covered by a public TV channel. We had access to a query log storing the queries issued to MUFIN between March 20th and October 28th, 2009. The log records 65,063 queries issued during 5,004 sessions by 3,390 distinct Web users. Even if this query log is not huge when compared to the size of a query log of a commercial Web search engine, to some extent we believe that it can be considered representative of the behavior of Web users coming into contact with a novel search paradigm, where text-based and content-based queries can easily be interleaved.

User Sessions. Splitting the sequence of all the queries submitted by a given user into *sessions*, i.e., subsequences of somehow correlated queries, is very challenging [29, 36, 24]. Indeed, we reconstructed user sessions from the MUFIN query log in a simple way. By using IP addresses we preliminary identified all the queries coming from the same user. The queries of each user were then sorted chronologically by timestamp,

[3] see, for example, the list of CBIR systems in `http://en.wikipedia.org/wiki/List_of_CBIR_Engines`.

and each one of this sequences split to create the user sessions. A splitting point occurs when two consecutive queries are temporally far away, i.e., theirs distance is more than 30 minutes.

Unlike other approaches, we do not represent a user session as a simple sequence, but rather as a tree of queries. In fact, the log contains information about the navigational behavior of users. For example, a user who, during her session, goes back to a previously evaluated result page, and uses any result object to issue a new query. This behavior is quite common in the query log analyzed and may generate possibly complex trees of queries. Whenever possible, we used the HTTP referrer information to reconstruct the actual user behavior. Otherwise, we simply considered two consecutive queries in a session as linked.

Table 1 reports some statistics about the $5,004$ user sessions obtained from the query log. In particular, the first row of the table reports the minimum, average, and maximum number of queries per user session. It is interesting to note that the average length of user sessions in our log is much longer than whose measured on Web search engines query logs (usually only a bit larger than two queries [33]), thus indicating a very different interaction of users with the two different search tools. Moreover, the second row of the table refers to the out-degree of internal nodes of session trees. Note that the average out-degree of internal nodes is 1.19, indicating that returning back to a previously retrieved set of results for submitting another query is quite common. For the same reason, the average depth of the session trees (9.44) is remarkably smaller of the average number of queries issued per session (13.00).

Table 1: Some stats on the $5,004$ user sessions.

	min	avg	max
Queries per session	1	13.00	5303
Nodes out-degree	1	1.19	101
Session depth (tree level)	1	9.44	530

Since our query log record both text- and content-based queries, we assigned *Txt* and *Sim* labels to differentiate among them. The latter category contains queries generated by clicking on either an image displayed as a result of a previous query, or images taken from the Web through the browser plugin, or images displayed on the MUFIN home page that shows some randomly chosen pictures of the CoPhIR collection. The queries labeled with *Sim* are $38,727$ (79.16% of the total), while $10,197$ are the *Txt* queries (20.84%). About 15% of the *Sim* queries were submitted through the browser plug-in.

Table 2 reports the frequencies of the various transitions/arcs among node types present in the user tree sessions. Note that the total number of transitions to nodes of type A is different from the total number of nodes labeled A, because a node can have multiple outgoing edges as discussed above, or a user can end her session. For example, the total number of links from *Sim* nodes to *Sim/Txt* nodes are less than the total number of *Sim* nodes (38,727). In this case, the small number of outgoing edges suggests that the user is likely to end her session after having seen the result of a similarity query. Also, we can see that users searching by similarity unlikely switch to textual search. In fact, if the user issues a new query after a similarity one, the probability $p(Sim \rightarrow Txt)$ of having textual query is below 0.1. On the other hand, a textual query has a similar probability of being followed by a new textual or by a similarity search query ($p(Txt \rightarrow Txt) = 0.54$, while $p(Txt \rightarrow Sim) = 0.46$).

Table 2: Frequencies of main transitions in user sessions.

From \ *To*	*Txt*	*Sim*
Txt	5,489	4,718
Sim	2,411	26,919

Table 3 reports statistics on *chain patterns* mined from the log. Given a session tree, a chain pattern

Table 3: Per length query-chains frequency and "next-node" probabilities.

type	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Txt	10197	5489	3359	2184	1496	1071	782	600	474	372	293	226	176	133	101
		0.54	0.61	0.65	0.68	0.72	0.73	0.77	0.79	0.78	0.79	0.77	0.78	0.76	0.76
Sim	38727	26919	21486	17933	15354	13329	11738	10470	9439	8530	7766	7114	6544	6022	5560
		0.70	0.80	0.83	0.86	0.87	0.88	0.89	0.90	0.90	0.91	0.92	0.92	0.92	0.92

Table 4: Star patterns mined from the log.

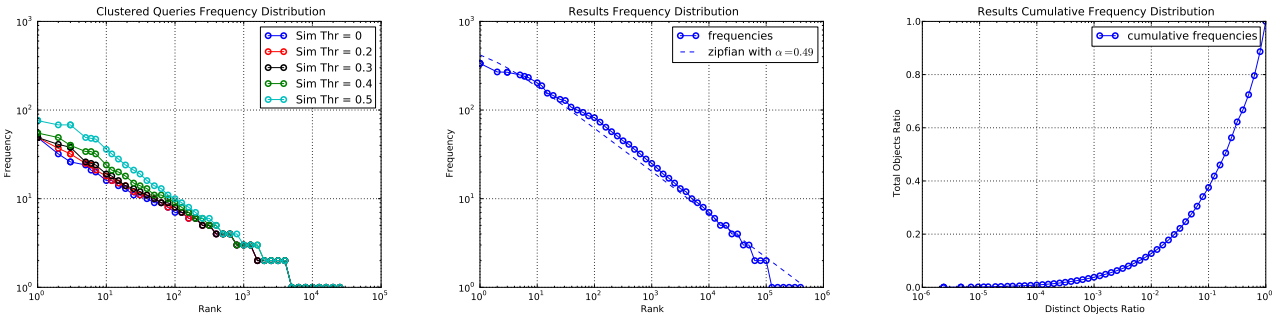
Parent	Children	1	2	3	4	5	6	7	8	9	10
Txt	Txt	4721	515	130	45	25	12	9	8	6	4
Sim	Sim	23065	2638	674	268	113	59	29	17	12	9
Txt	Sim	2932	899	397	199	113	60	34	25	16	11

consists of a path of parent-child relationships between nodes of the same type. We report the absolute amounts of the chains having a length of up to 15 nodes, but also the conditional probability of issuing the i-th query given that a chain of length $i-1$ has been issued. It is interesting to show that content-based queries are the most capable of "trapping" users. After the first *Sim* query, there is a 70% conditional probability that the user will issue another *Sim* query in the same session. This probability tends to increase with the length of the chain.

In the log we found 5560 *Sim* chains having length 15. If we consider that there are only 101 textual chains of the same length, we can assert that the easiness of query reformulation and specification is much higher for the similarity search paradigm, than in traditional textual search.

In Table 4, we report the number of *stars patterns* present in the log. For stars pattern we intend a set of nodes of a given type having the same parent of a possibly different type. This kind of patterns corresponds to a node that originates different searches of the same type. The most common star pattern, when considering up to five children, has both root and child-nodes labelled as *Sim*, even though the conditional probability of observing a second or a third child are rather small, respectively 12% and 3%. An interesting pattern found in the log is given by a *Txt* root node followed by *Sim* children. This pattern indicates a quite common behavior: a users who exploits the image result of a textual query forwarded to Flickr to start one or more similarity search queries over the MUFIN index. If a *Txt* node has one child, the probability of having a second or a third child are quite larger than before, respectively 31% and 14%.

