A Brief Index for Proximity Searching

Eric Sadit Téllez[1], Edgar Chávez[1,2], and Antonio Camarena-Ibarrola[1]

[1] Universidad Michoacana
[2] CICESE

Abstract. Many pattern recognition tasks can be modeled as proximity searching. Here the common task is to quickly find all the elements *close* to a given query without sequentially scanning a very large database.

A recent shift in the searching paradigm has been established by using *permutations* instead of distances to predict proximity. Every object in the database record how the set of reference objects (the permutants) is *seen*, i.e. only the relative positions are used. When a query arrives the relative displacements in the permutants between the query and a particular object is measured. This approach turned out to be the most efficient and scalable, at the expense of loosing recall in the answers. The permutation of every object is represented with κ short integers in practice, producing bulky indexes of $16\kappa n$ bits.

In this paper we show how to represent the permutation as a binary vector, using just one bit for each permutant (instead of $\log \kappa$ in the plain representation). The Hamming distance in the binary signature is used then to predict proximity between objects in the database. We tested this approach with many real life metric databases obtaining faster queries with a recall close to the Spearman ρ using 16 times less space.

1 Introduction

A metric space is composed by an universe of objects $\mathbb{U}$, and a distance function $d : \mathbb{U} \times \mathbb{U} \to \mathbb{R}$, such that for any $x, y, z \in \mathbb{U}$, $d(x, y) > 0$, $d(x, y) = 0 \iff x = y$, $d(x, y) = d(y, x)$ (symmetry), and obeying the triangle inequality: $d(x, z) + d(z, y) \geq d(x, y)$.

Some common tasks require distances expensive to compute (i.e. comparing fingerprints, searching by content in multimedia, etc) and hence sequential scan does not scale for large problems. Proximity queries are usually of two types, for a given database $S \subseteq \mathbb{U}$ with size $|S| = n$, $(q \in \mathbb{U}, r \in \mathbb{R})_d = \{x \in S \mid d(q, x) \leq r\}$, denote a *range query*. The other type of query is the focus of this paper, the k *nearest neighbor*, denoted $kNN_d(q)$, which retrieve the k closest elements to q in S, formally it retrieves the set $R \subseteq S$ such that $|R| = k$ and $\forall u \in R, v \in S - R$ it follows $d(q, u) \leq d(q, v)$.

Most indexes use the triangle inequality to avoid a sequential scan. Upper bounds of the distance between the query and the database objects can be obtained by computing some distances beforehand to the so-called *pivots* or by dividing the space in regions with the so-called *compact partitioning* indexes. Due to space restrictions we do not overview current approaches, nevertheless a

E. Bayro-Corrochano and J.-O. Eklundh (Eds.): CIARP 2009, LNCS 5856, pp. 529–536, 2009.

deeper and extended catalog for searching in metric spaces can be found in [1,2,3]. We will focus on the permutations index (described in detail below) because it has shown to be very scalable, indexing hundreds of millions of images in the Cophir project [6].

1.1 Overview of the Permutations Based Index

The motivation behind this indexing method [4] is to shift the problem of comparing directly the query object against every object in the database to comparing the *perspective* in which a set of elements are perceived. Each database element has an unique perspective of the *permutants* (defined below) and the query is only compared to those elements having similar perspective of the permutants.

Let $\mathbb{S}$ be the database of objects, and $\mathbb{P} \subseteq \mathbb{S}$ be a set of distinguished objects from the database, called *permutants*. Assume x is the query. Each x defines a *permutation* Π_x, where the elements of $\mathbb{P}$ are written in increasing order of distance to x. Ties are broken using any consistent order, for example, the order of the elements in $\mathbb{P}$.

Definition 1. *Let* $\mathbb{P} = \{p_1,\ p_2, \ldots, p_k\}$ *and* $x \in \mathbb{X}$. *Then we define* Π_x *as a permutation of* $(1 \ldots k)$ *so that, for all* $1 \leq i < k$ *it holds either* $d(p_{\Pi_x(i)}, x) < d(p_{\Pi_x(i+1)}, x)$, *or* $d(p_{\Pi_x(i)}, x) = d(p_{\Pi_x(i+1)}, x)$ *and* $\Pi_x(i) < \Pi_x(i+1)$.

