Caching Content-based Queries for Robust and Efficient Image Retrieval

Fabrizio Falchi
ISTI-CNR
Pisa, Italy
fabrizio.falchi@isti.cnr.it

Claudio Lucchese
ISTI-CNR
Pisa, Italy
claudio.lucchese@isti.cnr.it

Salvatore Orlando
Università Ca' Foscari
Venezia, Italy
orlando@dsi.unive.it

Raffaele Perego
ISTI-CNR
Pisa, Italy
raffaele.perego@isti.cnr.it

Fausto Rabitti
ISTI-CNR
Pisa, Italy
fausto.rabitti@isti.cnr.it

ABSTRACT

In order to become an effective complement to traditional Web-scale text-based image retrieval solutions, content-based image retrieval must address scalability and efficiency issues. In this paper we investigate the possibility of caching the answers to content-based image retrieval queries in metric space, with the aim of reducing the average cost of query processing, and boosting the overall system throughput. Our proposal exploits the similarity between the query object and the cache content, and allows the cache to return approximate answers with acceptable quality guarantee even if the query processed has never been encountered in the past. Moreover, since popular images that are likely to be used as query have several near-duplicate versions, we show that our caching algorithm is robust, and does not suffer of cache pollution problems due to near-duplicate query objects. We report on very promising results obtained with a collection of one million high-quality digital photos. We show that it is worth pursuing caching strategies also in similarity search systems, since the proposed caching techniques can have a significant impact on performance, like caching on text queries has been proven effective for traditional Web search engines.

Categories and Subject Descriptors

H.3.3 [**Information Systems**]: Information Storage and Retrieval—*information search and retrieval, search process*

General Terms

Algorithms, Performance.

Keywords

content-based retrieval, query-result caching, metric space, query popularity, near-duplicate images.

1. INTRODUCTION

With the widespread use of digital cameras, more than 80 billion photographs are taken each year [15], and a significant part of them, over one billion[1], are published on the Web. According to [15], already in 2003 digital images contributed for the largest part of Web content, and their management promises to emerge as a major issue in the next years.

In this context, the interest in searching such huge collections of images by their content is rapidly growing [6]. The challenge of Web-scale Content–Based Image Retrieval (CBIR) systems is thus the ability to scale up, and to become, in the near future, a valid complement to the consolidated multimedia search paradigm based on textual metadata only.

Content-based similarity queries ask for retrieving from the indexed collection the most relevant objects, i.e. the closest to the query object according to the adopted distance function. Unfortunately, this search paradigm is inherently very expensive due to the curse of dimensionality that holds for image visual features, and the query processing cost grows very rapidly with the size of the collection indexed.

In this paper, we tackle the scalability issues of CBIR by investigating the possibility of retrieving the results of content-based queries from a *cache* located in front of the system. Our aim is to reduce the average cost of query resolution, thus boosting the overall performance. Being a very general and well studied paradigm, we based our cache on the mathematical foundations of *metric spaces* [24, 18, 12, 4, 1, 6, 2]. To the best of our knowledge, this is the first proposal of a caching framework designed to exploit the results of previously submitted content–based similarity search queries. In text-based Web Search Engines (WSEs), query logs and caching were extensively studied [7]. In particular, researchers discovered that topic popularity follows a Zipf-like distribution that allows also small caches to be very effective in capturing most popular queries and reducing the pressure to the WSE back-end.

Although large query logs are not yet available for CBIR systems, we argue that also for content-based queries, the distribution of topic popularity existing for text queries will

[1] source `http://www.lyra.com`

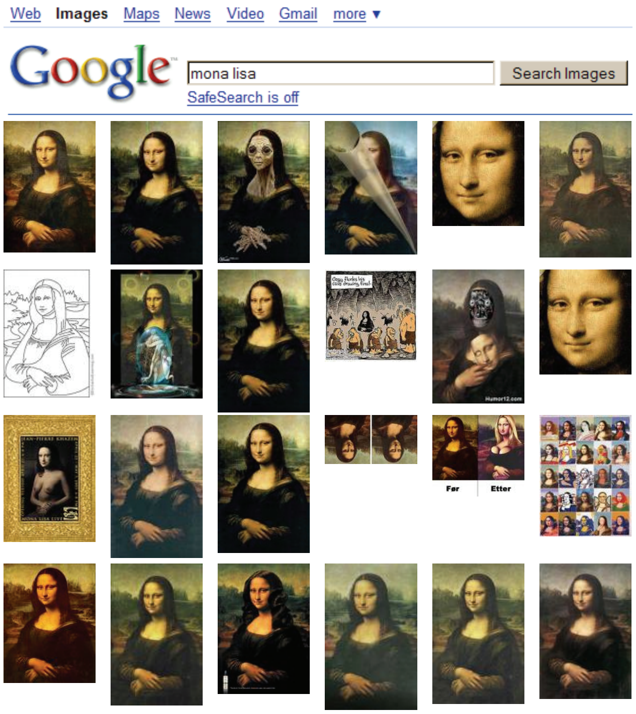


Figure 1: Top results for the query “mona lisa”.

be confirmed to some extent. This is suggested for example by the distribution of popularity rank scores on large-scale photo sharing sites [20].

However, the cache we propose is very different from a traditional cache for WSEs, which can be thought as a simple hash table, whose keys are the submitted queries, and the stored values are the associated pages of results, with some policy for replacement of cache entries based on the recency and/or frequency of references. In our case, the possible (and measurable) similarity among the query submitted and the cached objects can be exploited in order to increase the number of queries answered directly by the caching system. In fact, our cache is able to return an answer without querying the underlying content-based index in two very different cases: (a) an *exact* answer when exactly the same query was submitted in the past, and its results were not evicted from the cache; (b) an *approximate* answer composed of the closest objects currently cached when the quality of such approximated answer is acceptable according to a given measure. Obviously, the effectiveness of such a metric cache cannot be evaluated in terms of the usual hit-ratio, but has to consider also the quality of the approximate results returned.