Ground truth. Most of the $38,727$ *Sim* queries contained in the query log use images taken from the CoPhIR collection. For these images, we have the associated visual features, on whose knowledge we base the implementation of our similarity cache. The remaining *Sim* queries were instead submitted by using images taken from some Web page. Unfortunately, even if the log stores the URL of these images, 415 of them are not today available for downloading. Since we need to access them to extract the associated visual features, we were forced to remove these queries from the log. The cleaned query log still contains $30,514$ distinct query objects, accounting for a total of $38,312$ similarity queries. In order to analyze the level of sharing of images returned for user queries, and for establishing the *ground truth* necessary to measure approximation errors deriving from our caching approach, we computed, for each similarity query present in the query log, the Top-200 exact closest results present in the $100M$ images CoPhIR collection. Building such ground truth is not trivial: a brute force approach would require to compute about $3 \cdot 10^{12}$ distances! Moreover, centralized metric data structures cannot help since they does not scale to such a large number of objects. Thus, we addressed this scalability issue by extensively using optimized pivoted filtering techniques for speeding-up the computation [9]. Considering that we have to perform a large number of kNN operations on the same objects from the CoPhIR collection, we can in fact exploit the knowledge of the already computed distances between the current object and some queries to devise a lower bound to the distance between the current object and the current query. The above solution resulted very effective and



(a) (b) (c)

Figure 1: (a) Frequency distribution of query clusters. Distribution (b) and cumulative distribution(c) of image results.

allowed us to save more than 80% of distance computations in building our ground truth knowledge-base containing the Top-200 exact closest results present in the CoPhIR collection for all the distinct queries recorded in the log.

Query Locality. The most important aspect to be evaluated for the design of an effective query caching strategy is the presence of locality in users queries. In particular, query caching takes advantage of temporal locality, which occurs when the same queries are issued by many users in small temporal intervals. Indeed, it is well known that the topic popularity in text queries submitted to Web search engines is highly skewed, and even with a small cache we can obtain a significant hit ratio [16]. In the following, we investigate this skewness property in the context of similarity content-based queries. In particular, we focus on the *Sim* queries present in our query log. As above mentioned, the cleaned query log contains $38,312$ similarity queries, $30,514$ of which are distinct. These figures already give some idea of the potential benefits deriving from caching. The maximum hit ratio achievable on this query stream with a traditional cache would be equal to $(38,312 - 30,514)/38,312 \approx 0.20$. Once more we can note that the locality present in the query log of text-based Web search engines is remarkably higher. For example, the maximum hit ratio measured on a AltaVista query log studied in [16] is 0.57.

To confirm this assertion, one of the curves of Figure 1(a), labeled with *Sim Thr = 0*, shows the distribution of popularity of the *Sim* queries corresponding to distinct images. The queries are sorted by descending frequency. This distribution has a very little skew: the most frequent query image occurs in the log only 49 times! The highly skewed distribution of query topic popularity observed in the logs of textual Web search engines is very far from this figure. The high dimensionality of similarity queries, and the very sparse query space lead to a limited effectiveness of any traditional caching approach based on exact matches only.

On the other hand, the presence of near duplicates and/or very similar images on the Web may induce a more complex form of locality in similarity image queries. To investigate the presence of self-similarity among the queries in our log, we clustered them using a centroid-based algorithm. The algorithm aims to cluster together images that are similar one to each other, up to a fixed small similarity threshold. The queries in a cluster likely share something in their result sets, and our similarity cache may exploit this property. Figure 1(a) shows that, as the similarity threshold increases (*Sim Thr* ranging from 0.2 to 0.5), also the skewness of popularity distribution increased.

A complementary analysis was conducted on the result sets of the log queries. Note that the result sets depends on the indexed collection, and for this analysis we used the previously discussed *ground truth* built over the CoPhIR collection, i.e., the *Top*-200 exact closest results of each query. Figure 1(b) plots the popularities of the distinct images in the result sets associated with the various queries. Also these images were sorted in descending frequency order. Note the high skewness of this curve, with an exponent α of the fitting power law distribution equal to 0.49, thus confirming our intuition about the sharing of the results among the content-based queries. Finally, Figure 1(c) further corroborates this intuition. Suppose we store

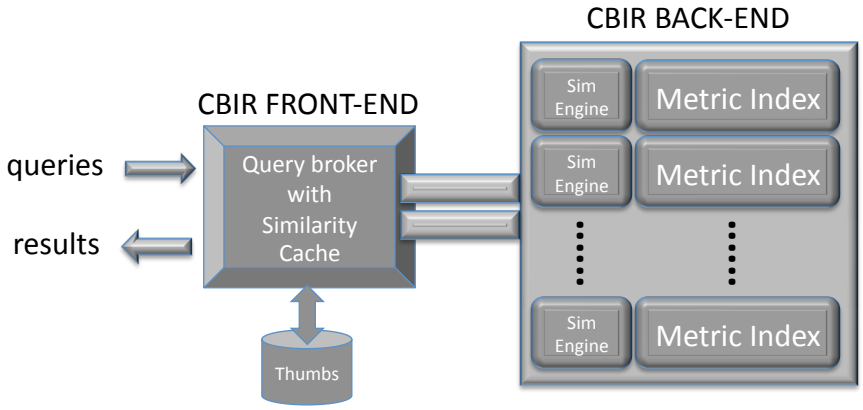


Figure 2: The architecture of a parallel/distributed CBIR system, with the similarity cache $\mathcal{C}$ placed in the front-end.

in a cache a fraction of the distinct images returned as result for the logged queries. Obviously, we start filling the cache with the most popular images. Then we measure the percentage of all the image results that can be retrieved from the cache, as a function of its size. From Figure 1(c) we can see that 1% of most popular images counts for about 15% of all the images retrieved for answering all the similarity queries present in the log.

Such large level of sharing of a small portion of the images in the collection is a worthwhile opportunity for caching result objects, that strongly motivate our similarity caching strategy.

4. Caching content-based query results

Let $\mathcal{C}$ be our similarity cache hosted on the front-end of a large-scale CBIR system, which stores recently or/and frequently submitted queries and associated results (see Figure 2). In order to compute the similarity between the cache content and the query, our method needs to keep in cache not only the *identifiers* of the results returned for a cached query, but also the *features* (e.g., the MPEG-7 visual descriptors) associated with them. As a consequence, look-up, update, eviction, and insertion operations on such a similarity cache are much more complex than those performed on a simple hash-based cache. However, a hit on the similarity cache is much cheaper to process than a query on the whole index, whose cost grows rapidly with the size of the indexed collection [40, 38].

Our similarity cache is based on the mathematical foundations of the *metric space*, which is a very general and well studied paradigm [40, 32, 21, 10, 6, 14, 8]. Therefore, the distance function exploited to measure the similarity between the features of a pair of images is assumed to be metric.

4.1. Using cache content for approximate answering

Let $\mathcal{D}$ be a collection of objects belonging to the universe of valid objects $\mathcal{U}$. Hereinafter we assume that each object $o \in \mathcal{U}$ are represented as vectors of F features (i.e., points in a F-dimensional vector space [10]), over which we define a *metric distance function*. In this work we assume that this holds for a CBIR system that manages a database D of objects. In CBIR an object is typically a description of the visual content of the images (e.g., color, texture) encoded in one or more features. A CIBR returns the most relevant images to the query by searching for similar features (objects) to the ones obtained from the query image. The relevance of the results are computed in terms of a metric distance between pairs of objects, i.e. the extracted features. In our experiments we used the MPEG-7 visual descriptors [31, 7] as features.

More formally, let d be a metric distance, $d : \mathcal{U} \times \mathcal{U} \Rightarrow \mathbb{R}$, which measures the dissimilarity between the vector representations of a pair of objects of $\mathcal{U}$. Thus, the following properties[10] hold:

$(p1)$	$\forall x, y \in \mathcal{U},\ d(x,y) \geq 0$	positiveness
$(p2)$	$\forall x, y \in \mathcal{U},\ d(x,y) = d(y,x)$	symmetry
$(p3)$	$\forall x \in \mathcal{U},\ d(x,x) = 0$	reflexivity
$(p4)$	$\forall x, y, z \in \mathcal{U},\ d(x,y) \leq d(x,z) + d(z,y)$	triangle inequality

There are basically two types of queries of interest in this metric space that can be used to retrieve from $\mathcal{D}$ the most similar objects to a given *query element/example* $q \in \mathcal{U}$. The these queries are *range queries*, and k *nearest neighbors queries* (k-NN queries).