Each database element u will be represented by a permutation Π_u. The query will be represented by Π_q using the same definition. Elements that are close will have similar permutations. Defining a similarity is central to obtain good results. An excellent predictor is the Spearman Rho, defined as the sum the squares of differences in the relative positions of each element in both permutations. That is, for each $p_i \in \mathbb{P}$ we compute its position in Π_u and Π_q, namely $\Pi_u^{-1}(i)$ and $\Pi_q^{-1}(i)$, and sum up the squares of the differences in the positions [4]. Formally defined below in 2.

Definition 2. *Given permutations* Π_u *and* Π_q *of* $(1 \ldots k)$, *Spearman Rho is defined as*

$$S_\rho(\Pi_u, \Pi_q) \ = \ \sum_{1 \leq i \leq k} \left(\Pi_u^{-1}(i) - \Pi_q^{-1}(i)\right)^2.$$

We use the same example depicted in [4] for illustrating the definition of $S_\rho(\Pi_q, \Pi_u)$. Let $\Pi_q = 6, 2, 3, 1, 4, 5$ be the permutation of the query, and $\Pi_u = 3, 6, 2, 1, 5, 4$ that of an element u. A particular element p_3 in permutation Π_u is found two positions off with respect to its position in Π_q. The differences between permutations are: $1-2, 2-3, 3-1, 4-4, 5-6, 6-5$, and the sum of their squares is $S_\rho(\Pi_q, \Pi_u) = 8$.

Note that we can compute $S_\rho(\Pi_q, \Pi_u)$ by obtaining the inverse of both permutations and then computing the Euclidean distance of the inverse. It is also

shown in [4] that we can use the sum of the absolute of the differences, without the squares, without noticeable penalization in the index recall.

The result is a table of n rows (one per database element) and k columns (one per permutant). Each cell needs $\lceil \log_2 k \rceil$ bits to store one permutation at each row. The indexing cost is kn distance computations plus $O(nk \log k)$ CPU time to sort all the permutations.

The search has two phases. The first sorts the database according to the permutation distance and selects as candidates the first elements. The second phase is to check the list. The permutation index allows kNN searches in pseudo-metric spaces, because the triangle inequality is not used explicitly. Our technique inherits this property allowing faster searches and smaller indexes.

2 The Brief Permutations

Our goal is to achieve the same performance of the permutations based index using only one bit to represent each permutant. The Algorithm 1 shows the algorithm *Encode* for condensing the permutation information into bit strings. In *Encode* we can note that for big enough m (e.g $m \geq \frac{|P|}{2}$) the permutants in the center will be rarely set to 1. In order to reduce this effect we compute a second swapped permutation codifying them in the same bit string, as depicted in Algorithm 2.

Algorithm 1. Bit-encoding of the permutation P under the module m

Encode(Permutation P, Positive Integer m)

```
1: Let P^{-1} be the inverse P.
2: C ← 0 {Bit string of size |P|, initialized to zeros}
3: for all i from 0 to |P| − 1 do
4:     if |i − P^{-1}[i]| > m then
5:         C[i] ← 1
6:     end if
7: end for
8: return C
```

The brief index encodes all the objects in the database in different bit-strings, the Hamming distance is used to compare objects, instead of the Spearman Rho. The searching is shown in Algorithm 3, I is the brief index. Computing Hamming distances is way faster than computing the Spearman Rho, and this is the only operation needed to satisfy queries.

To fix ideas, consider the following example. Let $m = 2$, $u = (3, 6, 2, 1, 5, 4)$, $r = (5, 3, 1, 6, 2, 4)$ and $q = (6, 2, 3, 1, 4, 5)$. After the inverse $u^{-1} = (4, 3, 1, 6, 5, 2)$, $r^{-1} = (3, 5, 2, 6, 1, 4)$ and $q^{-1} = (4, 2, 3, 5, 6, 1)$. Applying algorithm 1 we have $\hat{u} = (|1-4| > m, |2-3| > m, |3-1| > m, |4-6| > m, |5-5| > m, |6-2| > m) = (1, 0, 0, 0, 0, 1)$, supposing $|a - b| > m$ evaluates to 1 for true and 0 for false. Similarly, we obtain $\hat{r} = (0, 1, 0, 0, 1, 0)$ and $\hat{q} = (1, 0, 0, 0, 0, 1)$. If H is the hamming distance, $H(\hat{u}, \hat{q}) = 0$ and $H(\hat{r}, \hat{q}) = 2$. Clearly, q is the closer one to u, and this can be verified using S_ρ as $S_\rho(u, q) = 8$, and $S_\rho(r, q) = 46$.