Moreover, we want our cache to be robust with respect to the problem of *near-duplicate* images. A popular image has in fact thousands of slightly different versions in the Web. See for example Fig. 1, which show the top results returned by Google Images for the query “mona lisa”. In [10], Zobel et al. analyzed this phenomenon, and discovered that such images are usually variants derived from the same original image. Since it is very likely for these near-duplicate images to be used as query objects, a traditional cache that answers exact matches only, would be soon polluted with a lot of very similar result sets. In fact, even in the case of a slight resizing or cropping of the original image, the visual features used for specifying the content-based query would be different. For example, let us consider the first and second image shown in Fig. 1: in the case these two near-duplicate images would be subsequently used as queries to a CBIR, they would correspond to two different entries in a traditional cache. Our cache instead exploits the metric property of the distance measure for evaluating the quality of the approximate results that can be returned. In case of a near-duplicate variant of a previously cached query, we can thus return a high-quality answer without querying the search engine back-end, thus avoid inserting in the cache a duplicated result set.

It is worth noting that although this paper deals with CBIR, the caching technique proposed is completely general, and can be adopted in any scenario in which we need to boost large-scale similarity-based search services for metric objects (e.g., medical data, DNA sequences, financial data).

The rest of the paper is organized as follows. Section 2 briefly review related work, while Section 3 discusses the issues related to caching the results of similarity search queries, proposes a novel theoretical background, and a framework for evaluating the efficiency and effectiveness of integrating a caching subsystem within an actual search by content system. Section 4 describes experimental settings, and reports on the promising results of the experiments conducted. Finally, Section 5 draws some conclusions.

2. RELATED WORK

The research topics more related to our work are query result caching in WSEs, and techniques for performing or measuring effectiveness of approximate search in metric spaces.

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 [23, 19]. 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 [16, 14, 7]. Lempel and Moran proposed *PDC* (Probabilistic Driven Caching), a query answers caching policy based on the idea of associating a probability distribution with all the possible queries that can be submitted to a search engine [14]. 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 previously submitted queries. The distribution is built over statistics computed on the previously submitted queries. For all the queries that have not previously seen, the distribution function evaluates to zero. This probability distribution is used to compute a priority value that is exploited to order the entries of the cache: highly probable queries are highly ranked, and have a low probability to be evicted from the cache. Indeed, a replacement policy based on this probability distribution is only used for queries regarding pages subsequent to the first one. Furthermore, *PDC* is the first policy to adopt prefetching to anticipate user requests. To this purpose, *PDC* exploits a model of user behavior. A user session starts with a query for the first page of results, and can proceed with one or more *follow-up* queries (i.e. queries requesting successive page of results). When no follow-up queries are received within τ seconds, the session is considered finished. This model is exploited in *PDC* by demoting the priorities of the entries of the cache referring to queries submitted more than τ seconds ago. To keep track of query priorities, a priority queue is used. *PDC* results measured on a query log of AltaVista were 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 [7]. 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.

Unfortunately, since Web-scale search engines and associated logs are not yet available for content-based image queries, we cannot directly generalize results and analyses of caches for text-based WSEs. In particular, we cannot assume in a stream of content-based queries the presence of the same level of locality present in text-based searches. On the other hand, caching content-based similarity search queries has the interesting opportunity of exploiting similarities between the current query and the objects stored in the cache with the aim of giving efficiently also approximate answers.

In this paper we will make some pragmatic assumptions for characterizing user querying behavior: we will use real data coming from a popular photo-sharing site to build both the data collection (1 million digital photos), and the query logs (built synthetically by considering the actual usage of this million photos by real users).

Approximate search in metric spaces. Due to the generally expensive cost of similarity search in metric spaces, several approximate techniques have been studied to improve efficiency at the price of obtaining less accurate 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 [9], 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 the search cost. 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. Surveys of approximate similarity search techniques can be found in [24] and [18].

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.

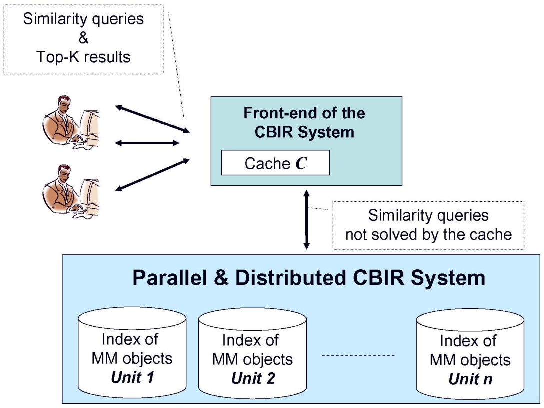

Similarity queries
&
Top-K results
Front-end of the CBIR System
Cache C
Similarity queries not solved by the cache
Parallel & Distributed CBIR System
Index of MM objects Unit 1
Index of MM objects Unit 2
Index of MM objects Unit n

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

3. CACHING CONTENT-BASED QUERY RESULTS

Let $\mathcal{C}$ be a cache placed in front of a CBIR system, storing recently / frequently submitted queries and associated results (see Figure 2). The rationale of exploiting cache $\mathcal{C}$ is that, when a hit occurs, we can avoid submitting the content-based query to the back-end of the information retrieval system, and we can soon return the results stored in the cache, thus saving the computational cost of processing the query and improving the overall throughput.

In order to compute the similarity between the cache content and the submitted query, our method needs to keep in cache not only the *identifiers* of the objects returned by a cached query, but also the *features* associated with these objects. Object features are in fact needed to measure the metric distances between them and a new submitted query object. As a consequence, look-up, insert, and delete operations on such a similarity-based, metric cache are much more complex and expensive than those performed on a simple hash-based cache. On the other hand, processing content-based similarity search queries is highly demanding, with a computational cost growing rapidly with the size of the collection indexed [24, 21].

3.1 Using cache content for approximate answering

Let $\mathcal{D}$ be a collection of objects belonging to the universe of valid objects $\mathcal{U}$, and let d be a *metric distance function*, $d : \mathcal{U} \times \mathcal{U} \Rightarrow \mathbb{R}$, which measures the similarity between two objects of $\mathcal{U}$. $(\mathcal{U}, d)$ corresponds to a metric space, which is the basic assumption of our work.

