



GCSE 2011: 28-30 December 2011, Dubai, UAE

Finding Good Permutants for Searching in Metric Spaces

Karina Figueroa Mora^{a,*}, Rodrigo Paredes^b

^a*Facultad de Ciencias Físico Matemáticas, Universidad Michoacana de San Nicolás de Hidalgo, México*

^b*Departamento de Ciencias de la Computación, Universidad de Talca, Curicó, Chile*

Abstract

The permutation index has shown to be very effective in medium and high dimensional metric spaces, even in difficult problems, for instance, when solving reverse k -nearest neighbor queries. Nevertheless, currently there is no study about which are the desirable features one can ask to a permutant set, or how to select good permutants.

Similar to the case of pivots, our experimental results show that, compared with a randomly chosen set, a good permutant set yields to fast query response or to reduce the amount of space used by the index. In this paper we start by characterizing permutants and studying their discrimination power, and then we propose an effective heuristic to select a good permutant candidate set. We also show empirical evidence that supports our technique.

© 2011 Published by Elsevier Ltd. Selection and/or peer-review under responsibility of GCSE 2011

Keywords: metric space indexing, probabilistic algorithms, indexing permutations

1. Introduction

Proximity or similarity searching is the problem of, given a dataset and a similarity criterion, finding elements of the set that are close or similar to a given query. This is a natural extension of the classical problem of exact searching. It is motivated by data types that cannot be queried by exact matching, such as multimedia databases containing images, audio, video, documents, and so on. In this new framework the exact comparison is just a type of query, while close or similar objects can be queried as well. There are several computer applications where the concept of similarity retrieval is of interest (see [1] for a comprehensive survey on applications). Some examples are machine learning and classification, image quantization and compression, text retrieval, computational biology, and function prediction.

Similarity queries can be formalized using the metric space model [1,2,3,4]. There is a universe X of objects and a distance function $d : X \times X \rightarrow \mathfrak{R}^+ \cup \{0\}$ defined among them. Objects in X do not

* Corresponding author. Tel.+52-443-3223500.

E-mail address: karina@fisimat.umich.mx, raparede@utalca.cl.

necessarily have coordinates (for instance, strings, images, audio or video). The distance function gives us a dissimilarity criterion to compare objects from the universe. Therefore, the smaller the distance between two objects, the more “similar” they are. The distance satisfies the following properties that make (X, d) a metric space: $d(x, y) \geq 0$ (positiveness), $d(x, y) = d(y, x)$ (symmetry), $d(x, x) = 0$ (reflexivity) and $d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality). These properties hold for many reasonable similarity functions.

The typical scenario of the metric space search problem considers that there is a finite database or dataset of interest $U \subset X$ of size n . Then, given a new object $q \in X$, a proximity query consists in retrieving objects from U relevant to q . There are two basic proximity queries or primitives. The first is the range query (q, r) , which retrieves all the elements in U which are within distance r to q . Formally, $(q, r) = \{u \in U, d(q, u) \leq r\}$. The second is the k -nearest neighbor query $NN_k(q)$, which retrieves the k closest-to- q elements in U . This is, $|NN_k(q)| = k$, and for all $u \in NN_k(q), v \in U \setminus NN_k(q), d(u, q) \leq d(v, q)$.

Given the database, these similarity queries can be trivially answered by performing $n = |U|$ distance evaluations. Yet, as the distance is assumed to be expensive to compute (think, for instance, in comparing two fingerprints), it is customary to define the complexity of the search as the number of distance evaluations performed, disregarding other components such as CPU time for side computations and even I/O time. Thus, the ultimate goal is to build offline an index in order to speed up online queries.

One of the most effective similarity search indices is the Permutation Index [5] (PI), particularly well suited for medium and high dimensional spaces. Despite that its practical version is focused on retrieving an approximate answer for a given query, its effectiveness allows us to consider it as an (almost) exact answer. As a matter of fact, experimental results shown in [5] reveal that the PI is extremely promissory.

Nevertheless, there is no study about which are the desirable features one can ask to a permutant set, or how to select good permutants. This is the main focus of this paper.

Similar to the case of traditional pivots [6,7], our experimental results show that a good permutant set yields to fast query response or to reduce the amount of space used by the index compared to a randomly chosen set. In this paper we start by characterizing permutants and studying their discrimination power. Next, we propose an effective heuristic to select a good permutant candidate set. Finally, we also show empirical evidence that supports our technique.

2. Related Work

2.1. Previous work on choosing pivots

In general, metric space indices based on pivots choose them randomly among the database objects. However, there are some preliminary attempts to choose pivots [4,10], which are based in two ideas: good pivots are far away from the rest of the database, and also far away of each other pivots. These techniques have good performance in some metric spaces but they fail in others.

