A Metric Cache for Similarity Search

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

Similarity search in metric spaces is a general paradigm that can be used in several application fields. It can also be effectively exploited in content-based image retrieval systems, which are shifting their target towards the Web-scale dimension. In this context, an important issue becomes the design of scalable solutions, which combine parallel and distributed architectures with caching at several levels.

To this end, we investigate the design of a similarity cache that works in metric spaces. It is able to answer with exact and approximate results: even when an exact match is not present in cache, our cache may return an approximate result set with quality guarantees. By conducting tests on a collection of one million high-quality digital photos, we show that the proposed caching techniques can have a significant impact on performance, like caching on text queries has been proved 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

Query-result caching, metric space, content-based retrieval.

1. INTRODUCTION

There are facts that are going to change our way to access the information on the Web. In the next three years, we will create more data than has been produced in all the human history, and most of them will be in multimedia form: images, video, music, etc. With the widespread use of digital cameras, more than 80 billion photographs are taken each year [9], and a significant part of them are published on the Web. Digital images already contribute for the largest part of Web content (about 24% in 2003 according to [9]), and their management promises to emerge as a major issue in the next years.

In this context, the interest in Content–Based Image Retrieval (CBIR) techniques is rapidly growing, even if used in conjunction with the usual methods for indexing and searching multimedia contents which currently exploit text and tags only. The final goal is to improve the precision perceived by users. A recent survey [4] reports on 56 CBIR systems, most of them exemplified by prototype implementations where the typical size of the indexed database is counted in thousands of images. However, to be able to be relevant with respect to the new challenges posed by the vast amount of images published on the Web, CBIR systems should be able to scale up their target, shifting from small scale, often highly specific, datasets to a much larger scale. Unfortunately, the CBIR process is inherently expensive, and the query processing cost grows very rapidly with the size of the collection indexed. Thus the only possibility to guarantee a scalable design of a CBIR system is to exploit a parallel and distributed architecture, with caching at several levels for reducing the average amount of computation / communication required to solve a similarity query.

In this paper, we tackle the scalability issues of future CBIR systems by investigating the possibility of retrieving the results of similarity-based queries from a cache posed in front of a CBIR system. Our aim is to reduce the average cost of query resolution, thus boosting the overall performance.

CBIR exploits the concept of *distance* between a query object and a collection of objects: similarity search can be seen as the process of retrieving from the indexed collection the most relevant objects, where the ranking criterion is given in terms of the distance from the query. Though this principle works for any distance measure, because of the mathematical foundations of metric distances, similarity search in metric spaces received increasing attention in the last decade [4, 17, 13, 3, 1, 2].

It is worth stressing the fact that the caching techniques proposed in this paper are general, and can be used 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).

LSDS-IR'08, October 30, 2008, Napa Valley, California, USA.

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 queries. Due to the peculiarities of the search paradigm, a cache-hit can be defined differently from traditional applications of caching in Web search engines. In addition to *exact* query matches, which can only occur when exactly the same query is submitted multiple times, also *approximate* ones may be looked-up from the query results stored in the cache. Note that, in principle, all the objects stored in the cache are relevant for a query, and their relevance measured by means of the distance function. We called our caching system *metric cache*, because we are able to exploit the metric property of the distance measure for evaluating the quality of the approximate results that our cache can guarantee. Obviously, the effectiveness of our metric cache cannot be evaluated in terms of the usual hit-ratio, but has instead to consider also the quality of the approximate results returned.

The rest of the paper is organized as follows. Section 2 discusses 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 analyzes the characteristics of typical queries adhering to our search-by-example paradigm, describes experimental settings, and reports on the promising results of the experiments conducted. Finally, Section 5 draws some conclusions and outlines future work.

2. RELATED WORK

The research topics more related to our work are query result caching in Web search engines, and techniques for approximate searching in metric spaces.

Query result caching. Caching is a very useful technique on the Web: it enables a shorter average response time, it reduces the workload on back-end servers, and utilized bandwidth [11].

Query logs constitute the most valuable source of information for evaluating the effectiveness of Web search engine caching systems storing the results of past queries. Many studies confirmed that users share the same query topics according to a Zipfian distribution [16, 14]. This high level of sharing justifies the adoption of a server-side 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 [10, 8, 5].