The database $\mathcal{D}$ can be queried for similar objects by using two different kind of queries: *range queries*, and *k nearest neighbors queries*. A range query $R_{\mathcal{C}}(q, r)$ returns all the objects in the database at distance at most r from a given

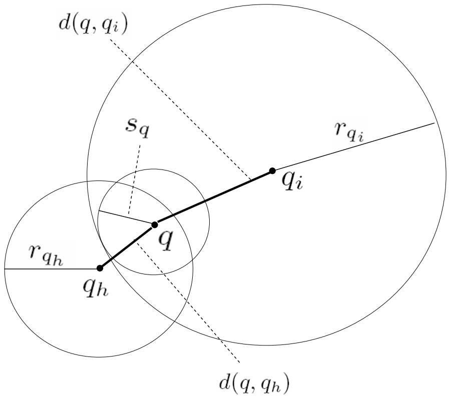


Figure 3: A query object q falling into two hyperspheres containing $kNN(q_i, k)$ and $kNN(q_h, k)$, i.e. the k nearest neighbors of either a cached query object q_i or q_h. The radius s_q is set by using Lemma 1.

query object $q \in \mathcal{U}$, i.e. $R_{\mathcal{D}}(q, r) = \{o \in \mathcal{D} \mid d(q, o) \leq r\}$. A kNN query, $kNN_{\mathcal{D}}(q, k)$, returns instead the k nearest objects to query q. We call r_q the radius of the smallest hypersphere centered in q and containing all its k nearest neighbors in $\mathcal{D}$. Note that the above definition of kNN is sound if there is only one object in $\mathcal{D}$ at distance r_q from q, otherwise the k-th nearest neighbor is not unique. For the sake of simplicity, and without loss of generality, we assume that the above condition is always satisfied, and therefore that $|R_{\mathcal{D}}(q, r_q)| = k$ which implies $kNN_{\mathcal{D}}(q, k) = R_{\mathcal{D}}(q, r_q)$.

We will focus on kNN queries (for some fixed k) since such query are more interesting in similarity search applications such as CBIR. However, the case of range queries is very similar and even simpler. Thus, we assume that cache $\mathcal{C}$ stores a set of past queries and their k results. We use an overloaded notation, saying that $q_i \in \mathcal{C}$ if the cache contains the query q_i along with all the objects in $kNN_{\mathcal{D}}(q_i, k)$. Moreover, we say that $o \in \mathcal{C}$ if object o belongs to any set $kNN_{\mathcal{D}}(q_i, k)$ associated with a cached query q_i. Note that o always belongs to collection $\mathcal{D}$, while q_i may not.

Let us now consider a query q arriving to the CBIR system. If another occurrence of q was previously submitted, and still not evicted from $\mathcal{C}$, then its exact results are known, and they can be immediately returned to the user. Conversely, if q is currently not present in $\mathcal{C}$, but some similar query was previously seen, we are interested in understanding whether the cached objects could be used to return some approximate results for $kNN_{\mathcal{D}}(q, k)$, and, if so, we would like to provide some a-priori measure for the quality of such results. In the following we will show that it is possible to understand whether or not some of the objects stored in the cache are among the top $k' \leq k$ neighbors of the new query q.

Let $R_{\mathcal{C}}(q, r)$ and $kNN_{\mathcal{C}}(q, k)$ be the results of the above defined range and kNN queries processed over the collection of cached objects. The following theorem and corollary hold:

THEOREM 1. *Given a new incoming query object q, and a cached query object $q_i \in \mathcal{C}$ 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 w.r.t. 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 $k' = |R_{\mathcal{C}}(q, s_q(q_i))| \leq k$.

PROOF. 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

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

which means that $o \in R_{\mathcal{D}}(q_i, r_{q_i})$, and since we are assuming no ties, $R_{\mathcal{D}}(q_i, r_{q_i}) = kNN_{\mathcal{D}}(q_i, k)$, meaning that $o \in \mathcal{C}$, being $kNN_{\mathcal{D}}(q_i, k)$ a cached result set.

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 new incoming query object q, let*

$$s_q = \max(0, \max_{q_i \in \mathcal{C}}(r_{q_i} - d(q, q_i)))$$

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 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', $k' = |R_{\mathcal{C}}(q, s_q)| \leq k$, nearest neighbors of q in $\mathcal{D}$. The result of such range query can be used to build an approximate result set for query $kNN(q, k)$, where the top k' objects of the approximate answer are *the same* as the top k' results of the exact answer.

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

In a preliminary work [8] we introduced two different algorithms, RCache (Result Cache), and QCache (Query Cache), to manage cache $\mathcal{C}$ by exploiting the above results. Since QCache resulted to be more efficient than RCache, whereas the quality of the returned approximate results were comparable, in this paper we only sketch the RCache cache algorithm, and focus the attention on discussing and evaluating QCache in a scenario where near-duplicate images, which abundantly populate the Web, are submitted as query objects by the users of a large-scale CBIR system.

3.2 The RCache algorithm

RCache exploits a hash table $\mathcal{H}$ used to store and retrieve efficiently query objects and their result lists. RCache also adopts a metric index $\mathcal{M}$ that is used to perform kNN searches over the cached objects in order to return approximate answers when possible.

Whenever the cache is looked-up for the k objects that are the most similar to a given query object q, $\mathcal{H}$ is first accessed to check for an exact hit, which occurs when q and its kNN results are already stored in the cache. In this case, the associated result set $kNN_{\mathcal{D}}(q, k)$ is promptly returned. In case this exact hit does not occur, the cache metric index $\mathcal{M}$ is accessed for finding the k objects closest to q among all the objects stored in cache.

Along with each returned object $o \in \mathcal{M}.kNN(q, k)$, $\mathcal{M}$ stores q_o, the query object of the kNN query that returned o among the results. By using this information, RCache is able to compute the maximum safe radius s_q. Using s_q we can check if there exists some k', $k' \leq k$, for which the top k' results obtained from $\mathcal{C}$ are guaranteed to be among the results that would be retrieved from $\mathcal{D}$.

Unfortunately, RCache requires to build a metric index over all the objects returned by the kNN queries stored in $\mathcal{C}$. This makes cache management very complex, and the computational cost of (approximate) query hits high. On the other hand, QCache, whose algorithm is discussed in the next Section, needs to build a metric index over the query objects only. As a consequence, cache management results less expensive and more efficient.

3.3 The QCache algorithm

The RCache algorithm can be considered, to some extent, the most straightforward way for realizing our metric cache, where the approximate hits are returned only if the requested quality guarantee can be ensured. Given a query object q, if no exact hit is found, all the cached objects are used to determine the best possible result set, i.e. to compute $kNN_{\mathcal{C}}(q, k)$. Later RCache uses the cached information relative to the the result set to determine the maximum safe radius s_q, in turn used to evaluate how many objects in the approximate result set are the same as in the exact answer.

Unlike RCache, QCache maintains a metric index only over the query objects of previously submitted kNN queries, whose result sets are stored in the cache. This reduces by a factor k, where k is the parameter of the kNN queries, the number of objects indexed in the the metric index.

