MiPai: using the PP-Index to build an efficient and scalable similarity search system

Andrea Esuli
Istituto di Scienza e Tecnologie dell'Informazione
Consiglio Nazionale delle Ricerche - Italy
andrea.esuli@isti.cnr.it

Abstract—**MiPai is an image search system that provides visual similarity search and text-based search functionalities. The similarity search functionality is implemented by means of the Permutation Prefix Index (PP-Index), a novel data structure for approximate similarity search. The text-based search functionality is based on a traditional inverted list index data structure. MiPai also provides a combined visual similarity/text search function.**

I. Introduction

MiPai[1] is an image search system that provides visual similarity search and text-based search functionalities on the CoPhIR collection [1], consisting of 106 million images crawled from the Flickr photo sharing website. Each image is represented by five MPEG-7 visual descriptors, used to perform visual the similarity search, and it has structured textual content associated (e.g., title, description, tags, comments), used by the text search component.

This paper details the architecture of MiPai, its search functions and Web interface, and also introduces its core data structure, the PP-Index, a novel data structure that allows to build efficient and scalable similarity search services.

II. The PP-Index

The Permutation Prefix Index (PP-Index), is an approximate similarity search data structure that belongs to the family of the *permutation-based indexes* (PBI), independently introduced by Amato and Savino [2], and Chavez et al. [3].

The intuition behind PBIs is that two similar objects have a *similar view of the surrounding world*, i.e., they are likely to see the elements of a set of *reference objects* in the same order of distance, according to a given distance function that models the concept of similarity.

More formally, given a set of objects D, from a domain $\mathcal{O}$, and a distance function $d : \mathcal{O} \times \mathcal{O} \rightarrow \mathbb{R}^+$, the PP-Index is built by computing a *permutation prefix* Π^l_o for each $o \in D$. Π^l_o is the sequence of the identifiers of the l closest reference objects to o with respect to d, determined on a set of reference objects $R = \{r_0, \dots, r_{|R|-1}\} \subset \mathcal{O}$. In our case, R is actually composed by randomly selected elements of D.

A *prefix tree* T, kept in main memory, stores all the permutation prefixes for objects in D. A *data storage* file, kept of disk, contains the relevant data for each object (i.e., its identifier and the information required to compute its distance to any object in $\mathcal{O}$), sequentially written following the order resulting from an ordered visit of T.

The key difference between the PP-Index and the other PBI methods, is that [2] and [3] use the permutation space as a transformed similarity space, used to estimate the distance order between objects and the query, while the PP-Index uses permutation prefixes for a very fast determination of a small *set of candidate objects*, from which the best elements to be included in the result are selected using the original distance function.

Specifically, given a k-NN query for an object $q \in \mathcal{O}$, Π^l_q is computed and it is used to search for the longest prefix match in T whose subtree points to at least z *candidate objects*. The k-NN result is computed on such subset of $z' > z$ candidate objects. The key advantage of this method is that the z' objects are all stored in adjacent positions of the data storage file, thus allowing extremely efficient data access, typically resulting in a single read operation.

Various optimizations can be applied to the prefix tree in order to reduce its memory occupation [4]. A specific optimization for the PP-Index, applicable when the z is fixed, consists of reducing all the subtrees that point to less than z objects to a single leaf node, given that none of such subtrees will be ever selected by the search function. This is a crucial optimization that allows to drastically reduce the memory occupation of the prefix tree T, allowing scalability to very large collections and/or to keep multiple index instances loaded at the same time on a single computer. For example, for an index built using the first 100 million images of the CoPhIR collection, the optimization of the prefix tree for a value $z = 1000$ reduces its memory occupation from 354.5 MB to just 6.5 MB.

The above described search process is very efficient. Processing a 100-NN query on a PP-Index built on the entire CoPhIR collection has an average time cost of 0.239 seconds, measured on a test set of 100 randomly selected images from CoPhIR, held out from indexed data. The resulting effectiveness is relatively low, with an average 18.3% *recall* and 8.1% *relative distance error*[2] (RDE) [5].