Unfortunately, similar analysis and results can not directly referred to the case of caching the results of content-based similarity search queries. In fact, due to the lack of large-scale search engines for this kind of queries, query logs of some statistical significance are not available. Thus, no study confirms the presence of the same level of locality in a stream of content-based queries coming from a multitude of users. We will address this problem, and we will make some pragmatic assumptions for characterizing user querying behavior. In particular 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).

On the other hand, a cache for the results of content-based similarity search queries is complex, and has the interesting opportunity of trying to exploit similarities between the current query and the objects stored in the cache with the aim of giving efficiently approximate answers and not barely of increasing the throughput of the retrieval process when exactly another occurrence of an already seen query is encountered.

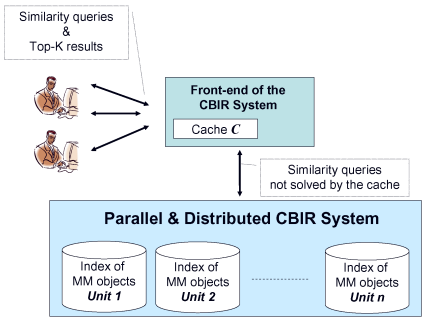

Figure 1: The architecture of our CBIR system, with the cache $\mathcal{C}$ placed in the front-end.

Approximate search in metric spaces. Due to the generally expensive cost of searching for similarity in metric spaces, approximate techniques have been studied to improve efficiency at the price of precision. As suggested in [6], approaches to approximate similarity search can be broadly classified into two categories: those which exploit *transformations of the metric space*, and approaches which *reduce the subset of data to be examined*. In the transformation approaches, approximation is achieved by changing the object representation and/or the distance function with the goal of reducing search cost. The second category of approaches are based on reducing the amount of data examined. To this aim, two basic strategies are used: *early termination*, and *relaxed branching*, 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 [17] and [13].

In this paper we will introduce a novel class of approximation technique, which is based on the reusing of results of previous submitted search queries. We will thus evaluate the efficacy of our approach by using measures proposed in the literature for evaluating approximate similarity search.

3. CACHING CBIR QUERY RESULTS

Let $\mathcal{C}$ be a cache placed in front of a CBIR system, storing the recently / frequently submitted queries and associated results (see Figure 1). The rationale of exploiting a cache $\mathcal{C}$ is that, when a hit occurs, we can avoid submitting the content-based query to 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. However, our cache $\mathcal{C}$ is very different from a traditional cache for a text retrieval system such as a Web Search Engine [8, 5], which, to some extent, 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.

We think that in content-based similarity search, even when the current query object was never seen in the past, the possible (and measurable) similarity among this object and the cache content can be exploited. In other words, we want $\mathcal{C}$ to be able to return an answer to as many submitted queries as possible: (a) an exact answer when exactly the same query object was submitted in the past, and its results were not evicted from the cache; (b) an approximate answer composed of the objects currently cached which are the closest to the current query, otherwise. In case the quality of the approximated answer is acceptable according to a given measure, the system can return it to the requesting user without querying the back-end.

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 [17, 15].

In this paper we present two different algorithms to manage the cache $\mathcal{C}$ and answer queries in case of both exact and approximate hits: RCache (Result Cache) and QCache (Query Cache). The two algorithms share the same theoretical background but differ deeply in their design.

3.1 Using the cache to answer approximate queries

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

The database $\mathcal{D}$ is queried by using two different kind of queries: *range queries* and *k nearest neighbors queries*. A range query returns all the object in the database at distance at most r from a given 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 returns the k nearest objects to query object q, denoted with $kNN_{\mathcal{D}}(q, k)$. We call r_q the radius of the smallest hyper-sphere 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 queries are interesting for many similarity search applications such as CBIR. Thus the $\mathcal{C}$ 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 database $\mathcal{D}$, while q_i may not.