The main idea of QCache is to solve a kNN query in an approximate way by first finding a set of suitable query objects among the cached ones, and then using their neighbors, i.e. the result sets of the corresponding kNN queries, to produce the approximate answer. Using this set of suitable query objects, QCache can also determine the maximum safe radius s_q, and only if it turns out to be not trivial ($s_q > 0$), it proceeds with the query resolution. Note that QCache works in the opposite way of RCache, which first determines the complete result set of the kNN query, and then the safe radius s_q to evaluate the quality of the result set.

Let us detail the QCache algorithm, by first discussing how it computes the maximum safe radius q of an approximate query. Given a query object q, suppose that s_q is not trivial ($s_q > 0$). Let $\widehat{q}$ be the query associated with it:

$$\widehat{q} = arg\,max_{q_i \in \mathcal{C}}(r_{q_i} - d(q_i, q))$$

Given the cached query object $\widehat{q}$, the cached result set of $kNN_{\mathcal{D}}(\widehat{q}, k)$ contains the k' objects that are guaranteed to be the top-k' nearest neighbors of q in $\mathcal{D}$, $k' \leq k$.

In order to provide the best possible approximate answer, we need: 1) to find efficiently $\widehat{q}$ among all the cached query objects; 2) to choose the additional $k - k'$ objects that are needed to complete the approximate result set of $kNN_{\mathcal{C}}(\widehat{q}, k)$.

Unfortunately, discovering $\widehat{q}$, or equivalently the maximum safe radius s_q, is not straightforward. We have seen that s_q is the maximum value (if greater than 0) of $(r_{q_i} - d(q, q_i))$ for the various $q_i \in \mathcal{C}$. If we assume that the various values of r_{q_i} have a stable mean $\overline{r}$, we can approximate s_q by maximizing $(\overline{r} - d(q, q_i))$. This can be done by searching for the cached query q_i being the nearest to q.

This strategy may not find $\widehat{q}$. As an example, consider Figure 3, which shows that, in order to determine the largest safe radius s_q, we may need to consider a cached query (q_i) that is not the closest one (q_h) to new incoming query q.

In order to increase the chance of success in finding $\widehat{q}$, we thus pick the k_c cached query objects that are the closest one to q, i.e. the k_c queries maximizing $(\overline{r} - d(q, q_i))$, and search among them the one that actually maximizes the safe radius $(r_{q_i} - d(q, q_i))$. The rationale of this method is to leverage the variance of r_{q_i}, still avoiding to explore exhaustively all the queries stored in cache.

Let us denote with $\widetilde{s}_q$ the resulting approximated value of s_q, and with $\widetilde{q}$ the corresponding query object. It holds that $\widetilde{s}_q$ is a lower bound of s_q. Given $\widetilde{q}$ and $\widetilde{s}_q$, we can guarantee that the cached result set $kNN_{\mathcal{C}}(\widetilde{q}, k)$ contains k^*, $k^* \leq k$, of the k nearest neighbors of q in $\mathcal{D}$. These k^* objects $o \in kNN_{\mathcal{D}}(\widetilde{q}, k)$ are at distance at most $\widetilde{s}_q$ from q.

Finally, we need to choose a set of $k - k^*$ additional objects to produce the approximate answer. Still, we want to avoid searching among all the cached objects, and thus limit the search to the kNN objects close to the k_c cached queries previously selected. Intuitively, by having selected the k_c cached queries being the closest ones to q, the probability of finding a good results within the cached k nearest neighbors of such queries is very high.

Algorithm 1 Algorithm QCache

```
 1: procedure LookUp($kNN(q, k)$)
 2:     if $q \in \mathcal{H}$ then
 3:         $R_{q,k} \leftarrow \mathcal{H}$.get($q$)
 4:     else
 5:         $Q \leftarrow \mathcal{M}.kNN(q, k_c)$
 6:         $O \leftarrow \{o \in kNN_{\mathcal{D}}(q_j, k) \mid q_j \in Q\}$
 7:         $R_{q,k} \leftarrow O.kNN(q, k)$
 8:         $\widetilde{q} \leftarrow \arg\max_{q_i \in Q} s_q(q_i)$
 9:         $\widetilde{s}_q \leftarrow s_q(\widetilde{q})$
10:         if sufficientQuality($R_{q,k}$, $\widetilde{s}_q$) then
11:             $\mathcal{H}$.move-to-front($\widetilde{q}$)
12:         else
13:             $R_{q,k} \leftarrow kNN_{\mathcal{D}}(q, k)$
14:             $\mathcal{H}$.put($q$, $Results$)
15:             $\mathcal{M}$.put($q$)
16:         end if
17:     end if
18:     return $R_{q,k}$
19: end procedure
```

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 k_c cached queries that are the closest to q (line 5). The resulting set of queries Q is first used to retrive all the associated result lists, and to extract from them only the k closest to the query q (line 7). Such objects form the approximated result list $R_{q,k}$. Then Q, is used to find an approximation $\widetilde{s}_q$ of the maximum safe radius and the corresponding query $\widetilde{q}$ (lines 8–9). 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 10). If this quality is not sufficient, then the query is forwarded to $\mathcal{D}$, and its (exact) results are added into the two indexing structures $\mathcal{H}$ and $\mathcal{M}$ (lines 13–15).

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. Also, if $\widetilde{q}$ was used to produce an good approximate result, then a move-to-front is applied to $\widetilde{q}$ (line 11), i.e. it is marked to be the most recently used, thus allowing useful queries to persist in cache.

It is worth noticing that different quality measures can be used to decide on the acceptance of an approximate result set. In our implementation, we chose empirically a very simple measure that accepts any answer set associated with a non trivial value of $\widetilde{s}_q$. As discussed above, unlike RCache, in QCache we can anticipate this quality check, by calculating $\widetilde{s}_q$ before constructing the result set. In this way, if $\widetilde{s}_q \leq 0$, we can avoid merging multiple answer sets (line 7), which requires a number of distance computations.

Speeding-up result construction. In principle, searching the objects that are the closest ones to q among k_c answer sets requires $k_c \times k$ distance computations. In order to avoid those expensive distance computations, we implemented a sort of pivot-based filtering technique [3], which uses the information about the distance of an object from its corresponding query to skip *a priori* some of the $k_c \times k$ candidates. In more detail, we take advantage of the information stored along with $kNN_{\mathcal{D}}(q_i, k)$, $\forall q_i \in Q$, where $Q = \mathcal{M}.kNN(q, k_c)$. The query object q_i can be seen as a pivot for all the objects in $x \in kNN_{\mathcal{D}}(q_i, k)$.