There are two works on how to systematically select a good set of pivots [6,7]. In [6], the authors propose an efficiency measure to compare two pivot sets, maximize the mean of the distribution of distances among pivots. Based on that measure they also provide three techniques to select good sets of pivots: (i) produce several pivot sets at random and choose the set with the highest average distance; (ii) choose pivots incrementally, the first pivot p_1 is the object that have the maximum average distance to other objects, the second one p_2 is selected so that the subset $\{p_1, p_2\}$ has the maximum average distance, and so on until selecting all the pivots; and (iii) starting with a random set of pivot, in each iteration the pivot having the smallest contribution to the average distance is removed and replaced by a better pivot.

In [7], the authors propose a dynamic pivot selection technique that combines the third alternative in [6] with the Sparse Spatial Selection (SSS) technique [10]. It uses the same SSS's pivot insertion

criterion, but checking if any of the already selected pivots becomes redundant (in the sense that its contribution to some efficiency criterion is low) or the new pivot candidate is redundant with respect to the current pivot set.

2.2. The Permutation Index

Let $P \subset U$ be a set of permutants (also called anchors). Each element $u \in U$ induces a preorder \leq_u on P given by the distance from u towards each permutant, defined as $y \leq_u z \Leftrightarrow d(u,y) \leq d(u,z)$, for any pair of permutants $y, z \in P$. The relation \leq_u is a preorder and not an order because some permutants can be at the same distance of u . So, it could be possible to find $y, z \in P, y \neq z$, such that $y \leq_u z \wedge z \leq_u y$.

Let $\Pi_u = i_1, i_2, \dots, i_{|P|}$ be the permutation of u , where permutant $p_{i_j} \leq_u p_{i_{j+1}}$. Permutants at the same distance take an arbitrary but consistent order. Every object in U computes its preorder of P and associates it to a permutation, which is stored in the index (this index does not store distances). Thus, a simple implementation needs $n|P|$ space. The crux of this index is that two equal objects must have the same permutation, while similar objects would hopefully have similar ones. So, if Π_u is similar to Π_q we expect that u is close to q . So, we have changed the problem from searching U to searching the set of permutations for each object in U .

At query time, we compute Π_q and compare it with all the permutations stored in the index. So, we traverse (the permutations of objects in) U in the order \leq_{Π_q} induced by Π_q (by increasing permutation dissimilarity). If we limit the number of distance computations we obtain a probabilistic search algorithm. Fortunately, the order \leq_{Π_q} induced is extremely promissory, as reported in [5] for range queries.

Similarity between the permutations of q and u can be measured by Kendall Tau (K_τ), Spearman Footrule (S_F), or Spearman Rho (S_ρ) metric [11], among others. K_τ can be seen as the number of swaps that a bubble-sort-like algorithm has to do in order to make two permutations equal. Using $\Pi^{-1}(i_j)$ to denote the position of permutant p_{i_j} in the permutation Π , S_F and S_ρ are defined as follows:

$$S_F(\Pi_u, \Pi_q) = \sum_{j=1, \dots, |P|} |\Pi_u^{-1}(i_j) - \Pi_q^{-1}(i_j)|, \quad \text{and} \quad S_\rho(\Pi_u, \Pi_q) = \sqrt{\sum_{j=1, \dots, |P|} |\Pi_u^{-1}(i_j) - \Pi_q^{-1}(i_j)|^2}.$$

As S_ρ is monotonous, we can simple use S_ρ^2 . A numerical example follows in order to illustrate these permutation metrics. Let $\Pi_q = (42153)$ and $\Pi_u = (32154)$ be the query and object $u \in U$ permutations, respectively. So, $K_\tau(\Pi_u, \Pi_q) = 7$, $S_F(\Pi_u, \Pi_q) = 8$ and $S_\rho^2(\Pi_u, \Pi_q) = 16$.

3. Finding Good Permutants

In the following we will describe our approach to select good permutants. Our approach is based on experimental observations that lead us to the final methodology to obtain a good permutant set. We start by characterizing the permutants and then we propose a heuristic to select a permutant set.

3.1. Characterizing the permutants

Given any permutant, its maximum difference of positions between the query permutation Π_q and the permutation of any object in the database Π_u is $|P|-1$. This is obvious, however it suggests a simple way to determine how much a permutant can contribute in the searching process. Permutants at the beginning or the end of a permutation can produce an important increase in the permutation distance, while the ones

in the middle of the permutation produce a mild increase. Note that it is such increase the one that changes the order \leq_{n_q} induced by the permutation of the query Π_q .

Based in this observation we start our study by considering how contribute a portion of the permutation to the order \leq_{n_q} induced by the query. To illustrate our point we perform the following experiment. We divide the permutation Π_q in three equal portions, and show the results when searching the metric space using the permutants falling within two thirds of the permutation Π_q (that is, we exclude a portion). To do this, we compute the permutation distance using the following formula

$$S'_F(\Pi_u, \Pi_q) = \sum_{j=[1, |P|] \wedge \Pi_q^{-1}(i_j) \notin \text{portion}} |\Pi_q^{-1}(i_j) - \Pi_u^{-1}(i_j)|.$$

For this experiment, we simple use (\mathcal{R}^D, L_2) considering a uniformly distributed dataset composed by 80,000 vectors with $D \in [8, 128]$ components, and show the average over 100 randomly chosen queries. According to the results of [5], we use sets of permutants of size $D \pm 2$.