Algorithm 2. Bit-encoding using permutation of the center. Interchangeable with Encode.

EncodePermCenter(Permutation P**, Positive Integer** m**)**

Let P^{-1} be the inverse P
$C \leftarrow 0^{|P|}$ {Bit string of size $|P|$, initialized to zeros}
$M = \frac{\lfloor |P| \rfloor}{4}$
for all i from 0 to $|P| - 1$ **do**
  $I \leftarrow i$
  if $\lfloor \frac{I}{M} \rfloor \bmod 3 \neq 0$ **then**
    $I \leftarrow I + M$
  end if
  if $|I - P^{-1}[i]| > m$ **then**
    $C \leftarrow C|(1 << i)$
  end if
end for
return C

The binary mapping works because in essence it reproduces the same behavior than S_ρ with coarse granularity using two possible values (i.e. 0 and 1). Suppose two vectors u and v and we want to select the closest vector to q. In the brief representation we are neglecting (not adding) the displacements smaller than a certain threshold. This is compensated by using a larger number of permutants in the index.

As we are using bit encoded values we can use XOR bit-operation ($\oplus$) using the bit parallelism inherent in the computer integer operations computing 32 or 64 operations per instruction instead of the most expensive operations difference and product used in the S_ρ. The count of the enabled bits can be calculated using a previously calculated table for one or two bytes.

We can resume that $0 \oplus 0 = 0$ means an small movement difference, $0 \oplus 1 = 1$ meaning a big difference. $1 \oplus 1$ can significant a really big difference or an small one, in order to encode in just one bit each permutant we choose only one, and to be able to use the hamming distance we choose $1 \oplus 1 = 0$. Choosing $1 \oplus 1 = 1$ can be efficiently computed using $\oplus$ as OR instead of XOR.

Algorithm 3. Procedure to search kNN for q

SearchKNN(Hamming Index I**, Permutants** $\mathbb{P}$**, Distance** d**, Object** q**, Positive Integer** m**, Positive Integer** k**, Positive Integer** $Cand$**)**

$P \leftarrow$ Get permutation for q under $\mathbb{P}$ and d.
$\hat{h} \leftarrow Encode(P, m)$.
$R \leftarrow$ Retrieve the lNN for $\hat{h}$ with metric index I using $l = Cand$ {Remember that $R \subseteq S$}.
$Res \leftarrow [\,]$
for all $s \in R$ **do**
  $Res \leftarrow Res + [(d(s, q), s)]$
end for
$Res \leftarrow$ sort Res by the first argument in the tuple, keep the smallest k results.
return Res

3 Experiments

The tested databases were taken from the *metric space* library[1] and the natix project's site[2]. Our implementation is available as open source from

[1] Metric space library `www.sisap.org`
[2] Natix web site is `www.natix.org`

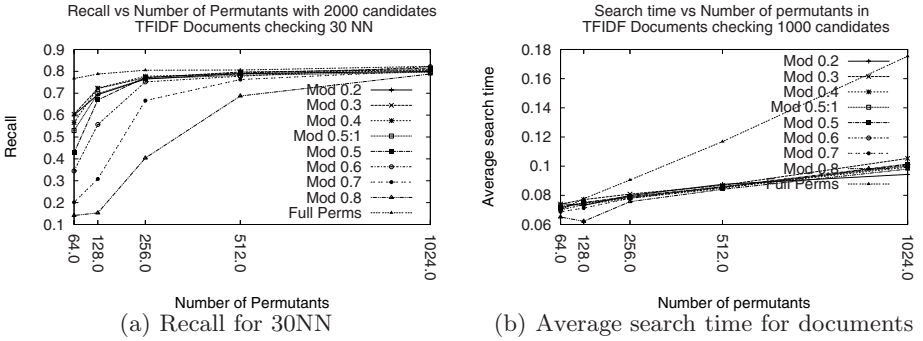
www.natix.org. All indexes share the same distance function's implementation. The experiments were performed in a laptop computer with Intel Core 2 at 2.4 GHz and 2GiB of RAM, running MacOS X 10.5.6. The indexes run in main memory and without parallelization. We have tested our index in a large number of real life databases, due to space constrains we show the results in three databases; documents for textual information retrieval, color's histogram vectors for multimedia information retrieval, and fingerprints of songs used for music information retrieval. Please note that the brief representation of the permutant space allows keeping the index in main memory. Secondary memory access is only needed when checking candidates.