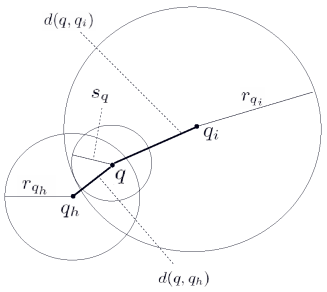

Figure 2: A query object q falling into two hyper-spheres containing $kNN(q_i, k)$ and $kNN(q_h, k)$.

Let us now suppose that a new query object q arrives. If $q \in \mathcal{C}$, then the results are known and they can be immediately returned to the user. Conversely, we are interested in understanding whether the objects cached in $\mathcal{C}$ 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 usual range and kNN queries on the collection of cached objects, the following theorem and corollary hold:

THEOREM 1. *Given a new incoming query object q, and a previously cached 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 the new query q w.r.t. the cached query 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 to the query object q, where the top k' objects of the approximate answer are *the same* of the top k' results of the exact answer.

Figure 2 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 hyper-sphere 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.

3.2 RCache

The RCache (Result Cache) algorithm is sketched in Algorithm 1. It makes use of a hash table $\mathcal{H}$ which is used to store and retrieve efficiently queries and their results 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 their kNN results are already stored in the cache (line 2–3). 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 currently stored in the cache (line 5).

Along with each returned object $o \in \mathcal{M}.kNN(q, k)$, $\mathcal{M}$ stores Q_o, the references to all the cached queries q_i that returned o among the results. In order to calculate the maximum safe radius s_q, it is not needed to consider all the cached queries. In fact, given the cached object nearest to the query object $x = kNN_{\mathcal{C}}(q, 1)$, it is sufficient to take into account only the queries Q_x:

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

By computing s_q over Q_x (lines 6-8), and looking at the k results returned from the cache metric index $\mathcal{M}$, we can evaluate their distances from q and determine how many of them are safe. In other words, 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 the same results that would have been retrieved from the whole database $\mathcal{D}$.

The computed values k' and s_q can thus be used to evaluate the quality of the approximate results returned by the cache $\mathcal{C}$ (line 9), and thus deciding whether it is necessary to actually answer the query from $\mathcal{D}$ or not. When the approximation is considered satisfying, the set $\mathcal{M}.kNN(q, k)$ is returned as query answer (line 17), otherwise, the exact $kNN_{\mathcal{D}}(q, k)$ is computed (lines 12) and returned (line 17), while the pair $(q, kNN_{\mathcal{D}}(q, k))$ is inserted in $\mathcal{C}$. To this end:

- The pair $(q, kNN_{\mathcal{D}}(q, k))$ is added to the hash table $\mathcal{H}$, where query object q is used as hash key (line 13);
- Each object $x \in kNN_{\mathcal{D}}(q, k)$ is added to $\mathcal{M}$ if not already present. In case x is already in $\mathcal{M}$, query object q is added to the list of queries Q_x associated with x (line 14).

Whenever, during the insertion of an object in $\mathcal{M}$ its size limit is reached, according to a simple LRU policy, the last recently used pair $\langle q_i, kNN_{\mathcal{D}}(q, k)\rangle$ is evicted from $\mathcal{H}$, and the references to q_i are removed from every object $x \in kNN_{\mathcal{D}}(q_i, k)$ stored in $\mathcal{C}$ and indexed in $\mathcal{M}$. As a consequence of this removal, Q_x may become empty: in this case x itself is removed from $\mathcal{M}$. Also, if s_q was used to produce a good approximate result, then a move-to-front is applied to q (line 10), i.e. it is marked to be the most recently used, thus allowing useful queries to persist in cache.

Algorithm 1 Algorithm RCache

```
1: procedure LookUp(q, k)
2:     if q ∈ H then
3:         R_{q,k} ← H.get(q)
4:     else
5:         R_{q,k} ← M.kNN(q, k)
6:         x ← R_{q,k}.top
7:         Q_x ← x.queries
8:         s_q ← max_{q_i ∈ Q_x}(0, r_{q_i} − d(q_i, q))
9:         if sufficientQuality(R_{q,k}, s_q) then
10:            H.move-to-front(q)
11:        else
12:            R_{q,k} ← kNN_D(q, k)
13:            H.put(q, Results)
14:            M.put(q)
15:        end if
16:    end if
17:    return R_{q,k}
18: end procedure
```

3.3 QCache

In this section we describe QCache, a different algorithm that tries to reduce the cost of kNN searches among all the objects currently stored in the cache. Rather than indexing every single object in the result sets of the cached queries as Algorithm RCache does, QCache builds a metric index $\mathcal{M}$ only over the query objects, thus reducing the number of indexed objects by a factor of k. The idea is to search a set of suitable queries among the cached ones, and to use their neighbors to produce an approximate answer.