[1]http://mipai.esuli.it/

[2]$Recall(k) = \frac{|D^k_q \cap P^k_q|}{k}$ $\quad RDE(k) = \frac{1}{k}\sum_{i=1}^{k} \frac{d(q,P^k_q(i))}{d(q,D^k_q(i))} - 1$
where D_q is the list of the elements of D sorted by their distance $d(,)$ with respect to q, D^k_q is the list of the k closest elements, P^k_q is the list returned by the algorithm, and $L^k_q(i)$ returns the i-th element of the list L.

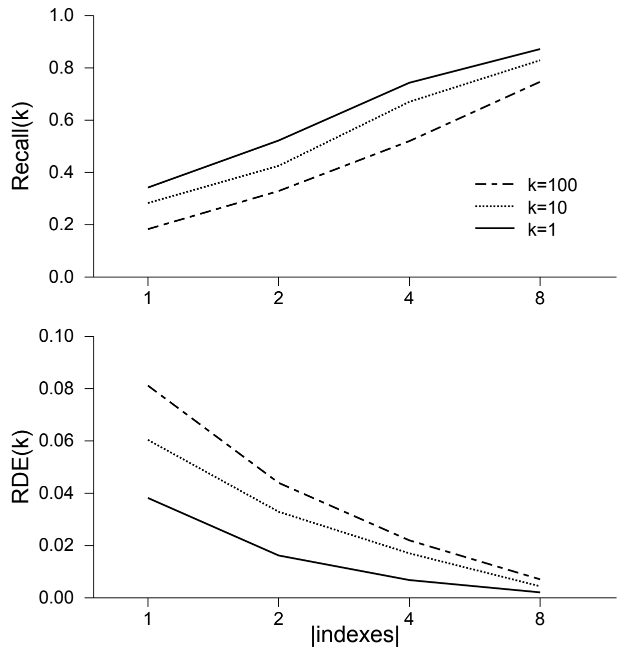


Fig. 1. Multiple index search strategy on the 100M index, using $|R| = 1,000$ and $z = 1,000$.

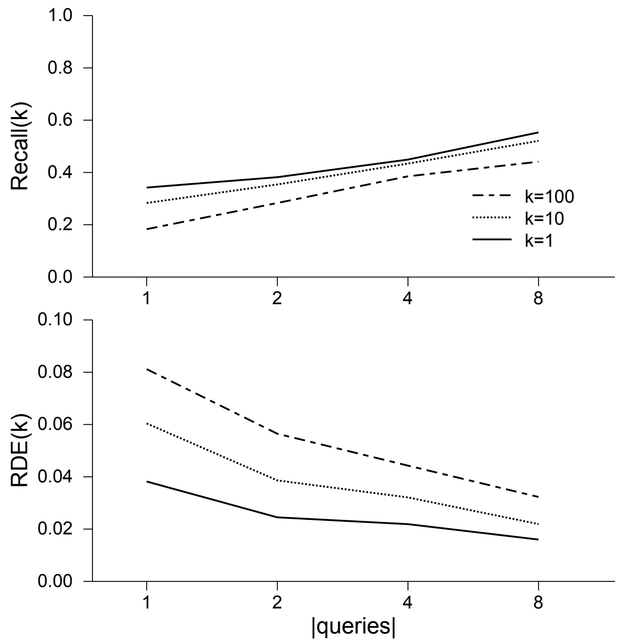


Fig. 2. Multiple query search strategy on the 100M index, using $|R| = 1,000$ and $z = 1,000$.

Fortunately, given such "weak searcher", it is easy to boost its effectiveness by adopting two simple strategies:

Multiple query strategy: pairs of elements of Π_q^l can be swapped in order to generate additional permutation prefixes to be passed to the search function, thus retrieving additional candidates among the objects assigned to permutations neighboring the one representing the query. This strategy can be easily parallelized on multi-processor hardware.