Let z be the k-th candidate approximate result found during this searching process. Because of the triangular inequality, given the result o_j of the query q_i, $|d(q_i, o_j) - d(q_i, q)| \geq d(q, z)$ implies $d(q, o_j) \geq d(q, z)$, i.e. o_j is not closer than z. Since $d(q_i, o_j)$ is stored in cache, and $d(q_i, q)$ was calculated at line 5, the candidate result z can be pruned without computing its distance from q.

3.4 Space and time cost modeling

In order to build an approximate result, QCache must be able to compute the similarity between every cached object $o \in \mathcal{C}$ and the submitted query object q. We denote with Sz the capacity of the cache, measured as the number of objects (i.e. the associated features) stored. Assuming that for each cached query object, only the top-k results are stored, the cache can contain the result sets of Sz/k queries. Also notice that some redundancy may occur whenever an object belongs to the result set of multiple cached queries.

Regarding the time complexity, this is affected by the approximate results construction phase. A cache hit costs $O(1)$ to retrieve the result set from the hash table $\mathcal{H}$, just as traditional caches for web search engines. In case of a miss, the metric index $\mathcal{M}$ is accessed with a cost proportional to its size, i.e. to the number of queries stored $O(Sz/k)$. In fact, the cost of searching with metric space based data structures grows linearly with the size of the collection indexed [21]. If the result quality is not considered sufficient, then also the image database must be accessed with a cost $O(|\mathcal{D}|)$, and the new result must be inserted into the cache by updating $\mathcal{M}$ with a cost of $O(Sz/k)$. Suppose that the hit rate ($Hit\%$), the approximate hit rate ($AHit\%$) and the miss rate ($Miss\%$) are known, then we can model the cost as follows:

$$\begin{aligned} Cost &= O(1) \cdot Hit\% \; + \; O(Sz/k) \cdot AHit\% \; + \\ & \quad (2 * O(Sz/k) + O(|\mathcal{D}|)) \cdot Miss\% \end{aligned} \tag{1}$$

In the experimental section we will show that even if the cost of an approximate hit is linear w.r.t. to the cache size, the improvement in performance is significant, since this cost is actually very small compared with the cost of accessing the underlying image database.

4. RESULTS ASSESSMENT

4.1 The data used

There are two important data sources we needed to set up for our experiments: the image database and the stream of queries.

The collection of images we used consists of a set of one million objects randomly selected from the CoPhIR collection[2]. CoPhIR is the largest publicly available collection of high-quality images metadata. Each contains five MPEG-7 visual descriptors (*Scalable Color, Color Structure, Color Layout, Edge Histogram, Homogeneous Texture*), and other textual information (title, tags, comments, etc.) of about 60 million photos (still increasing) that have been crawled from the Flickr photo-sharing site[3].

With regard to the query stream, although there exist several CBIR prototypes[4], there is no public query log available. Moreover, even if some of such prototypes offer an on-line demo, none of them aims to give a large-scale service to several users. Thus, even if their query logs were made public, they would not be representative of the actual use of a large-scale search-by-content service.

Thus, we generated a synthetic query log according to a web-based scenario, where most of the queries consist of images available on the web. It makes sense to assume that a query object provided by a user is an image found on a popular web-page, or one of the results obtained from the image search facility of a web-search engine.

First, we took into consideration the usage information made available by Flickr and stored in CoPhIR. In fact, for each image, we know the number of times it was seen by any internet user. Note that the views distribution follows a Zipf-like distribution (see Fig. 4), and it results to be similar to the query topic distribution present in the query logs of textual web search engines [7]. Thus, we assumed that

[2] CoPhIR stands for COntent-based Photo Image Retrieval, see `http://cophir.isti.cnr.it`.

[3] `http://www.flickr.com`

[4] See, for example, the list of CBIR systems in `http://en.wikipedia.org/wiki/Content-based\_image\_retrieval`

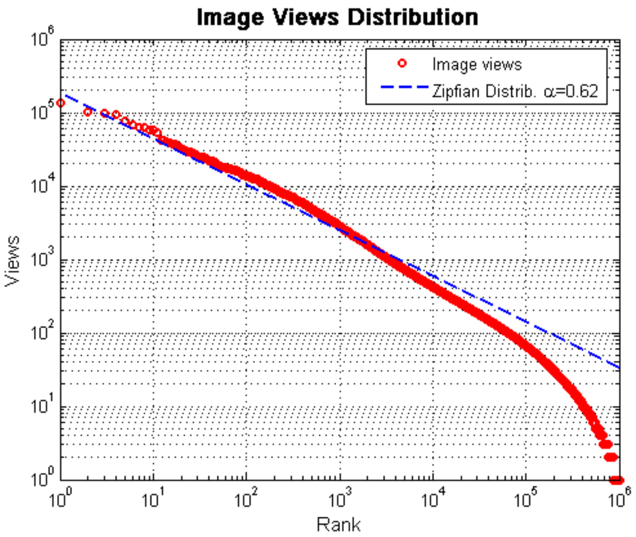


Figure 4: Number of times each Flickr image in our 1 million photo collection has been viewed.

the probability of an image to be used as a query object submitted to a web-scale CBIR system is proportional to the number of times it was viewed.

Second, we took into account the presence of near duplicates in web images. According to a human labeling experiment described in [11], (1) a large number of images in the web are duplicates; (2) these are generated via manipulations of the original image such as cropping, scaling, contrast adjustment; (3) the more an image is popular the larger is its duplication rate. Given the results of this study, we can estimate that about 8% of the images in the web are near-duplicates. Due to the setting of the experiment, only a subset of the images considered were labeled, and therefore, 8% is actually an underestimate.

Thus, to build an image query log, we took a different collection of 1 million images from which we sampled the queries. The use of another collection makes our algorithm less biased towards the presence in the query log of objects that exist in the database. We first injected a total of 8% of near duplicate images, by applying a duplication rate to each image proportional to its popularity, i.e. number of views. Also, we divided evenly the popularity of an image among the image itself and its duplicates.

Near-duplicate images were obtained from their original version by applying random scaling, or cropping, or contrast adjustment, or border addition, or a combination of cropping and contrast adjustment according to the presence of these alterations measured in [11]. Finally, we sampled with replacement 100,000 objects from such collection, where the probability of a photo to be selected is proportional to its popularity.