Supposing that the maximum safe radius s_q is not trivial, i.e. $s_q > 0$, we denote with $\widehat{q}$, the query object associated with it:

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

Given the cached query object $\widehat{q}$, it holds that set $kNN_{\mathcal{D}}(\widehat{q}, k)$, which is also cached, contains the top k' neighbors of object q in $\mathcal{D}$. In other words, once we found $\widehat{q}$, we determined the k' cached objects that are guaranteed to be the top k' nearest neighbors of q.

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

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 object q_i nearest to q.

This strategy may not find $\widehat{q}$. As an example, consider Figure 2, which shows that, in order to determine the largest safe radius s_q, we may need to consider a cached query object (q_i) that is not the closest one (q_h) to q.

In order to improve this approximation, we will pick the k_c cached queries closest 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 $kNN_{\mathcal{D}}(\widetilde{q}, k)$ contains k', $k' \leq k$, of the nearest neighbors of q in the database $\mathcal{D}$. These are the objects $o \in kNN_{\mathcal{D}}(\widetilde{q}, k)$ being 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 2 Algorithm QCache

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

In Algorithm 2 we show the pseudo-code of our QCache algorithm. 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.

It uses the metric index $\mathcal{M}$, which contains only the recent past queries $q_i \in \mathcal{C}$, 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 retrieve all the associated result lists, and to extract from them only the k objects closest to q (line 7).

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).

Also in this case, $\mathcal{H}$ is managed with a simple LRU policy.

Speeding-up the 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, 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.

3.4 Space and time cost modeling

In order to build an approximate result, RCache and QCache must be able to compute the similarity between every cached object $o \in \mathcal{C}$ and the submitted query object q. To this end, the cache must store not only the *identifiers* of the query results, but also the *features* associated with these objects. Such features are needed by the distance function d in order to evaluate the dissimilarity of two objects. We denote with Sz the capacity of the cache, measured as the number of objects (i.e., the associates features) stored. Note that, in the case of the QCache algorithm, result lists are stored independently, and therefore, 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 results 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, that is the number of indexed objects $O(Sz)$ for RCache and the number of indexed queries $O(Sz/k)$ for QCache. In fact, the cost of searching with metric space based data structures grows linearly with the size of the collection indexed. In [15] it is shown that, under the uniform i.i.d. assumption, the probability of visiting a leaf of the index is 100% when having 1 million objects and 45 dimensions, meaning that every object has to be compared with the query. In our setting, the five descriptors exceeds 200 dimensions, and it is therefore reasonable to assume a linear cost w.r.t. the database size.

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 $\mathcal{M}$, incurring in the same cost of a search operation. 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:

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

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

The collection we used in our experiments consists in a set of one million objects randomly selected from CoPhIR (`http://cophir.isti.cnr.it`), the largest publicly available collection of high-quality images meta-data. 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.) of about 54 million photos (still increasing) that have been crawled from Flickr (`http://www.flickr.com`).

However, the main problem we have to face with in order to assess the effectiveness of our CBIR caching strategy, is the lack of actual CBIR systems usage information. In fact, although there are several CBIR prototypes available (See, for example, the list of CBIR systems in `http://en.wikipedia.org/wiki/Content-based\_image\_retrieval`), some of which offering also an on-line demo, none of them aims to give a large-scale service to several users. Thus, even if their query logs would be publicly available, they would not be representative of the actual use of a large-scale, search-by-content service.

Therefore we generated a synthetic query log using 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, which is a measure of its popularity with respect to the other images in the collection. Note that the views distribution follows a Zipfian curve, as shown in Figure 3(g), which results to be very similar to the query distribution present in the query logs of textual web search engines [5].