Definition 1. *[Range query]*
A range query *returns all the objects in $\mathcal{D}$ at distance at most r from a given query object $q \in \mathcal{U}$. More formally, let $R_{\mathcal{D}}(q,r)$ be the result set of the query, $R_{\mathcal{D}}(q,r) \subseteq \mathcal{D}$, defined as $R_{\mathcal{D}}(q,r) = \{o \in \mathcal{D} \mid d(q,o) \leq r\}$.*

Definition 2. *[kNN query]*
A kNN query *returns the k closest objects in $\mathcal{D}$ to q. More formally, let $kNN_{\mathcal{D}}(q,k)$ be the result set of the query, where $kNN_{\mathcal{D}}(q,k) \subseteq \mathcal{D}$ and $|kNN_{\mathcal{D}}(q,k)| = k$, for which the following property holds: $\forall\ x \in kNN_{\mathcal{D}}(q,k)$, and $\forall\ y \in (\mathcal{D} \setminus kNN_{\mathcal{D}}(q,k))$, $d(q,x) \leq d(q,y)$.*
Let r_q be the radius of the smallest hypersphere centered in q that contains all its k nearest neighbors in $\mathcal{D}$. More formally, $\exists\ r_q$ such that $\forall\ x \in kNN_{\mathcal{D}}(q,k)$, we have that $d(q,x) \leq r_q$.

Since a CBIR system is usually asked to retrieve the *Top@k* relevant images to a given query example q, in the following we will assume that a kNN queries (for some fixed k) is exploited.

Our cache $\mathcal{C}$ stores a set of past queries and associated results. Therefore, when we say that $q_i \in \mathcal{C}$, we actually mean that $\mathcal{C}$ non only contains query $q_i \in \mathcal{U}$, but also all the objects $o \in kNN_{\mathcal{D}}(q_i,k) \subseteq \mathcal{D}$. Note that objects o always belongs to collection $\mathcal{D}$, while a query q_i may not. Users can in fact submit any image as query example to the CBIR system. Moreover, the same object $o \in \mathcal{D}$ can be shared by several result sets $kNN_{\mathcal{D}}(q_i,k)$ stored in $\mathcal{C}$, and thus if $o \in \mathcal{C}$, then it may belong to any set $kNN_{\mathcal{D}}(q_i,k)$ associated with a cached query q_i.

Let us now consider a query q arriving to the CBIR system. The look-up of $\mathcal{C}$ can result in two possible *cache hits* (besides obviously a *cache miss*):

- *exact hit*: if another occurrence of q was previously submitted, and still not evicted from $\mathcal{C}$, then its exact k results in $kNN_{\mathcal{D}}(q,k)$ are known, and can be immediately returned to the user.
- *approximate similarity hit*: if q is not currently present in $\mathcal{C}$, but some similar queries were previously seen, we are interested in understanding whether the cached objects could be used to return some "approximate" results for $kNN_{\mathcal{D}}(q,k)$. A cache hit occurs if the quality of such results is good enough, otherwise we have a miss.

Before discussing our algorithm to manage the cache, we formalize the problem, and discuss some theoretical results that can be used to evaluate the quality of an approximate similarity hit.

Let $R_{\mathcal{C}}(q,r)$ and $kNN_{\mathcal{C}}(q,k)$ be respectively the results a range query and a kNN query processed over the collection of the objects stored in $\mathcal{C}$. Thus, $R_{\mathcal{C}}(q,r) \subseteq \mathcal{D}_{\mathcal{C}}$ and $kNN_{\mathcal{C}}(q,k) \subseteq \mathcal{D}_{\mathcal{C}}$, where $\mathcal{D}_{\mathcal{C}} = \{o \in \mathcal{D} \mid o \in kNN_{\mathcal{D}}(q_i,k)$ and $q_i \in \mathcal{C}\}$. Therefore, the following theorem and corollary hold.

Theorem 1. *Given a query q, and a previously submitted query $q_i \in \mathcal{C}$ such that $d(q,q_i) < r_{q_i}$, let*

$$s_q(q_i) = r_{q_i} - d(q,q_i)$$

be the safe radius *of q with respect to the cached query object q_i. The following holds:*

$$R_{\mathcal{C}}(q, s_q(q_i)) = R_{\mathcal{D}}(q, s_q(q_i)) = kNN_{\mathcal{D}}(q,k')$$

where $k' = |R_{\mathcal{C}}(q, s_q(q_i))| \leq k$.

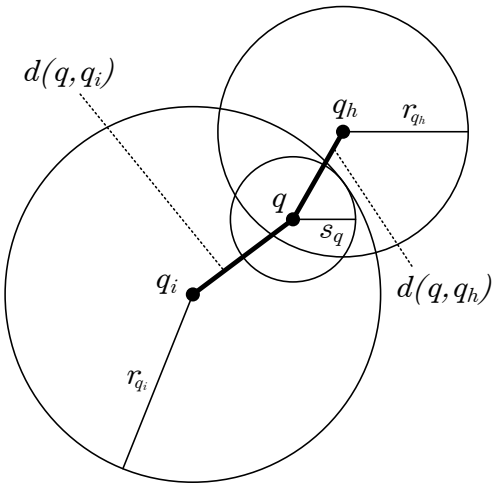


Figure 3: A query object q falling into two hyperspheres containing $kNN(q_i,k)$ and $kNN(q_h,k)$, i.e. the k nearest neighbors of either a cached query object q_i or q_h. The radius s_q is set by using Lemma 1.

Proof 1. *Let o be an object in $R_{\mathcal{D}}(q,s_q(q_i))$. From the triangle inequality property, we can derive that $d(o,q_i) \leq d(o,q) + d(q,q_i)$. Since $d(o,q) \leq s_q(q_i)$, we have that*

$$\begin{aligned} & d(o,q_i) \leq \quad d(o,q) + d(q,q_i) \\ \Rightarrow \quad & d(o,q_i) \leq \quad s_q(q_i) + d(q,q_i) \\ \Rightarrow \quad & d(o,q_i) \leq \quad r_{q_i} - d(q,q_i) + d(q,q_i) \\ \Rightarrow \quad & d(o,q_i) \leq \quad r_{q_i} \end{aligned}$$

which means that $o \in R_{\mathcal{D}}(q_i,r_{q_i})$. By assuming no ties[4]*, we have that $R_{\mathcal{D}}(q_i,r_{q_i}) = kNN_{\mathcal{D}}(q_i,k)$. Considering that $kNN_{\mathcal{D}}(q_i,k)$ is a cached result set, then we can conclude that $o \in \mathcal{C}$.*

We have proved that $o \in R_{\mathcal{D}}(q,s_q(q_i))$ implies $o \in \mathcal{C}$, and therefore $o \in R_{\mathcal{C}}(q,s_q(q_i))$, i.e. $R_{\mathcal{D}}(q,s_q(q_i)) \subseteq R_{\mathcal{C}}(q,s_q(q_i))$. Moreover, $\mathcal{C} \subseteq \mathcal{D}$ implies $R_{\mathcal{C}}(q,s_q(q_i)) \subseteq R_{\mathcal{D}}(q,s_q(q_i))$. Thus $R_{\mathcal{C}}(q,s_q(q_i)) = R_{\mathcal{D}}(q,s_q(q_i))$. Finally, $|R_{\mathcal{D}}(q,s_q(q_i))| = k'$ and, by definition of kNN, $R_{\mathcal{D}}(q,s_q(q_i)) = kNN_{\mathcal{D}}(q,k')$.