4.2 Experimental settings

We used a publicly available M-Tree [5] implementation[5] for indexing both the one million images dataset and the queries cached by QCache.

The dissimilarity (or distance) between two images was evaluated with a weighted sum of the distances between each of the five MPEG-7 descriptors used. The distance function applied to each MPEG-7 descriptor we used is the one proposed by the MPEG group [13, 17]. The distance between two images in our database is thus metric, according to our assumption.

As a rule for deciding whether or not the approximate results retrieved from the cache can be returned to the user, we checked whether the maximum safe radius defined in Corollary 1 is greater than zero. We will show in the following that this condition is able to guarantee a good quality of the results.

The main goal of our proposal is to answer approximate results whenever the exact results are not found in the cache. Therefore, the choice of quality measure for the approximated results is particularly important. Given a result set $R_{q,k}$ of k objects considered as the closest ones to query object q, we define the following error measures:

- Relative Error on the Sum of distances (RES)

$$RES(q, R_{q,k}) = \frac{\sum_{x \in R_{q,k}} d(q,x)}{\sum_{y \in kNN_{\mathcal{D}}(q,k))} d(q,y)} - 1$$

- Relative Error on the Maximal distance (REM)

$$REM(q, R_{q,k}) = \frac{\max_{x \in R_{q,k}} d(q,x)}{r_q} - 1$$

We conducted several tests for validating our proposal. To measure cache effectiveness we used the first 20,000 queries of the log as training-set to warm-up the cache, while we measured efficacy on the remaining test-set of 80,000 queries. Every query performed was a kNN with parameter $k = 20$.

Most of the tests were conducted by varying the size of the cache. We measure the size of the cache as the number of objects for which we need to store the associated features, as discussed in Section 3.4. Consider that the 100% of images' features must be indexed by the underlying database, while only a smaller $x\%$ of objects features are maintained by the cache, and only $\frac{x\%}{20}$ are actually managed by the metric index $\mathcal{M}$ of QCache, assuming that the top $k = 20$ results are stored for each cached query.

Finally, in our setting, the size of the features associated with an image is about 1KB, which corresponds to the size of the five MPEG-7 visual descriptors discussed above. Therefore, a cache of size 5% of our image collection stores about 50,000 objects, has a size of about 50MB, but indexes 2,500 queries only.

4.3 Cache hit-ratios

Fig. 5 reports exact, approximate, and total hit ratios as a function of the size of the cache. First of all, we note that the number of exact hits is not very high, meaning that a traditional cache would not be effective in a similar scenario. This is both due to the presence of duplicates, and to the α value of the Zipfian image views distribution which is smaller than in traditional text-search query logs.

Conversely, the approximate hit ratios achieved are remarkably higher. With the largest cache we experimented, it is possible to answer more than one fourth of the queries by storing only one tenth of the database. Since processing a similarity query over a metric index has a cost proportional to the size of the indexed collection [22], this result shows

[5] http://lsd.fi.muni.cz/trac/mtree/

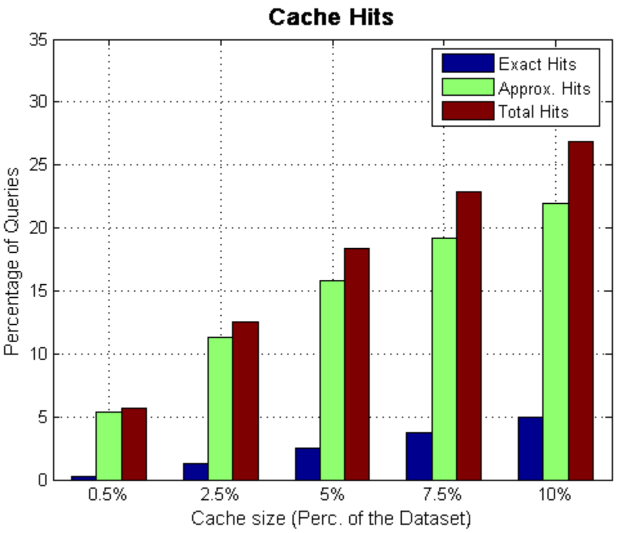


Figure 5: QCache exact and approximate hit ratios as a function of cache size.

that the proposed caching technique can have an impressive impact on the overall performance of a CBIR system. According to Eq. 1, a total number of hits of about 25%, reduces the average cost query answering by about the same amount. In addition, being the cache kept in the main memory, in case the CBIR system adopts a disk-resident index, the improvement in throughput would be much larger.

4.4 Approximation quality

In the following we will show also that the quality of the approximate answers returned by QCache is very good so that our metric cache can be considered effective also for approximating similarity queries on the basis on the reuse of results of past queries.

Fig. 6(a) plots the values of the RES and REM error measures. The plot reports RES and REM errors measured for the approximate results actually returned by the cache, but also for 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. Finally, the error of approximate answers is quite low, slightly above 5% for all cache sizes but the smallest one.

In Fig. 6(b) we report also the RES distribution in case of approximate hit. With a cache size of 5%, almost 90% of the returned results have an error smaller than 10%, while the number of results having an error greater than 20% is negligible.

In all the above experiments we used $k_c = 10$, that is we allowed the QCache algorithm to use only 10 result lists to build an approximate answer and to approximate the maximum safe radius. In Figure 7 we show the sensitivity of the algorithm to this parameter. In order to discover the exact maximum safe radius s_q, in principle all the cached queries should be checked. Our intuition was that the cached query object $\widehat{q}$ associated with s_q is likely to fall very close the user submitted query object q. Indeed, with $k_c = 1$ we have that the approximated radius $\widetilde{s}_q$ is very close to s_q ($\widetilde{s}_q/s_q > .94$), and that $\widetilde{s}_q$ is exactly equal to s_q almost 90% of the times (Figure 7(a)). While $\widetilde{s}_q$ is not strongly affected by k_c, the error in the answer set may degrade significantly for small values of k_c. This is because for smaller k_c we must use a smaller collection of objects to build the answer set. However, for $k_c \geq 5$ the error becomes stable and $k_c = 10$ seems to be an optimal trade-off (Figure 7(b)).