Thus, we assumed that the the probability of an image to be used as a query for a Web–scale CBIR system is due to its popularity, and we synthesized a content–based query log by exploiting a sampling with replacement of $100,000$ objects from our one million images dataset, where the probability of a photo to be selected is proportional to the number of times that the photo has been viewed on Flickr.

4.2 Experimental settings

We used a publicly available M-Tree implementation (available at `http://lsd.fi.muni.cz/trac/mtree`) for indexing the one million images dataset and the cache content of both QCache and RCache algorithms.

The dissimilarity (or distance) between two images has been evaluated with weighted sum of the distances between each of the five MPEG-7 descriptors used. The distance function for each MPEG-7 descriptor we used is the one proposed by the MPEG group [7, 12]. The distance between two images in our database is thus metric, according to our metric space assumption. As a rule for deciding whether or not the approximate results found can be returned to the user, we checked whether at least the first approximate result is guaranteed to be correct according to Theorem 1. We will show in the following that this condition is able to guarantee a good quality of the results. Since the main goal of our proposal is answering approximate results whenever the exact results are not stored in the cache, particularly important is instead the choice of the measures of quality of approximated results. Given a result set $R_{q,k}$ of k objects ordered by their proximity to a query object q, we used *Relative Error on the Sum of distances* (RES)

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

and *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. Most of the tests were conducted by varying the size of the cache, measured as described in Section 3.4. Consider that for each image we need 1KB to store its five visual descriptors. Therefore, a cache of size 5% is about 50MB large. Finally, all the reported experimental results were obtained for $k = 20$.

4.3 Cache hit-ratios

Figures 3(a,d) report exact, approximate, and total hit ratios as a function of the size of the cache. On the basis of these figures we can do some important considerations. First of all, we note that the proposed techniques exceed the threshold of 20% hits by exploiting a cache which store the features of only the 5% of the dataset. Since processing a similarity query over a metric index has a cost directly proportional to the size of the indexed collection [17, 15], this result shows that the proposed caching techniques can have an impressive impact on the overall performance of a CBIR system. 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 even larger. To this regard we wont remember that our content-based query log was constructed according to the method described above, and that the hit ratios figures achieved can be considered only indicative of the actual performance achievable with our techniques.

More importantly, the contribution to the total hit ratio achieved can be almost equally subdivided between exact and approximate cache answers. This strongly motivates our proposal for a metric cache which does not simply return the exact matches. In the following we will show also that the quality of the approximate answers returned is very good, so that our metric cache can be considered effective also for approximating similarity queries on the basis of the reuse of past knowledge. Regarding the comparison of the hit ratios achieved by the two solutions, from the plots we can see that, as expected, RCache slightly outperforms QCache. However, the difference in performance is quite small and becomes more remarkable only when the cache size is increased. This trend particularly affects the number of exact matches achieved, and has a very simple motivation. In fact, as above mentioned, at parity of size, RCache can store more objects than QCache due to the absence of duplicated objects.

4.4 Approximation quality

Figures 3(b,e) plot the values of the RES and REM error measures, as a function of the size of the cache. The two Figures refer to the two algorithms RCache and QCache, respectively. The plots report RES and REM errors measured

for the approximate results actually returned by the cache, but also for the cases in which instead the cache returned a miss since the expected quality of approximation was not considered good enough. As it can be seen, the error measured for the misses is always much larger than the one measured for the hit cases. Moreover, its absolute value in the last case is very low, well under the 10% in all cases but one. It is worth noting that the increase of the QCache size does not have a significant impact on the quality of the approximated results returned (i.e., *Approx. Hits*). We can in fact see that the error values for *Approx. Hits*, reported in Figure 3(b), does not decrease as for RCache in Figure 3(e). In fact, in the QCache algorithm the approximate results are obtained by searching between the results of the cached k_c queries closest to the newly submitted one (all the graphs were obtained for $k_c = 10$). Thus, even if the size is incremented, the cardinality of the set of results among which searching for approximate results, does not increase.

With cache RCache instead, the error measured in approximate results decreases while the cache size is increased (see Figure 3(e)). In fact, in this case all the results of cached queries are stored in the metric index which is used for retrieving the approximate results. Thus, RCache can obtain better approximate results by increasing cache size. Unfortunately, this higher quality of RCache is paid with a higher cost of the searching operation. While RCache indexes $x\%$ objects, QCache needs to search only among $x\%/k$ queries, resulting in a speed-up of factor k.