The presented results compares the full permutation index and a set of brief indexes using different modules, the modules are presented in the Figures as the ratio between $\frac{m}{|P|}$. The Encode algorithm is used for all of them, except for curves named *Mod 0.5:1* whose are based on the EncodePermCenter algorithm.

3.1 Documents

A collection of 25157 short news articles in the $TF \times IDF$ format from Wall Street Journal $1987 - 1989$ files from TREC-3 collection. We use the angle between vectors as distance measure [5].

We extracted 100 random documents as queries, these documents were not indexed. Each query searches for 30 nearest neighbors (a metric index, like BKT [2] needs to check up to 98% of the database for this task). Figure 1(a) shows the recall for 30NN. Please note that the number of distance computations is the number of permutants plus the number of candidates, then for $30NN$ recall of 0.82 we need to review only the 6% of the database instead of the 98% in the alternative metric index (not shown for space constrains). If instead of $30NN$ we search for the nearest neighbot the recall increases to 97%. The recall for the brief permutations is closely related with the module, also note that as the number of permutants increases the module effect decreases. We can see that module 0.5



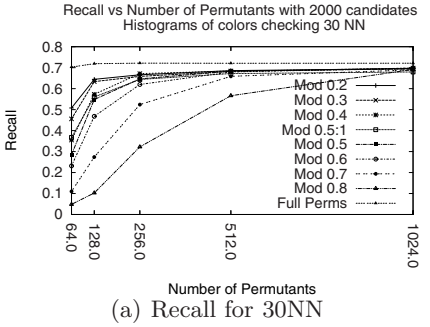
(a) Recall for 30NN (b) Average search time for documents

Fig. 1. Experiment results for brief index against full permutations using the documents $TF \times IDF$ collection and vector's angle as distance

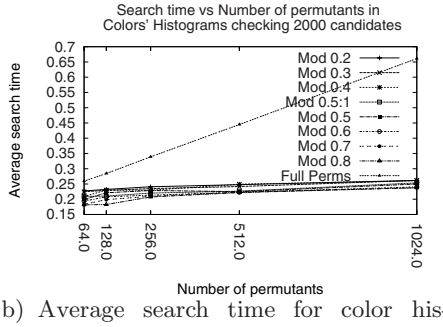
is a fair choice for any number of permutants. We can see in both recall Figures that the brief index performs slightly better than the full permutants. Figure 1(b) shows the average time per search needed for each number of permutants, naturally the brief representation is faster.

3.2 Vectors

We selected a set of 112544 color histograms (112-dimensional vectors) from an image database[3] We choose randomly 200 histogram vectors and we applied a *perturbation* of ± 0.5 on one random coordinate. The search consist on finding 30NN under L2 distance. The BKT needed to check 65% of the database. We achieved only a recall of 0.7 for 2000 checked candidates (equivalent to review a 2% of the database). This behavior is inherent to the permutations based index, the exact reasons of this behavior is unknown, but this experiment shows that the behavior is inherited by the brief index. Even with this *poor* recall, it's an excelent approximation for achieving fast searches for massive Multimedia Information Retrieval approaches [6] where a recall of 0.5 is reported for a larger database.