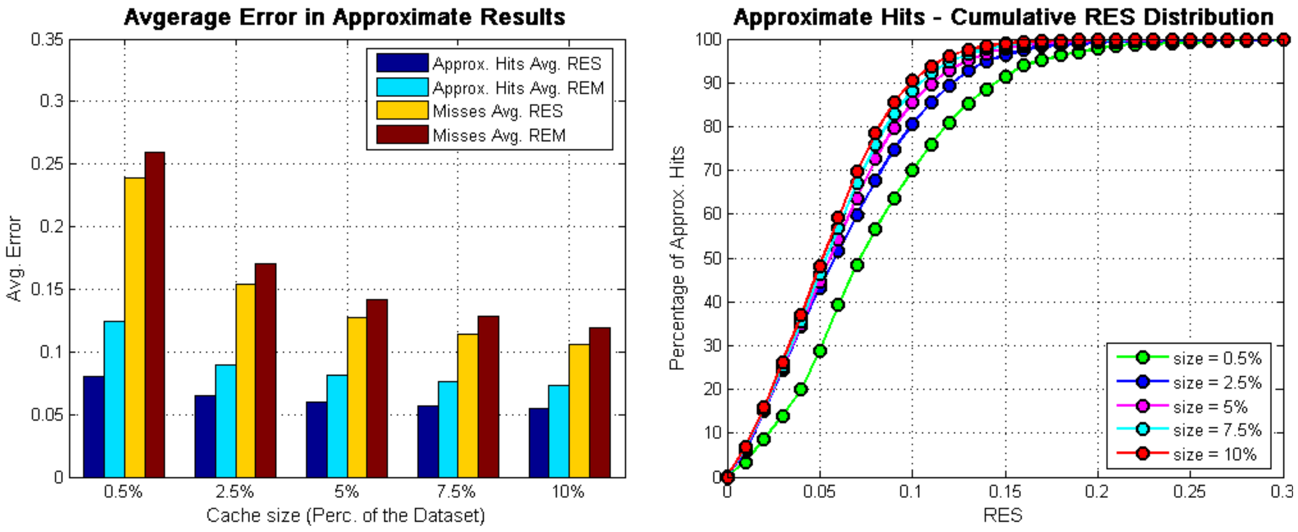
Finally, we show in Fig. 8 a comparison between the top-k most similar images returned by the QCache algorithm and the exact results of three different queries taken from our query log. The three queries are photos of a bug, a portrait and a sunset, whose approximate results provide an error of $RES = 0.07$, $RES = 0.09$ and $RES = 0.12$ respectively. For the sake of fairness, we are not showing the result sets with the smallest error, but the ones having an error close to the average error provided by the QCache algorithm, i.e. $RES = 0.055$. For the bug query, the top-4 approximate results are in the top ten exact results, meaning a recall of 40%. The portrait query has a smaller 30% recall with respect to the top ten exact results, even if the top-5 approximate results are probably perceived more interesting than the top-5 exact results. As expected from the high RES value, the recall of the sunset query is only 10%, but also in this case the approximate results provided are perceived to be almost as important as the exact ones.

In conclusion, we have shown that, for an error level slightly larger than the average error measured for the QCache algorithm, the approximate results provided have a good recall (objectively relevant), and they are also "visibly" similar to the query (subjectively relevant). Thus, we can claim that we succeeded in our goal of using past query results to construct new results sets relevant for the user.

5. CONCLUSIONS

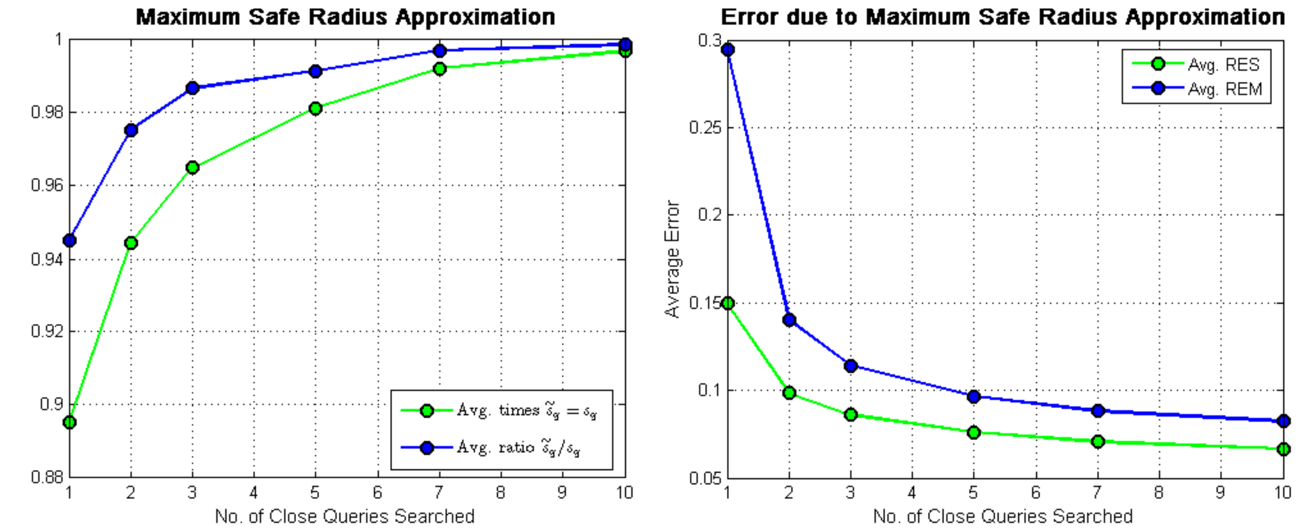
We have proposed the exploitation of a metric cache for improving throughput and response time of a large-scale CBIR system adopting the paradigm of similarity search in metric spaces. Unlike traditional caching systems, our proposal might return a result set also when the submitted query object was never seen in the past. In fact, the metric distance between the current and the cached objects is used to drive cache lookup, and to return a set of approximate results when some guarantee on their quality can be given. As a result, our caching algorithm is efficient and even robust with respect to the near-duplicate images which abundantly populate the Web and might be very commonly used to formulate popular content-based queries. The validity of the proposed approach was confirmed by several tests conducted on a collection of one million digital photos and a query log synthetically built on the basis of a popularity-based sampling, where the frequency of the occurrences of a photo in the query log is proportional to the number of times that photo was actually viewed in the flickr photo-sharing site from which it was taken.

We measured very interesting hit ratios. As an example, with a cache which is, in size, the 5% of the total size of the whole index, we obtained hit ratios slightly smaller than 20%. More importantly, the quality of approximated results returned by our caches are very good, and the largest contribution to the hit ratios due to approximate cache answers. This strongly motivate our proposal, and show that it is worth pursuing this research direction. The generality



(a) (b)

Figure 6: Average error (a), and error distribution (b) of approximate answers returned by QCache to the users.