Corollary 1. *Given a query object q, let*

$$s_q = s_q(\widehat{q}) \qquad with \qquad \widehat{q} = arg\,max_{q_i \in \mathcal{C}}(r_{q_i} - d(q_i,q)) \tag{1}$$

be the maximum safe radius *of q w.r.t. the current content of cache $\mathcal{C}$. We can exactly solve in $\mathcal{C}$ the range query $R_{\mathcal{D}}(q,s_q)$, i.e., $R_{\mathcal{C}}(q,s_q) = R_{\mathcal{D}}(q,s_q)$.*

The above Theorem and Corollary state that there is a radius s_q for which we can exactly solve the range query $R_{\mathcal{D}}(q,s_q)$ by only using the cached objects. In turn, the result of such range query corresponds to the top k', $k' = |R_{\mathcal{C}}(q,s_q)| = |R_{\mathcal{D}}(q,s_q)| \leq k$, nearest neighbors of q in $\mathcal{D}$. These objects can be thus used to build an approximate result set for query q, where k' of the objects retrieved from $\mathcal{C}$ are *exactly* the top k' results of $kNN_{\mathcal{D}}(q,k')$.

Figure 3 shows a simple example with objects and queries in a two-dimensional Euclidean space. Intuitively, every cached query q_i induces the complete knowledge of the metric space up to distance r_{q_i} from q_i.

[4]Note that, according to Definition 2, if there were more objects in $\mathcal{D}$ at a distance r_q from q, the composition of $kNN_{\mathcal{D}}(q,k)$ would not be unique. For the sake of simplicity, in this proof we assume no ties and thus the unicity of $kNN_{\mathcal{D}}(q,k)$, which implies that $kNN_{\mathcal{D}}(q,k) = R_{\mathcal{D}}(q,r_q)$ and thus $|R_{\mathcal{D}}(q,r_q)| = k$. However the theorem and the proof could be easily generalized to take into account the possible ties of more objects at distance r_q from q.

If any subsequent query q is found to be inside the hypersphere centered in q_i with radius r_{q_i}, then, as long as we look inside this hypersphere, we also have complete knowledge of the k', $k' \leq k$, nearest neighbors of q.

4.2. The QCache *algorithm*

In a previous work [17] we introduced two different algorithms, RCache (Result Cache) and QCache (Query Cache), which exploit the above Theorem and Corollary to manage a similarity cache $\mathcal{C}$. Since QCache resulted to be more efficient than RCache, whereas the quality of the returned approximate results were comparable, in this paper we focus on QCache and discuss some further improvements to the adopted similarity caching algorithm.

QCache adopts a traditional hash table used to detect whether an incoming query was already seen in the past and not evicted from the cache. If an exact hit occurs, the associated results can be soon returned to the user. Otherwise, even if the submitted query was never seen before, QCache can still provide an approximate result set from $\mathcal{C}$, without querying the CBIR back-end. In addition, it is possible to guarantee the quality of the returned objects. In summary, QCache:

- in addition to exact hits, is able to answer with approximate similarity hits, thus improving the overall hit ratio and thus increasing the CBIR system throughput;
- evaluates and guarantees the quality of the approximate results returned to users.

A straightforward way to implement such a similarity cache is to evaluate the distance between a new query q and the queries currently stored in $\mathcal{C}$. If an identical past query is not present in $\mathcal{C}$, but there is a query q_i that is sufficiently similar to q, then the associated results can be retrieved from the cache and returned to the user. The assumption that similar queries have similar results is strongly reasonable under the metric space assumption. Note that, even if q and q_i are not very close, this method could also work reasonably if, for each cached query, we store a large set of nearest objects (much more than k) and then select the closest ones as the result set for q. A similar approach is followed in [28]. Conversely, we only cache $kNN_{\mathcal{D}}(q_i, k)$ for each cached query q_i. Then, rather than searching for the most similar query q_i to q, we look for the h most similar queries present in the cache that may answer q with sufficient accuracy. This choice of caching short result lists has some interesting advantages. First, the CBIR back-end is not overloaded with *wide*, and thus expensive, queries. Second, this allows to store more queries in the limited-size cache, thus resulting in a larger representativity of $\mathcal{D}$ in the the cached data.

Hence, when a new query q arrives, and it does not exactly match any past queries, QCache has to perform similarity search over the cache content to choose the h most similar past queries and build the approximate result set. Unfortunately, similarity search is computationally expensive, and searching over all the queries and the associated result objects may impact negatively on the efficiency of the cache. We overcome this issue by exploiting a metric data structure[5] for indexing the past queries only, and by separately storing the associated results. This reduces the size of the index by a factor of k. Consequently, also the time for searching over the similarity index is significantly improved.

QCache thus exploits this index to look for the h cached queries that are the most similar to q. Let Q be such set, $|Q| = h$. The k closest objects to q, hereinafter $\mathcal{R}_{q,k}$, are then selected among the $h \times k$ results of the queries in Q. The larger h is, the more detailed the knowledge about the portion of space surrounding q and the probability of providing high quality approximate results are. Unfortunately, when the value of h is increased, also the cost of building $\mathcal{R}_{q,k}$ increases correspondingly.

However, in order to efficiently build $\mathcal{R}_{q,k}$, instead of trivially computing $h \times k$ distances, we exploit a max heap data structure and a pivot-based filtering technique [9], which considers each query $q_i \in Q$ as a pivot for its associated results. In detail, the search process works as follows. The heap is initially filled with the k results from the first query in Q. Then the sets of results of the remaining queries are considered one at a time. At any point during this process, let z be the object stored in the root of the heap, i.e., the farthest object from q among the current set of candidates in $\mathcal{R}_{q,k}$. Let o_{ij} be an object in the result set of

[5]The current implementation use an M-tree data structure [13].

$q_i \in Q$ to be evaluated for checking its inclusion in $\mathcal{R}_{q,k}$. We can exploit the following lemma to prune o_{ij}, i.e. to decide that surley $o_{ij} \notin \mathcal{R}_{q,k}$ without computing the actual distance $d(q, o_{ij})$.

Lemma 1. *If $|d(q_i, o_{ij}) - d(q_i, q)| \geq d(q, z)$, then o_{ij} is surely not closer to q than z, i.e. $d(o_{i,j}, q) \geq d(z, q)$.*

Proof 2. *From the triangle inequality property of metric spaces, we can easily deduce that $d(q, o_{ij}) \geq |d(q_i, o_{ij}) - d(q_i, q)|$. Whenever $|d(q_i, o_{ij}) - d(q_i, q)| \geq d(q, z)$ holds, also $d(q, o_{ij}) \geq d(q, z)$ holds, i.e., the candidate o_{ij} is surely not closer to q than z.*

Note that, besides $d(q, z)$, both $d(q_i, o_{ij})$ and $d(q_i, q)$ are already known, and thus if $|d(q_i, o_{ij}) - d(q_i, q)| \geq d(q, z)$ we can conclude that $o_{ij} \notin \mathcal{R}_{q,k}$. Conversely, if $|d(q_i, o_{ij}) - d(q_i, q)| < d(q, z)$, the actual distance $d(q, o_{ij})$ must be computed. If $d(q, o_{ij}) < d(q, z)$, we add o_{ij} to $\mathcal{R}_{q,k}$ and update the max heap accordingly.

Once the approximate result set $\mathcal{R}_{q,k}$ has been built, QCache has to asses its quality, thus deciding whether it can be returned as a approximate hit. Otherwise a miss occurs, and the CBIR back-end must be queried. To this end, QCache adopts the following two-step heuristic strategy.