In Figures 3(c,f) we report the distribution of RES for various cache sizes obtained from the cache for RCache and QCache, respectively. The results reported are for the approximate hits. These figures basically show that the number of bad approximations is negligible. As discussed above, our metric caches return an Approximate Hit whenever the closest result is guaranteed to be correct. However, $k' \geq 1$ results can actually be correct in the approximate result set returned. In Figure 3(h) we show the distribution of the k' correct results, which turn out to be within the approximate result set answered by algorithm QCache. We can see from the graph that in a significant number of cases $k' \geq 1$. For example, in the 20% of times the cache returned an approximate answer, this resulted to contain at least the top-6 correct nearest neighbor results.

Finally, in Figure 3(i), we report an evaluation of the impact of caching on the whole CBIR system on varying cache size. Such costs were evaluated as described in Section 3.4 and using the actual hit/miss ratios measured in the above experiments. It is possible to see that, with a cache size of 5%, which results in a cache hit of 20% (see Figure 3(a)), the QCache algorithm reduces the cost of a query by 20% on average. As expected, the cost of the RCache algorithm is larger.

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. Two different caching strategies have been designed, and their simple theoretical background detailed in the paper. Differently from traditional caching systems, both our proposals RCache and QCache might return a result set also when the submitted query was never seen in the past. In fact, cached objects are considered important knowledge, and when some conditions occur, the results of past queries are used by both algorithms to approximate the results returned as query answer. The two algorithms differ mainly in the kind of objects inserted in their metric index: result objects for RCache, queries for QCache. Thus the metric index exploited by RCache is larger than the one used by QCache, as the expected cache look-up time. On the other hand, operations performed on QCache are less expensive, but both the quality and the opportunities for approximate results slightly worse.

The expected behavior of the two approaches 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 hit ratios of up to 30%. As an example, with a cache which is, in size, the 5% of the total size of the dataset, we exceeded the 20% hit ratio threshold with both algorithms. More importantly, the quality of approximated results returned by our caches are very good, and the contribution to the hit ratios almost equally subdivided between exact and approximate cache answers. This strongly motivates our proposal, and shows that it is worth pursuing this research direction. The generality 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. Future work will regard more extensive experimentation on the large-scale CBIR system developed within the SAPIR project which currently index a collection of 50 million images.

Acknowledgments
Work partially supported by the IST FP6 Project SAPIR (Search In Audio Visual Content Using Peer-to-Peer IR), contract no. 45128.

6. 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. Surveys*, 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] E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín. Searching in metric spaces. *ACM Comp. Surveys*, 33(3):273–321, 2001.

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

[5] 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.

[6] H. Ferhatosmanoglu, E. Tuncel, D. Agrawal, and A. El Abbadi. Approximate nearest neighbor searching in multimedia databases. In *Proc. of 17th ICDE*, 2001.

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

[8] R. Lempel and S. Moran. Predictive caching and prefetching of query results in search engines. In *Proc. of the 12th WWW Conference*, pages 19–28, New York, NY, USA, 2003. ACM Press.