(a) (b)

Figure 7: Sensitivity of the QCache algorithm to parameter k_c, i.e. the number of the closest cached query objects used by the algorithm to answer a query: effects of k_c (a) on the approximate computation of s_q, and (b) on the average error in computing the approximate result set.

of metric spaces that are the only assumption at the basis of our work, makes our contribution even more important as it can be applied at a large variety of scenarios.

6. ACKNOWLEDGMENTS

This work was partially supported by the SAPIR (Search In Audio Visual Content Using Peer-to-Peer IR) project, funded by the European Commission under IST FP6 (Contract no. 45128) and by the NeP4B (Networked Peers for Business) project, funded by the Italian Ministry of Research and high education on the Basic research funds (FIRB 2005).

7. REFERENCES

[1] 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.

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

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

[4] E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín. Searching in metric spaces. *ACM*

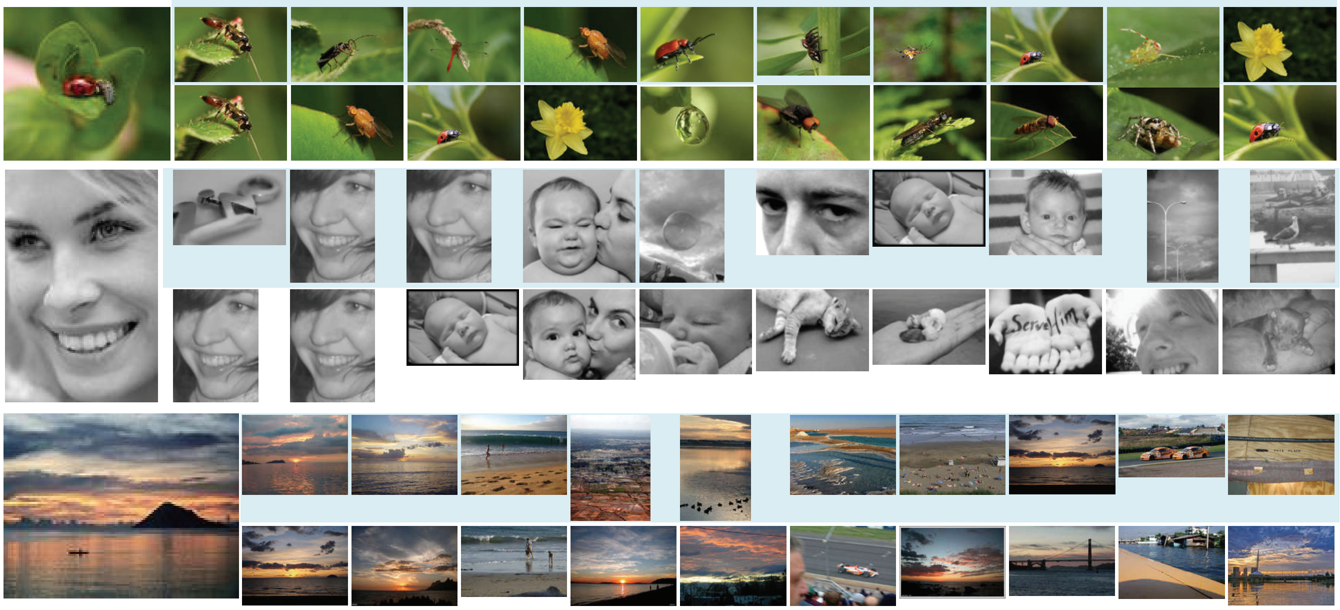


Figure 8: Three queries: bug ($RES = 0.07$), portrait ($RES = 0.09$) and sunset ($RES = 0.12$). For each query object we report the exact top-k results (top row) and the approximate ones returned by QCache (bottom row).

Computing Surveys (CSUR), 33(3):273–321, 2001.

[5] 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.

[6] R. Datta, D. Joshi, J. Li, and J. Z. Wang. Image retrieval: Ideas, influences, and trends of the new age. *ACM Computing Surveys*, 2007.

[7] 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.

[8] F. Falchi, C. Lucchese, S. Orlando, R. Perego, and F. Rabitti. A metric cache for similarity search. In *6th Workshop on Large-Scale Distributed Systems for Information Retrieval (LSDS-IR'08) - in conjunction with ACM CIKM'08*, October 2008.

[9] 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.

[10] J. J. Foo, J. Zobel, R. Sinha, and S. M. M. Tahaghoghi. Detection of near-duplicate images for web search. In *CIVR '07: Proceedings of the 6th ACM international conference on Image and video retrieval*, pages 557–564, New York, NY, USA, 2007. ACM.

[11] J. J. Foo, J. Zobel, R. Sinha, and S. M. M. Tahaghoghi. Detection of near-duplicate images for web search. In *CIVR '07: Proceedings of the 6th ACM international conference on Image and video retrieval*, pages 557–564, New York, NY, USA, 2007. ACM.

[12] G. R. Hjaltason and H. Samet. Index-driven similarity search in metric spaces (survey article). *ACM Trans. Database Syst.*, 28(4):517–580, 2003.

[13] ISO/IEC. Information technology - Multimedia content description interfaces. Part 6: Reference Software, 2003. 15938- 6:2003.

[14] 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.

[15] P. Lyman and H. R. Varian. How much information, 2003. retrieved from http://www.sims.berkeley.edu/how-much-info-2003.

[16] E. P. Markatos. On Caching Search Engine Query Results. *Computer Communications*, 24(2):137–143, 2001.

[17] P. Salembier and T. Sikora. *Introduction to MPEG-7: Multimedia Content Description Interface*. John Wiley & Sons, Inc., New York, NY, USA, 2002.

[18] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Computer Graphics and Geometric Modeling. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

[19] 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.

[20] R. van Zwol. Flickr: Who is looking? In *In proceedings of the 2007 IEEE / WIC / ACM International Conference on Web Intelligence (WI 2007)*, November 2007.

[21] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In A. Gupta, O. Shmueli, and J. Widom, editors, *VLDB'98, Proceedings of 24rd International Conference on Very*

Large Data Bases, August 24-27, 1998, New York City, New York, USA, pages 194–205. Morgan Kaufmann, 1998.

[22] 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.

[23] 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.

[24] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity SearchThe Metric Space Approach*, volume 32 of *Advances in Database Systems*. 233 Spring Street, New York, NY 10013, USA, 2006.