1. *Safe radius.* By using the theoretical results of the previous section, we could consider $\mathcal{R}_{q,k}$ an approximate hit if $s_q = s_q(\widehat{q}) > 0$, and we are able to determine that the top k' objects in $\mathcal{R}_{q,k}$, $k' > 1$, are exact. However, the QCache algorithm aims to return the answer $\mathcal{R}_{q,k}$ by only exploiting Q, a subset of all the cached queries. Therefore Q may not contain $\widehat{q}$, and thus we may not able to compute s_q. Figure 3 shows an example of this case. Suppose that $|Q| = h = 1$. In order to determine the largest safe radius s_q we would need to consider a query (q_i in the figure) that is not the closest to q (q_h in the figure). However, by only considering the queries in Q, we can compute $\widetilde{s}_q = s_q(\widetilde{q})$, where $\widetilde{q} = \arg\max_{q_i \in Q} s_{q_i}$. From our tests, for not trivial values of $h = |Q|$, $\widetilde{s}_q$ is very often exact, or is a very good lower bound for s_q. We can thus employ $\widetilde{s}_q$ to count the number k' of the top correct objects in $\mathcal{R}_{q,k}$. If $k' > 1$, then we consider the approximate result a good quality approximate hit.
   However, we experimentally evaluated that only using this technique for assessing the quality of $\mathcal{R}_{q,k}$ is often too conservative. For example, $\mathcal{R}_{q,k}$ very often returns correct objects even when $s_q \leq 0$, and thus q is not contained in any of the hyperspheres of the queries in Q. Therefore, rather than considering $\mathcal{R}_{q,k}$ as a miss, we further estimate its quality by using the second heuristic discussed below.
2. *Probability distributions of distances from q.* Given the closest h queries in Q, we assume that their corresponding results have a similar distribution in this portion of the search space. Therefore, also $\mathcal{R}_{q,k}$ should comply with such distribution in order to be a high quality result. More formally, let d_j^i be the distance of the i-th result from its corresponding query $q_j \in Q$. We assume that the distances d_j^i, for every $q_j \in Q$, are distributed as a normal distribution $\mathcal{N}(\mu_i, \sigma_i^2)$. Let r^i be the distance from q of the i-th result in $\mathcal{R}_{q,k}$, we measure its probability of being an accurate result as the probability of r^i being drawn from $\mathcal{N}(\mu_i, \sigma_i^2)$, denoted with $P(r^i \mid Q)$. The quality of $\mathcal{R}_{q,k}$ can finally be estimated over all the k results r_i as follows:

$$P(\mathcal{R}_{q,k} \mid Q) = \prod_{1 \leq i \leq k} P(r^i \mid Q)$$

   Actually, we use the logarithm of $P(\mathcal{R}_{q,k} \mid Q)$, and we approximate the probability $P(r^i \mid Q)$ with the value assumed by the probability density function of $\mathcal{N}(\mu_i, \sigma_i^2)$ applied to r^i. Therefore, the quality of $\mathcal{R}_{q,k}$ is estimated with the following goodness function:

$$\begin{aligned}
\mathsf{Goodness}(\mathcal{R}_{q,k} \mid Q) &= \log \prod_{1 \leq i \leq k} \mathcal{N}(r^i; \mu_i, \sigma_i^2) \\
&= \log \prod_{1 \leq i \leq k} \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp\left(-\frac{(r_i - \mu_i)^2}{2\sigma_i^2}\right) \\
&= -\sum_{1 \leq i \leq k} \log\left(\sqrt{2\pi\sigma_i^2}\right) + \frac{(r_i - \mu_i)^2}{2\sigma_i^2}
\end{aligned}$$

Thus, QCache has a tuning parameter, i.e. a minimum required goodness threshold that we denote with γ. Only if $\mathsf{Goodness}(\mathcal{R}_{q,k}|Q) < \gamma$, we consider $\mathcal{R}_{q,k}$ as a miss, otherwise QCache returns this result set as a approximate hit.
Finally, since the queries $q_j \in Q$ being far away from q may be less significative for evaluating the objects' distribution around q, we weighted the various d_j^i so that far queries have exponentially smaller influence. More specifically, μ_i is estimated as follows:

$$\mu_i = \frac{1}{k_c}\frac{1}{W}\sum_{q_j \in Q} d_j^i \cdot d(q, q_j)^{-5}$$

where W is a normalizing constant summing up the weights $d(q, q_j)^{-5}$. Similarly for the variance σ_i^2.

Finally, it is worth mentioning that the usual update of the cache to implement the LRU replacement policy not only occurs for exact hits, but also for approximate hits. Thus, whenever an approximate result set $\mathcal{R}_{q,k}$ is considered of sufficient quality, and returned to the user as a hit, the query $\widetilde{q}$ associated with the estimated maximum safe radius $\widetilde{s}_q$ is promoted, i.e., it is marked as the most recently used in order to procrastinate its eviction from the cache.

Algorithm 1 Algorithm QCache

```
1:  procedure LookUp(q, k)
2:      if q ∈ H then
3:          Results ← H.get(q)
4:          H.move-to-front(q)
5:      else
6:          Q ← kNN_M(q, h)
7:          O ← {o ∈ kNN_D(q_j, k) | q_j ∈ Q}
8:          R_{q,k} ← kNN_O(q, k)
9:          q̃ ← arg max_{q_i ∈ Q} s_q(q_i)
10:         s̃_q ← s_q(q̃)
11:         if sufficientQuality(R_{q,k}, s̃_q) then
12:             H.move-to-front(q̃)
13:             Results ← R_{q,k}
14:         else
15:             Results ← kNN_D(q, k)
16:             H.put(q, Results)
17:             M.put(q)
18:         end if
19:     end if
20:     return Results
21: end procedure
```

The algorithm. In Algorithm 1 we show the pseudocode of our QCache algorithm. We make use of a hash table $\mathcal{H}$ which is used to store and retrieve efficiently queries and their result lists, by using queries as hash keys. Given a new incoming query object q, the first step is to check whether q is present or not in cache (line 2).

If this is not the case, the algorithm tries to use stored results to devise an approximate answer. This step uses the metric index $\mathcal{M}$, which contains only the recent past query objects $q_i \in \mathcal{C}$ to find the h cached queries that are the closest ones to q (line 6). The resulting set Q is first used to retrieve from the cache all the associated $h \times k$ results (line 7), and to extract from them only the k closest objects to query q (line 8). Such objects form the approximated result list $\mathcal{R}_{q,k}$. Then Q, is used to find an approximation $\widetilde{s}_q$ of the maximum safe radius and the corresponding query $\widetilde{q}$ (lines 9–10). The safe radius $\widetilde{s}_q$ is needed to understand which of the top results are guaranteed to be correct according to Theorem 1.

The expected quality of the approximate answer, considering also the safe radius $\widetilde{s}_q$, is then evaluated (line 11). However, as discusses above, even if we cannot find any of these top exact objects or the safe radius $\widetilde{s}_q$ is negative, the function exploits the probability distributions of the object distances from q to assess the quality of $\mathcal{R}_{q,k}$. If the quality is considered good enough, we still have an approximate hit. Otherwise, we have a miss, and the query is forwarded to $\mathcal{D}$. Its (exact) results are added into the two indexing structures $\mathcal{H}$ and $\mathcal{M}$ (lines 15–17).

Note that the hash table $\mathcal{H}$ is managed with a simple LRU policy. If a new insertion cannot be satisfied because the cache is full, a pair $\langle q_i, kDD_{\mathcal{D}}(q_i, k)\rangle$ is evicted from $\mathcal{H}$, and $\mathcal{M}$ is updated consequently. Finally, if $\widetilde{q}$ is used to produce an good approximate result, then a move-to-front is applied to $\widetilde{q}$ (line 12), i.e. it is marked to be the most recently used, thus allowing useful queries to persist in cache.

5. Results Assessment

To the best of our knowledge, this is the first rigorous evaluation in the area of multimedia content-based search involving a real-life query log, generated by the search activity of a large number of users, over a large collection of objects. As discussed in Section 3, our evaluation is based on a query log of the MUFIN Image Search system, which answers to content-based similarity queries over the CoPhIR collection.