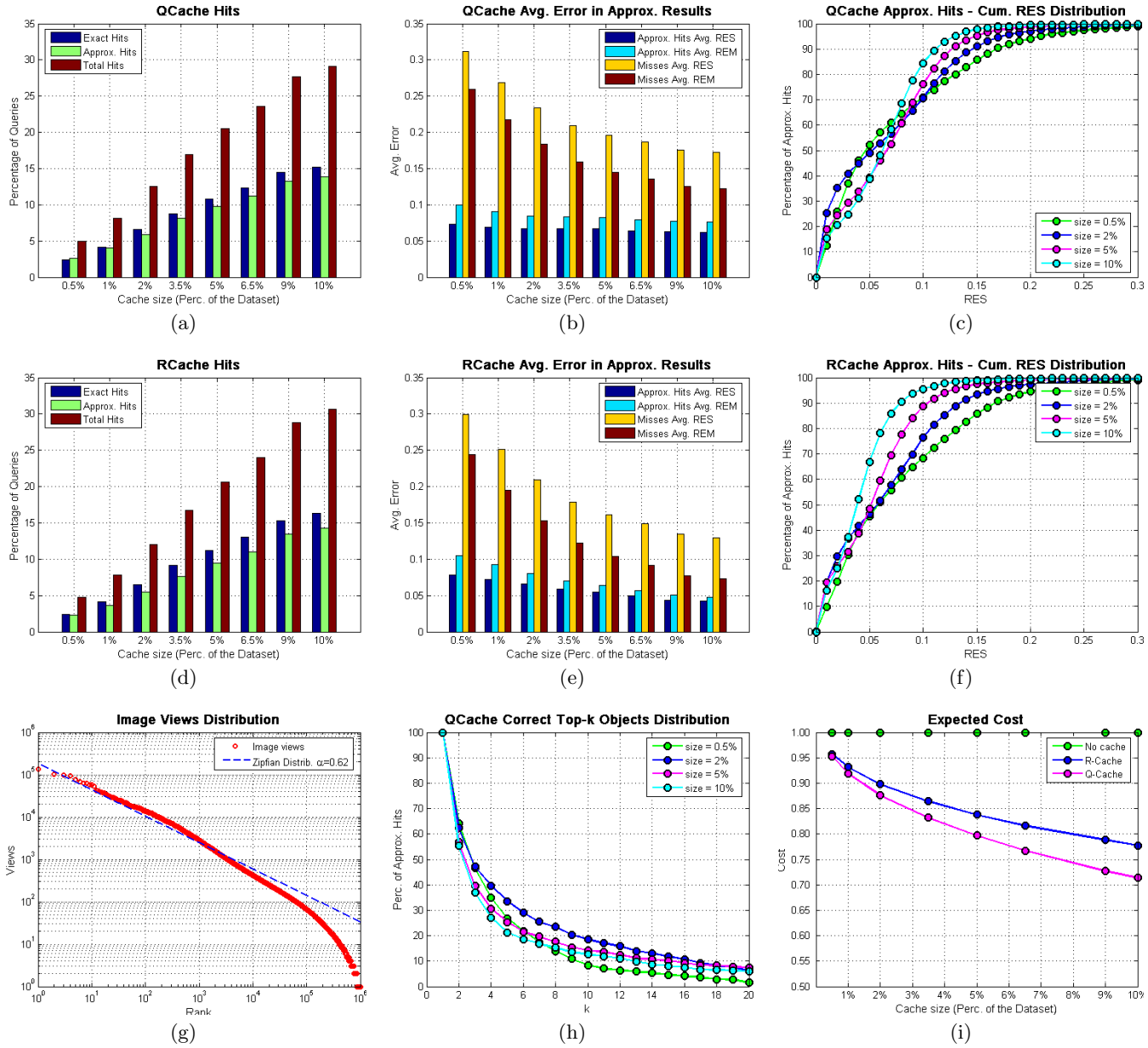

Figure 3: Performance on varying cache size: QCache **(a) and** RCache **(d) exact and approximate hit ratios,** QCache **(b) and** RCache **(e) average error in approximate results,** QCache **(e) and** RCache **(f) error distribution on approximate results returned to the user. (g) Number of times each Flickr image in our 1 million photo collection has been viewed. (h)** QCache **distribution of Top-k exact results found in the approximate results. (i) Qualitative evaluation of caching algorithms performance.**

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

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

[11] S. Podlipnig and L. Boszormenyi. A survey of web cache replacement strategies. *ACM Comput. Surv.*, 35(4):374–398, 2003.

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

[13] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Computer Graphics and Geometric Modeling. Morgan Kaufmann Pub., CA, USA, 2006.

[14] 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.

[15] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proc. of 24th VLDB*, pages 194–205, 1998.

[16] Y. Xie and D. O'Hallaron. Locality in search engine queries and its implications for caching. In *Proceedings of 21st IEEE INFOCOM*, 2002.

[17] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity SearchThe Metric Space Approach*, volume 32 of *Advances in Database Systems*. NY, USA, 2006.