(a) Recall for 30NN



(b) Average search time for color histograms

Fig. 2. Results for brief index against full permutations using the color's histogram collection and L2 distance

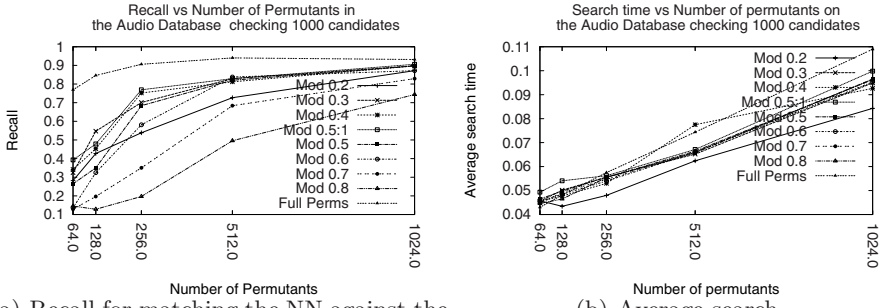
The behavior of this example is similar to the previous experiment, better times, less space for the same task.

3.3 Audio Fingerprints

A database of 10254 *multi-band spectral entropy signature* (MBSES) [7] using three byte's frame for each 46 ms. The signatures were extracted from full songs of assorted genre[4]. We use a non-metric distance called *probabilistic pairing psudo*

[3] The original database source is
`http://www.dbs.informatik.uni-muenchen.de/~seidl/DATA/histo112.112682.gz`

[4] The fingerprint's database is available from the `www.natix.org` website.



(a) Recall for matching the NN against the ground truth

(b) Average search

Fig. 3. Results for brief index against full permutations for the audio collection

metric [8] which is defined as the minimum hamming distance from one short sequence of length m against all m-grams inside a larger sequence. The distance's cost is $O(m \times (n - m + 1))$. We use excerpts of $20s$ as permutants and degraded excerpts of $20s$ as queries, both sets are disjoint.

Figure 3(a) shows a recall of 0.92 for the full permutation index, and 0.83 for the brief index using 512 permutants. Please note that BKT can be used at the expense of loosing some results because the probabilistic pairing pseudo metric do not follow the triangle inequality. The BKT gives a recall above 0.9 reviewing more than 40% of the database, resulting in 30 seconds per search. The brief index needs to review 512 distance's evaluations to compute the permutation, and 1000 distances verification (i.e. review 10% of the database, note that this is possible because permutants and queries have the same length). The verification is done using the transitivity kept by the distance, using only 12 or 24 frames, reducing the final cost of the query, Figure 3(b) shows the time per search.

4 Conclusions and Future Work

We have presented a new indexing method based on permutations. Our representation is able to use only one bit for each permutant, opposed to the 16 bit usual representation without noticeable impact in the recall of the index and 4 to 12 faster than full permutations. We are working on experiments in very large databases with an specialized indexing for Hamming distance for speeding up the searches (in the paper the times shows the effect in sequential scanning in the brief permutation space). Although *module* 0.5 is a good choice for any space and any number of permutants (specially when swapping center permutants as described in algorithm 2), a more careful tuning of the *modulus* used for obtaining the brief representation is needed.

Acknowledgments

We wish to thank the comments from the referees and from Rodrigo Paredes from University of Talca, in Chile, helping to improve the presentation.

References

1. Samet, H.: Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann Publishers, San Francisco (2006)
2. Chávez, E., Navarro, G., Baeza-Yates, R., Marroquín, J.L.: Searching in metric spaces. ACM Comput. Surv. 33(3), 273–321 (2001)
3. Böhm, C., Berchtold, S., Keim, D.A.: Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. ACM Computing Surveys 33(3), 322–373 (2001)
4. Chavez, E., Figueroa, K., Navarro, G.: Effective proximity retrieval by ordering permutations. IEEE Transactions on Pattern Analysis and Machine Intelligence 30(9), 1647–1658 (2008)
5. Baeza-Yates, R.A., Ribeiro-Neto, B.A.: Modern Information Retrieval. ACM Press / Addison-Wesley (1999)
6. Amato, G., Savino, P.: Approximate similarity search in metric spaces using inverted files. In: InfoScale 2008: Proceedings of the 3rd international conference on Scalable information systems, ICST, Brussels, Belgium, Belgium, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), pp. 1–10 (2008)
7. Ibarrola, A.C., Chávez, E.: A robust entropy-based audio-fingerprint. IEEE, Los Alamitos (2006)
8. Chavez, E., Camarena-Ibarrola, A., Téllez, E.S., Bainbridge, D.: A permutations based index for fast and robust music identification. Technical Report. Universidad Michoacana (2009)