The distance between two images is computed as a weighted sum of the distances between each of the five MPEG-7 descriptors used. These distances were proposed by the MPEG group [22, 31]. The distance between two images is thus metric, according to our metric space assumptions. For more details see [4].

A total of 38,312 similarity queries where extracted from the MUFIN query log. Among them, we used the chronologically first 8,312 queries to warm-up the cache, and the remaining 30,000 to actually measure the effectiveness of our caching algorithm.

Whenever a cache miss occur, the query q should be forwarded to CBIR back-end to retrieve the k nearest neighbors of q in the CoPhIR collection. Note that, in order to achieve scalability and near real-time responsiveness, the MUFIN system returns approximate results. Conversely, for a precise evaluation of the quality of the approximate answers returned by QCache, we preferred to pre-compute the exact results for each query in the log. We called this knowledge the "ground truth" associated with the query log and the image CoPhIR collection.

Most of the tests were conducted by varying the cache size, ranging from as little as 10MB to 500MB. Note that the size of the five visual features associated with an image is about 1KB. Therefore, a cache being 10MB large can store up to 10,000 objects, both results and queries, and index $10,000/(20+1) \approx 476$ queries, since we stored 20 results for each cached query. Finally, for each query in the log, the kNN query was issued with parameter $k = 20$, which is a reasonable number of results for an image search system, and QCache searched for the $h = |Q| = 20$ closest queries to build the approximate result. We found experimentally that a good minimum goodness threshold is $\gamma = 15$, this setting was used in all the tests but the ones measuring efficiency and efficacy of QCache as a function of γ.

5.1. Cache hit-ratios

Figure 4(a) reports the exact, approximate, and total hit ratios as a function of the cache size. The plot reports also the hit ratio of a traditional (exact) cache of infinite size, that works as an hash table of past queries. The first observation is that QCache has a remarkably large hit ratio, larger than 40% for the largest cache size experimented. However, even with the smallest cache size of 10MB, QCache achieves a hit ratio slightly larger than the infinite cache hit ratio ($\approx 26\%$). Note that a 500MB cache can store just 0.5% of the feature database. Since, on large collections, processing a similarity query via a metric index has a cost proportional to the size of the indexed collection [38], this result shows that the proposed caching technique can have an impressive impact on the overall performance of a CBIR system.

Finally, we would like to highlight an interesting trade-off arising when the cache may return approximate answers. In case of a miss, QCache may answer approximately without querying the underlying database. On the one hand, this improves the throughput of the system, since less queries are processed by the slower back-end. On the other hand, if the same query occurs again in the log, the cache will not store the

corresponding results, and it will probably answer again approximately. Therefore, approximate hits may reduce the number of exact ones. This explains why the number of exact hits is remarkably smaller of that measured for an infinite cache, even with the largest cache size.

However, the sum of exact and approximate hits succeeds in dramatically breaking the barrier of the infinite cache, allowing to answer even to queries that were never seen before. In the following, we evaluate the quality of such approximate results.

5.2. Approximation quality

We distinguish between ranking-based and distance-based quality measures. The former compare the given approximate results list with the exact result list. The quality is thus given by the position of the approximate results in the exact list. The latter are based on the distances of the results from the query. Rather than comparing the objects' positions, these measures take into consideration their distances from the query. Distance-based measures are quite relevant for similarity search, since they allow to measure the quality of approximate results on the basis of their similarity to the query.

In order to assess the efficacy of QCache, we measure the quality of the generated results by considering only the approximate hits and not the exact ones. Given an approximate result set $\mathcal{R}_{q,k}$ returned by QCache and the corresponding exact result set $kNN_{\mathcal{D}}(q,k)$, we consider the following quality measures:

- Distance-based error measures:
  - Relative Error on the Sum of distances (RES):
$$RES(\mathcal{R}_{q,k}) = \frac{\sum_{x \in \mathcal{R}_{q,k}()} d(q,x)}{\sum_{y \in kNN_{\mathcal{D}}(q,k))} d(q,y)} - 1$$
  - Relative Error on the Maximal distance (REM):
$$REM(\mathcal{R}_{q,k}) = \frac{\max_{x \in \mathcal{R}_{q,k}()} d(q,x)}{\max_{x \in kNN_{\mathcal{D}}(q,k)} d(q,x)} - 1$$
- Ranking-based quality measures:
  - Precision (P):
$$P(\mathcal{R}_{q,k}) = |\mathcal{R}_{q,k} \cap kNN_{\mathcal{D}}(q,k)|$$
  - Top-K Correctness (TK):
$$TK(\mathcal{R}_{q,k}) = \text{maximum } k' \text{ such that } |\mathcal{R}_{q,k} \cap kNN_{\mathcal{D}}(q,k')| = k'$$

RES is basically the well-known total distance ratio used in [37, 19], while REM is the relative error on distance for the k-th nearest neighbor – a generalization of the measure originally proposed in [2].

Figure 4(b) plots the values of the average RES and REM error measures on varying the cache size. The plot reports the quality of the approximate results actually returned by the cache, but also of the potential results that could be extracted from QCache and were not returned in case of a cache miss, i.e. when the expected quality of the result set was not considered good enough. The error measured for the misses is always much larger than the one measured for the hit cases, meaning that, on average, the estimate of the results quality was correct. More importantly, both the average and maximum approach 10% for the larger, and more realistic, cache sizes. This shows that approximate results produced by QCache exhibit good quality consistently.

In Figure 4(c,d) we report on Top-K Correctness and Precision. In this case, the impact of cache size is much more apparent. The fraction of approximate result lists containing at least 10 correct objects doubles from ≈20% to ≈40% when using the largest cache size. However, both Precision and Top-K Correctness witness the goodness of the QCache algorithm. About 90% of the approximate results contain at least 3 correct objects, and in 1/4-th of the cases those 3 results are exactly the 3 nearest neighbors. The top-K correctness drops more quickly. Still, almost 40% of the approximate results contain at least 10 correct objects.

5.3. Impact of the number h of close cached queries considered

In Figure 4(e) the hit ratio as a function of h is plotted. There are two interesting aspects. First, for small values of h, the number of approximate hits is very small. A small number of cached queries does not provide a sufficient number of high quality result objects. When increasing h, the QCache algorithm can choose among a larger number of results, and build a sufficiently good approximate results set. The number h of close queries considered may affect the performance of QCache. However, the cost of the look up does not increase significantly with h, while the hit ratio measured grows very quickly as the value of h is increased.

From the figure we can see however that the increase in the number of approximate hits may reduce the number of exact ones. In fact, as above discussed, when a query is answered with an approximate hit, the same is likely to happen for the next occurrences of the same query. This fact could merit further attention. We leave as future work the analysis of this issue, and of effective heuristics aimed at improving selectively the quality of approximation for popular queries.

5.4. Efficiency versus Efficacy tradeoff

Finally, in Figure 4(e) we analyze the sensitivity of the algorithm to the minimum quality threshold γ. Not only the threshold γ can be used to tune the quality of the returned approximate results, but it has an important role in the overall performance of the system. With a large value of γ, only high-quality results are returned by the cache. This implies that the hit ratio of the cache is reduced. In Figure 4(e), we can see that when $\gamma = 30$, the hit ratio is lower than 30%, but the precision increases to about 85%. Conversely, with a less strict threshold $\gamma = 0$, the precision of results drops to about 45%, but the hit ratio measured exceeds the 50% of the queries threshold. Note that the precision reported in this plot includes both exact and approximate hits.

The above numbers are very competitive in the field of CBIR approximate answering. In particular, the precision figures measured on the same dataset for the PP-Index [15], and MI-File[1] systems are 65%, and 55% respectively. On the other hand, MUFIN achieves almost 80% precision by exploiting an index distributed over 32 machines working in parallel. The precision of the QCache algorithm (considering both exact and approximate hits) ranges from 45% to 85%, depending on γ, still providing, for any of the adopted threshold, a hit ratio larger than a traditional exact cache of infinite size. Therefore, we can claim that our cache provides high quality results in the context of CBIR systems, and high hit ratios.