Multiple index strategy: multiple indexes can be built on the same collection D, by using different sets of reference objects R, producing different mappings of objects to permutations. This increases the probability of performing a complete exploration of the regions of the similarity space around the query. This strategy can be easily parallelized by distributing the indexes over different machines. Moreover, this strategy allows to support fault-tolerance, thanks to data replication, and load-adaptation, e.g., by varying the number of indexes/machines dedicated to answer a specific query in proportion to the current load of the system.

Figures 1 and 2 show the effectiveness improvements measured by separately applying the two strategies, which results in an average 74% recall, for $k = 100$, when issuing the queries on eight indexes. The search cost time is linearly proportional to the number of queries/indexes involved into the search process.

The effectiveness can be obviously further improved by combining the two strategies. For example, an eight indexes/eight queries configuration produces almost exact results (97% recall, $k = 100$) in an average 12.45 second time, with a completely sequential execution of the queries, on the demo machine described in Section III.

III. MiPai architecture

The MiPai system has a modular structure, in which each module is represented by a *Web service* or an user application (see Figure 3).

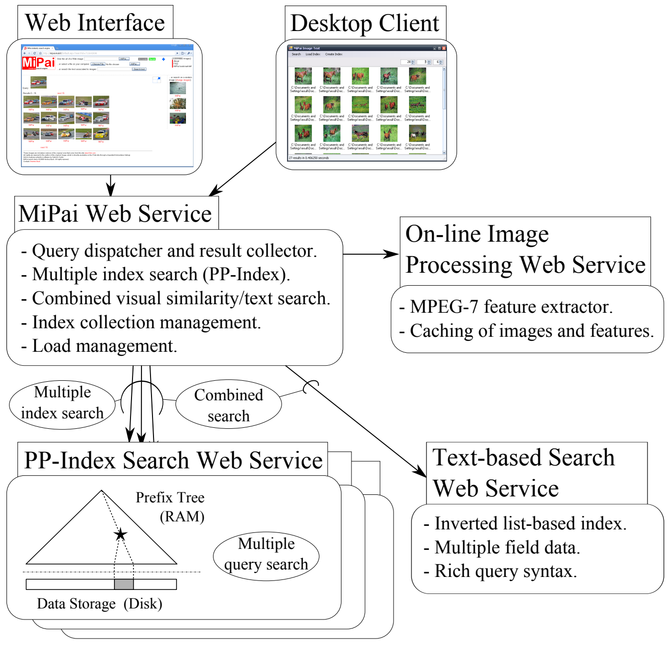
PP-Index search Web service: it provides the visual similarity search functionality on a single PP-Index structure. As described in Section II, the effectiveness/efficiency trade-off on a single index can be tuned by adopting a multiple query search strategy and by varying the z value. The similarity measure adopted for the experiments and for the demo system is a linear combination of the five similarity measures defined for the MPEG-7 visual descriptors that are associated to CoPhIR images, using the weights proposed in [6].

On-line image processing Web service: it extracts the MPEG-7 visual descriptors from a given image. Images can be uploaded by users from their PCs or be submitted as URLs pointing to external sites. In the latter case the service connects to the address specified by the URL and downloads the image. The service also provides a cache system for already-processed images.

Text-based search Web service: it manages an inverted list-based index of all the structured textual content associated to CoPhIR images (e.g., title, description, tags, comments). It uses a rich query language that allows to perform field-specific search, using standard *and/or* operators and also more complex *phrase* and *proximity* operators, or combinations of them. The Web service is implemented using the *Scheggia*[3] search library.

MiPai Web service: it is the core component of the MiPai system. It gathers all the available resources, also in multiple instances, and manages them in order to implement the search functionalities, e.g., producing more accurate results by using a multiple index search strategy, or combining the results of

[3]http://www.esuli.it/scheggia/



Web Interface
Desktop Client
MiPai Web Service
- Query dispatcher and result collector.
- Multiple index search (PP-Index).
- Combined visual similarity/text search.
- Index collection management.
- Load management.
On-line Image Processing Web Service
- MPEG-7 feature extractor.
- Caching of images and features.
Multiple index search
Combined search
PP-Index Search Web Service
Prefix Tree (RAM)
Multiple query search
Data Storage (Disk)
Text-based Search Web Service
- Inverted list-based index.
- Multiple field data.
- Rich query syntax.

