Caching Algorithms for Similarity Search

(Extended Abstract)

Fabrizio Falchi[1], Claudio Lucchese[1], Salvatore Orlando[2],
Raffaele Perego[1], and Fausto Rabitti[1]

[1] ISTI-CNR, Pisa, Italy
[2] Università Ca' Foscari, Venezia, Italy

Abstract. Similarity search in metric spaces is a general paradigm that can be used in several application fields. One of them is content-based image retrieval systems. In order to become an effective complement to traditional Web-scale text-based image retrieval solutions, content-based image retrieval must be efficient and scalable. In this paper we investigate 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 allows the cache to return approximate answers with acceptable quality guarantee even if the query processed has never been encountered in the past. 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. Moreover, we show that our caching algorithm does not suffer of cache pollution problems due to near-duplicate query objects.

1 Introduction

With the widespread use of digital cameras, more than 80 billion photographs are taken each year [1], and a significant part of them, over one billion[3], are published on the Web. In this context, the interest in searching such huge collections of images by their content is rapidly growing [2]. 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.

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. Being a very general and well studied paradigm, we based our cache on the mathematical foundations of *metric spaces* [3]. The cache we propose is very different from a traditional cache for text-based Web Search Engines (WSEs). 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

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

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. Our cache 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 algorithms and experimental results presented in this paper have been previously published in [4] while the scenario where near-duplicate images are submitted as query objects by the users of a large-scale CBIR system has been studied in more details in [5].

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 [6].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 [7].

Approximate search in metric spaces. As suggested in [8], 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*. Our proposal introduces a novel class of approximation techniques, based on reusing the results of previously submitted queries.

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. 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.

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 $R_{\mathcal{D}}(q, r)$ returns all the objects 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, $kNN_{\mathcal{D}}(q, k)$, 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 queries are more interesting in similarity search applications such as CBIR. 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 (proofs have been given in [4]):

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

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.

In a preliminary work [4] 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, in [5] we also evaluated 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. In this paper we only sketch the algorithms, and focus the attention on discussing and evaluating them reporting the results presented in [4].

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.

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, w.r.t. RCache, 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.

In both RCache and QCache whenever during the insertion of an object in the cache the size limit is reached data is replaced according to a simple LRU policy. RCache and QCache have been described in more details in both [4].

4 Results Assessment

The collection we used in our experiments consists of 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, and other textual information of about 100 million photos that have been crawled from Flickr (`http://www.flickr.com`). 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. 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.

First, we took into consideration the usage information made available by Flickr and stored in CoPhIR. We assumed that 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 [9], we can estimate that about 8% of the images in the web are near-duplicates. Thus, we took a different collection of 1 million images to make our algorithm less biased towards the presence in the query log of objects that exist in the database. and we 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 [9]. 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.

We used a publicly available M-Tree implementation[4] 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 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. 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)

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

[4] available at `http://lsd.fi.muni.cz/trac/mtree`

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. 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 $k = 20$.

4.1 Cache hit-ratios

Figures 1(a,d,g) report exact, approximate, and total hit ratios as a function of the size of the cache. 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 [3], this result shows that the proposed caching techniques can have an impressive impact on the overall performance of a CBIR system. 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. 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.

Regarding the query log with near-duplicate queries 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. 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.

4.2 Approximation quality

Figures 1(b,e,h) plot the values of the RES and REM error measures, as a function of the size of the cache. 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*). 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 $k_c = 10$). With cache RCache instead, the error measured in approximate results decreases while the cache size is increased (see Figure 1(e)). 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. Figure 1(h) shows that our QCache is robust to near-duplicates.

In Figures 1(c,f,i) we report the distribution of RES for various cache sizes. 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.

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. 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. Our caching algorithm is 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 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.