By looking at Figure 4(e) we can see that the value of γ drives the tradeoff between hit ratio and the quality of the returned results. It is easy to find a proper value accomplishing the desired quality of results. More interestingly, the threshold could be used dynamically according to some external policy. For instance, in presence of bursts of queries, a less strict threshold would allow to keep up with a large number of users and to reduce to load at the back end of the system. On the other hand, when user load is low, the quality of results returned may be pursued at the cost of an increased load on the CBIR back-end.

6. Conclusions

Starting from the analysis of a query log recording the behavior of Web users experimenting a uncommon paradigm of image search based on image content similarity, we have discussed and evaluated a model for caching query results based on the concept of similarity. Our goal was to show that similarity caching is a viable and effective way to improve efficiency of a large-scale CBIR system processing a realistic workload.

We studied a large log recording the queries issued to the MUFIN Image Search system in a 7-months period. Although its relatively limited size did not allow us to draw strong conclusions, the log is surely representative of the behavior of an average Web user. From the analysis of the logged user sessions we detected common patterns that highlight differences in the behavior of users of a CBIR and a traditional text-based search engine. In particular we devised a tree model for user sessions, and measured the frequencies of different chain and star patterns appearing in the log, which are motivated by the particular browsing possibilities offered by the user interface of the CBIR system. We also estimated and discussed the distribution of the popularity of the content-based queries present in the log. Unlike keyword-based searches,

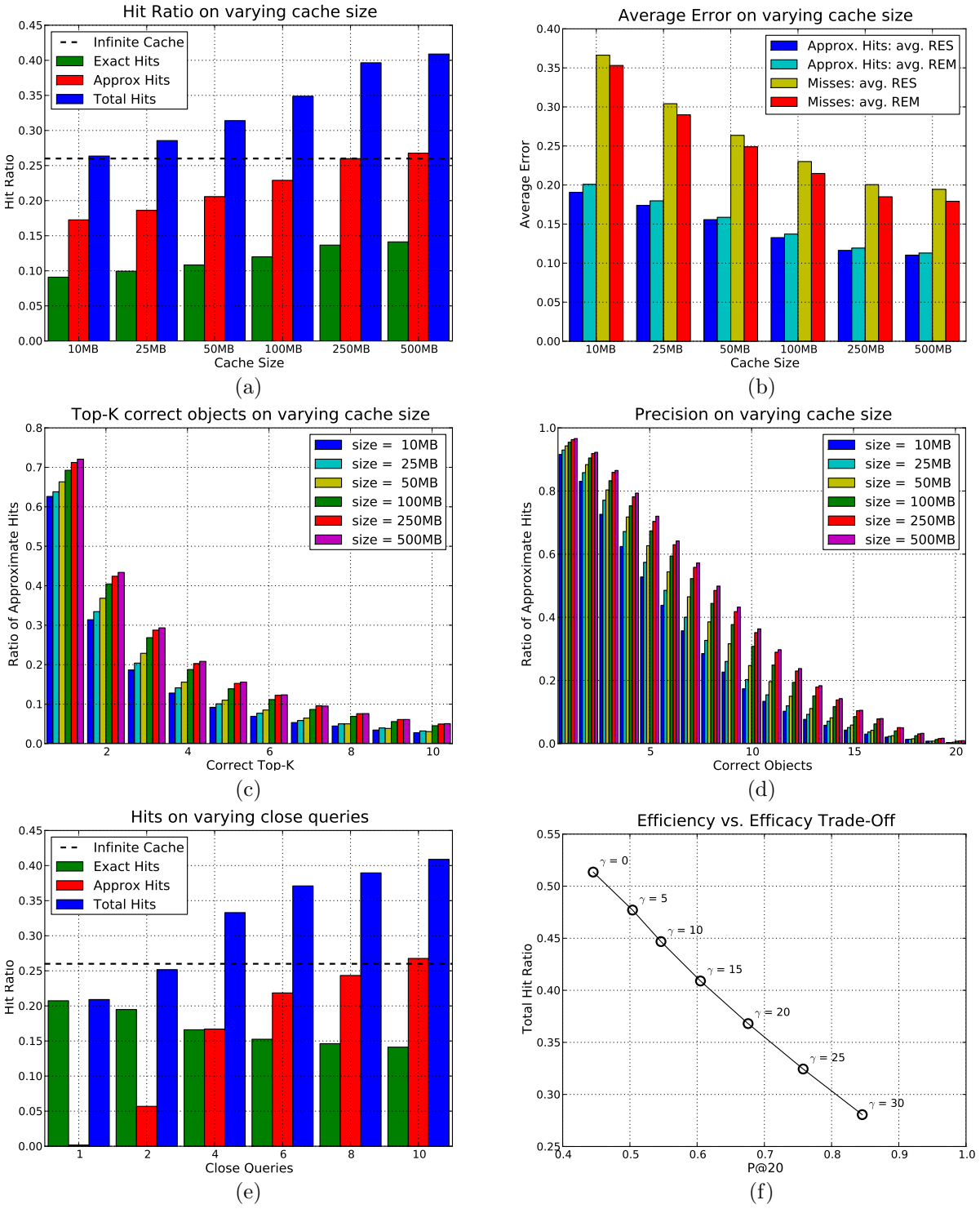


Figure 4: (a) QCache exact and approximate hit ratios, (b) average error, (c) Top-K Correctness, (d) Precision as a function of cache size. (e) QCache hit ratios and approximation error on varying number of close queries h. (f) Performance on varying γ.

the analysis showed a limited skewness that prevent a successful exploitation of the query result caching strategies developed for Web search engines. On the other hand, we identified the presence of self-similarities among the user queries, and, more importantly, we demonstrated that the distribution of most frequently relevant images is very skewed. This experimental evidence was possible due to the public availability of the collection indexed, which allowed us to derive the "ground truth", i.e., the exact set of top-200 results closest to each query image in the log. This evidence strongly motivated the exploitation in a similar setting of our novel caching approach, which, unlike traditional caching strategies, may return a set of approximate results also when the submitted query object was never seen in the past, or was previously evicted from the cache. By testing our similarity cache with the log query stream, we got very high hit ratios and tunable small approximation error figures. As an example, a similarity cache with a size equal to 500MB obtained a hit ratio equal to 42%, with a precision at 20 of 60%. Moreover, the largest contribution to the hit ratios was due to approximate cache answers, once more showing that it is worth pursuing this research direction. Our caching algorithm should not be considered an approximate search technique, but rather an orthogonal approach aiming at improving the response time of any exact or approximate similarity search system. The generality of metric space, which is the only assumption of our work, makes our contribution even more important, as our similarity cache can be applied at a large variety of application scenarios, such as biology, geography, multimedia, data cleaning and integration, etc.

Future work will regard the application of our similarity caching approach to other areas (e.g. advertisement, video recognition, data cleaning etc.). Moreover, even if our approach has its mathematical foundation and result quality guarantees on the notion of metric spaces, it would be interesting to investigate its applicability, with possible performance degradation, to similarity search systems in quasi-metric or non-metric feature spaces. Another interesting issue regards investigating the adoption of an incremental caching policy, similar to that proposed in [30], to improve the precision of results returned by the similarity cache for queries having an increasing popularity: when a query resulting in an approximate hit is going to become popular, it may be worthwhile to fill the cache with its precise results in order to improve the quality of results for future occurrences of the same query.

7. Acknowledgments

We would like to thank Pavel Zezula and his group at the Masaryk University (Brno, Czech Republic) for making available to us the query log of the MUFIN Image Search system. This work would have not been possible without their contribution. We acknowledge the partial support of S-CUBE (EU-FP7-215483), ASSETS (CIP-ICT-PSP-250527), and VISITO Tuscany (POR-FESR-63748) projects.