Fig. 3. Architecture of the MiPai system.

visual similarity search and text-based search. A MiPai Web service can also connect to other instances of MiPai Web services, thus acting as a meta-search engine.

Web interface: publishes the search functionalities to the user, using a simple user interface (see details in Section IV).

Desktop client: a search client application that could remotely connect to any instance of a MiPai Web service.

The modularity of the system allows to rapidly adapt its configuration to the operating environment, e.g., supporting the deployment of the system over a distributed system, or handling dynamically available resources.

All the above described components are implemented in c# and can run on both Windows and Linux platforms.

The current MiPai demo has all of its components running on a single desktop PC, with a 4 cores 2.4 GHz CPU, a single 1 TB disk, and 4 GB RAM. The multiple index configuration is composed by eight PP-Indexes, each one covering the entire CoPhIR collection. It is worth to note that all the eight indexes are simultaneously loaded on the machine, further showing scalability the potential of the PP-Index.

The time required to build a single PP-Index is 12.5 hours. This value is in line with the time required to index the textual content of the entire CoPhIR collection, which amounted to 14 hours.

Any similarity search request is processed by adopting a multiple index strategy on all the eight indexes, and it is performed by sequentially searching on each index. On each index, the search process uses a two-query search strategy with $z = 1,000$, for the "quick" configuration (see Section IV), or a four-query strategy, $z = 10,000$, for the "accurate" configuration.

IV. MiPai Web search interface

The MiPai Web interface is designed to free users from having to deal with PP-Index-specific search parameters, presenting them a simplified interface.

The user can start a visual similarity search by selecting one of the random images proposed by the system or, more relevant, can submit the URL of an image on the Web or upload a personal image. In the latter two cases the image processing Web service is used to extract the MPEG-7 data on the fly.

MiPai provides also a JavaScript *bookmarklet* which allows the user, while visiting any Web page, to quickly submit any image in that page for search on MiPai.

The user can start a search by submitting a textual query. The text-based search can also be used after a visual similarity search, in order to perform a *combined* visual/textual search. In this case the z' candidates retrieved by the PP-Index are ranked by a combined measure that averages the numeric quantities representing the distance of any object from the visual query and the degree of match of its associated textual content with the textual query.

Results are shown in a grid of 15 images, with a maximum of 100 images per query. Any image in results can be clicked in order to reach the original image on Flickr, or used as query in a new search. The size of images can be varied.

Instead of burdening the user with complex search parameters, MiPai gives the user the choice between two performance profiles, "quick" or "accurate", that correspond to different parameter settings for the search on the various data structures, as already detailed in Section III. Such parameters are also adapted to the load of the system, in order to always guarantee its responsiveness.

The user can also select an "expanded" view of the interface, which allows to selectively enable/disable any of the five visual descriptors, determining the ranking of the z' candidates by a differently-balanced distance measure (e.g., in order to give more relevance to edges over color distribution), and to give feedback on the quality of each result (currently only logged into a db).

References

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

[2] G. Amato and P. Savino, "Approximate similarity search in metric spaces using inverted files," in *INFOSCALE '08, Proceeding of the 3rd International ICST Conference on Scalable Information Systems*, Vico Equense, Italy, 2008, pp. 1–10.

[3] E. Chávez, K. Figueroa, and G. Navarro, "Effective proximity retrieval by ordering permutations," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 30, no. 9, pp. 1647–1658, 2008.

[4] D. R. Morrison, "Patricia—practical algorithm to retrieve information coded in alphanumeric," *J. ACM*, vol. 15, no. 4, pp. 514–534, 1968.

[5] M. Patella and P. Ciaccia, "The many facets of approximate similarity search," *SISAP '08, 1st International Workshop on Similarity Search and Applications*, pp. 10–21, 2008.

[6] M. Batko, P. Kohoutkova, and P. Zezula, "Combining metric features in large collections," *SISAP '08, 1st International Workshop on Similarity Search and Applications*, pp. 79–86, 2008.