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

References

1. Lyman, P., Varian, H.R.: How much information (2003) retrieved from http://www.sims.berkeley.edu/how-much-info-2003.
2. Datta, R., Joshi, D., Li, J., Wang, J.Z.: Image retrieval: Ideas, influences, and trends of the new age. ACM Computing Surveys (2007)
3. Zezula, P., Amato, G., Dohnal, V., Batko, M.: Similarity SearchThe Metric Space Approach. Volume 32 of Advances in Database Systems., 233 Spring Street, New York, NY 10013, USA (2006)
4. Falchi, F., Lucchese, C., Orlando, S., Perego, R., Rabitti, F.: 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)
5. Falchi, F., Lucchese, C., Orlando, S., Perego, R., Rabitti, F.: Caching content-based queries for robust and efficient image retrieval. In: Proceedings of EDBT 09: the twelfth International Conference on Extending Database Technology. (March 2009)

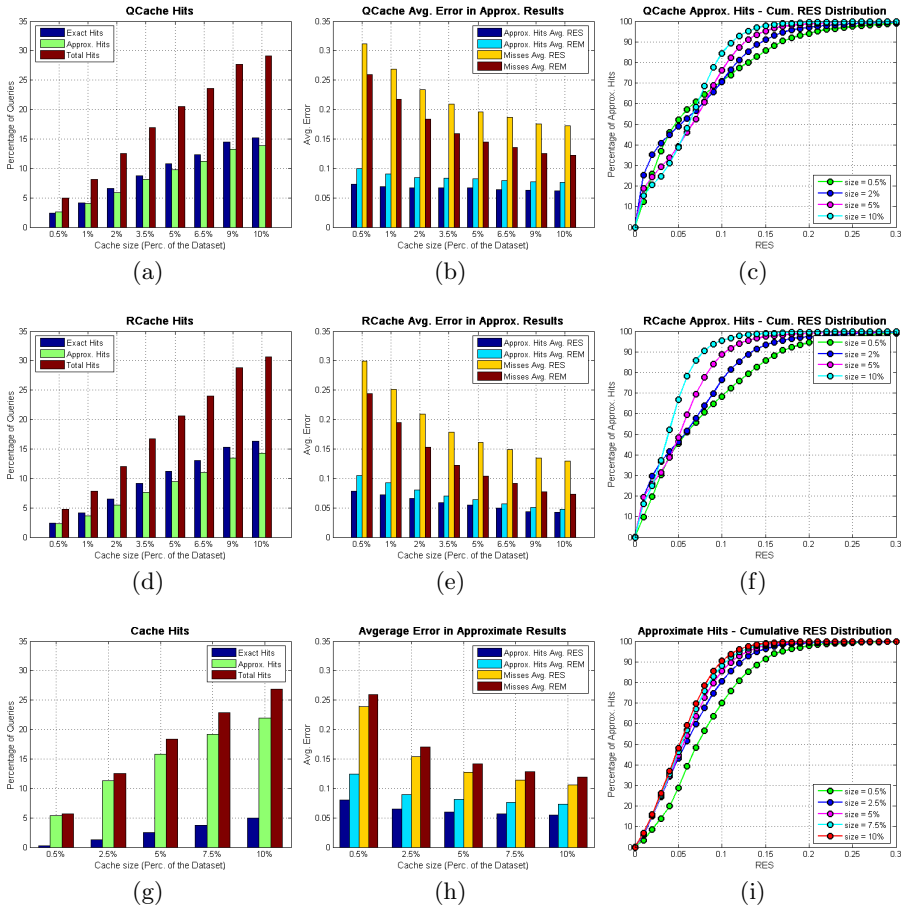

Fig. 1. QCache (a) and RCache (d) exact and approximate hit ratios, QCache (b) and RCache (e) average error in approximate results, QCache (c) and RCache (f) error distribution on approximate results returned to the user. Exact and approximate hit ratios (g), average error (h), and error distribution (i) of approximate answers returned by QCache to the users using the query log with near-duplicate query.

6. Xie, Y., O'Hallaron, D.: Locality in search engine queries and its implications for caching. In: Proceedings of IEEE INFOCOM 2002, The 21^{st} Annual Joint Conference of the IEEE Computer and Communications Societies. (2002)
7. Markatos, E.P.: On Caching Search Engine Query Results. Computer Communications **24**(2) (2001) 137–143
8. Ferhatosmanoglu, H., Tuncel, E., Agrawal, D., El Abbadi, A.: Approximate nearest neighbor searching in multimedia databases. In: Data Engineering, 2001. Proceedings. 17th International Conference on. (2001)
9. Foo, J.J., Zobel, J., Sinha, R., Tahaghoghi, S.M.M.: Detection of near-duplicate images for web search. In: CIVR '07: Proceedings of the 6th ACM international conference on Image and video retrieval, New York, NY, USA, ACM (2007) 557–564