References

[1] G. Amato and P. Savino. Approximate similarity search in metric spaces using inverted files. In *Proceedings of the 3rd international conference on Scalable information systems*, InfoScale '08, pages 28:1–28:10, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[2] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, 1998.

[3] R. A. Baeza-Yates, A. Gionis, F. Junqueira, V. Murdock, V. Plachouras, and F. Silvestri. Design trade-offs for search engine caching. *TWEB*, 2(4), 2008.

[4] M. Batko, F. Falchi, C. Lucchese, D. Novak, R. Perego, F. Rabitti, J. Sedmidubsky, and P. Zezula. Building a web-scale image similarity search system. *Multimedia Tools and Applications*, electronic edition, August 07, 2009.

[5] M. Batko, D. Novak, F. Falchi, and P. Zezula. Scalability comparison of peer-to-peer similarity search structures. *Future Generation Comp. Syst.*, 24(8):834–848, 2008.

[6] C. Böhm, S. Berchtold, and D. A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Comput. Surv.*, 33(3):322–373, 2001.

[7] P. Bolettieri, A. Esuli, F. Falchi, C. Lucchese, R. Perego, T. Piccioli, and F. Rabitti. Cophir: a test collection for content-based image retrieval. *CoRR*, abs/0905.4627, 2009.

[8] T. Bozkaya and M. Ozsoyoglu. Indexing large metric spaces for similarity search queries. *ACM Trans. Database Syst.*, 24(3):361–404, 1999.

[9] B. Bustos, G. Navarro, and E. Chávez. Pivot selection techniques for proximity searching in metric spaces. *Pattern Recogn. Lett.*, 24(14):2357–2366, 2003.

[10] E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín. Searching in metric spaces. *ACM Computing Surveys (CSUR)*, 33(3):273–321, 2001.
[11] E. Chavez Gonzalez, K. Figueroa, and G. Navarro. Effective proximity retrieval by ordering permutations. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30:1647–1658, September 2008.
[12] F. Chierichetti, R. Kumar, and S. Vassilvitskii. Similarity caching. In *PODS*, pages 127–136, 2009.
[13] P. Ciaccia, M. Patella, and P. Zezula. M-Tree: An efficient access method for similarity search in metric spaces. In *Proceedings of VLDB'97, August 25–29, 1997, Athens, Greece*, pages 426–435, 1997.
[14] R. Datta, D. Joshi, J. Li, and J. Z. Wang. Image retrieval: Ideas, influences, and trends of the new age. *ACM Computing Surveys*, 2007.
[15] A. Esuli. Pp-index: Using permutation prefixes for efficient and scalable approximate similarity search. In *Proceedings of the 7th Workshop on Large-Scale Distributed Systems for Information Retrieval (LSDS-IR09)*, pages 17–24, 2009.
[16] T. Fagni, R. Perego, F. Silvestri, and S. Orlando. Boosting the performance of web search engines: Caching and prefetching query results by exploiting historical usage data. *ACM Trans. Inf. Syst.*, 24(1):51–78, 2006.
[17] F. Falchi, C. Lucchese, S. Orlando, R. Perego, and F. Rabitti. A metric cache for similarity search. In *Proceeding of the 2008 ACM Workshop on Large-Scale Distributed Systems for Information Retrieval, LSDS-IR 2008, Napa Valley, California, USA, October 30, 2008*, pages 43–50, 2008.
[18] F. Falchi, C. Lucchese, S. Orlando, R. Perego, and F. Rabitti. Caching content-based queries for robust and efficient image retrieval. In *EDBT*, pages 780–790, 2009.
[19] H. Ferhatosmanoglu, E. Tuncel, D. Agrawal, and A. El Abbadi. Approximate nearest neighbor searching in multimedia databases. In *Data Engineering, 2001. Proceedings. 17th International Conference on*, 2001.
[20] C. Gennaro, G. Amato, P. Bolettieri, and P. Savino. An approach to content-based image retrieval based on the lucene search engine library. In *Research and Advanced Technology for Digital Libraries, 14th European Conference, ECDL 2010, Glasgow, UK, September 6-10, 2010. Proceedings*, volume 6273 of *Lecture Notes in Computer Science*, pages 55–66. Springer, 2010.
[21] G. R. Hjaltason and H. Samet. Index-driven similarity search in metric spaces (survey article). *ACM Trans. Database Syst.*, 28(4):517–580, 2003.
[22] ISO/IEC. Information technology - Multimedia content description interfaces. Part 6: Reference Software, 2003. 15938-6:2003.
[23] P. D. John F. Gantz, C. Chute, A. Manfrediz, S. Minton, D. R. W. Schlichting, and A. Toncheva. The diverse and exploding digital universe. an updated forecast of worldwide information growth through 2011. Technical report, 2008.
[24] R. Jones and K. L. Klinkner. Beyond the session timeout: automatic hierarchical segmentation of search topics in query logs. In *In Proc. CIKM'08*, pages 699–708, New York, NY, USA, 2008. ACM.
[25] R. Lempel and S. Moran. Predictive caching and prefetching of query results in search engines. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 19–28, New York, NY, USA, 2003. ACM Press.
[26] E. P. Markatos. On Caching Search Engine Query Results. *Computer Communications*, 24(2):137–143, 2001.
[27] D. Novak and M. Batko. Metric index: An efficient and scalable solution for similarity search. In *Proceedings of the 2009 Second International Workshop on Similarity Search and Applications*, SISAP '09, pages 65–73. IEEE Computer Society, 2009.
[28] S. Pandey, A. Z. Broder, F. Chierichetti, V. Josifovski, R. Kumar, and S. Vassilvitskii. Nearest-neighbor caching for content-match applications. In *WWW*, pages 441–450, 2009.
[29] B. Piwowarski and H. Zaragoza. Predictive user click models based on click-through history. In *In Proc. CIKM'07*, pages 175–182, New York, NY, USA, 2007. ACM.
[30] D. Puppin, F. Silvestri, R. Perego, and R. A. Baeza-Yates. Tuning the capacity of search engines: Load-driven routing and incremental caching to reduce and balance the load. *ACM Trans. Inf. Syst.*, 28(2), 2010.
[31] P. Salembier and T. Sikora. *Introduction to MPEG-7: Multimedia Content Description Interface*. John Wiley & Sons, Inc., New York, NY, USA, 2002.
[32] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Computer Graphics and Geometric Modeling. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.
[33] C. Silverstein, H. Marais, M. Henzinger, and M. Moricz. Analysis of a very large web search engine query log. *SIGIR Forum*, 33(1):6–12, 1999.
[34] F. Silvestri. Mining query logs: Turning search usage data into knowledge. *Foundations and Trends in Information Retrieval*, 4(1-2):1–174, 2010.
[35] T. Skopal and B. Bustos. On index-free similarity search in metric spaces. In *Proceedings of the 20th International Conference on Database and Expert Systems Applications*, DEXA '09, pages 516–531, Berlin, Heidelberg, 2009. Springer-Verlag.
[36] J. Teevan, E. Adar, R. Jones, and M. A. S. Potts. Information retrieval: repeat queries in yahoo's logs. In *In Proc. SIGIR'07*, pages 151–158, New York, NY, USA, 2007. ACM.
[37] E. Tuncel, H. Ferhatosmanoglu, and K. Rose. Vq-index: an index structure for similarity searching in multimedia databases. In *ACM Multimedia*, pages 543–552, 2002.
[38] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proc. 24th Int. Conf. Very Large Data Bases, VLDB*, pages 194–205, 1998.
[39] Y. Xie and D. O'Hallaron. Locality in search engine queries and its implications for caching. In *Proceedings of IEEE INFOCOM 2002, The 21st Annual Joint Conference of the IEEE Computer and Communications Societies*, 2002.
[40] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search. The Metric Space Approach*, volume 32 of *Advances in Database Systems*. 233 Spring Street, New York, NY 10013, USA, 2006.