Fig. 1(a) shows the results when we discard either the left, middle, or right portions of the permutation. The figure shows that the best results are obtained when we discard the middle portion; this is in agreement with the intuition that the permutants that fall in this section do not heavily increase the permutation distance. On the other hand, the figure also says that the permutants in the left portion are the most important ones in terms of inducing the order \leq_{n_q} , which is also intuitive as they are the closest ones to q .

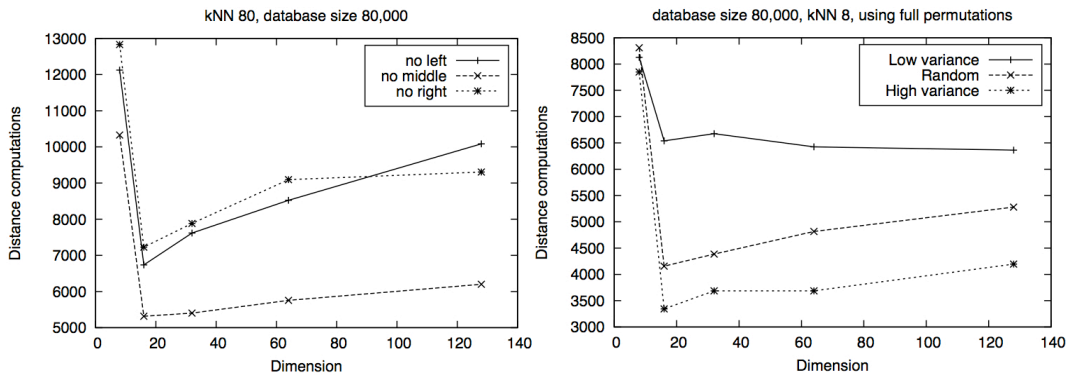


Fig. 1. (a) Retrieval performance when excluding one portion of the permutation; (b) Search performance of a random permutant set versus low and high variance permutant sets.

These observations lead us to the core of the following section, where we describe how to find good permutants. The main idea is trying to find permutants that do not fall in the middle portion. A simple way to do that is searching for permutants that concentrate their occurrences in some positions of the object permutations (hopefully in the left portion).

3.2. A simple heuristic to find good permutations

Given a permutant, one can compute the histogram of its positions in the permutations of all the elements in U . If that histogram is concentrated the permutant does have a preferred portion, and the variance of the histogram is large when compared with a permutant with a fairly uniform histogram.

Therefore, we now perform another preliminary experiment in order to verify this assumption. In this experiment we compare the retrieval performance of three sets. The first contains D randomly chosen permutants, the second has D permutants with low position variance, and the third are the D permutants with the highest variance. This experiment was done using the same setup of the previous section.

In order to choosing the permutants with the lowest and highest variance, we take a random sample of size $c = \lfloor (1 + \sqrt{1 + 8n})/2 \rfloor$ (where $n = |U|$). We choose this value since the full comparison among all the elements in the sample requires $c(c-1)/2 \leq n$ distance computations. So we maintain controlled the number distance computations. If $D > c$ we repeat the process until we have enough permutants to make the selection. However, since in practical cases $D = O(\sqrt{n})$ with one iteration is enough.

As can be seen from Fig. 1b, the best similarity query performance is verified with the set whose permutants present the largest variance. Note that the figure evidences that the difference is extremely noticeable, in particular in high dimensional spaces, where we verify that the high variance set performs only 79% of the distance computations required by the random set in dimension $D = 128$. For $D = 64$ the high variance set requires 76% of the distances computations of the random set. The second performance is achieved by the random chosen set. This behavior could be counter intuitive as we would expect that having a criterion is better that choosing permutants at random.

4. Experiments

In this section we show the performance of our heuristic in a real-world space of images. The set of image objects were taken from Flickr, using the URL provided by the SAPIR collection [12]. The content-based descriptors extracted from the images were: Color Histogram 3x3x3 using RGB color space (27dim vector), Gabor Wavelet (48dim vector), Efficient Color Descriptor (ECD) 8x1 using RGB color space (32dim vector), ECD 8x1 using HSV color space (32dim vector), and Edge Local 4x4 (80dim vector). The distance function used was Euclidean distance. The dataset has 1 million images.

Table 1 shows the performance of our technique when solving k -nearest neighbor queries. We test three values of k , namely 10, 100, and 1000, varying the size of the permutant set from 128 through 512 permutants, chosen using our heuristics in Section 3.

Usually, users retrieve a few numbers of images per query. So for illustrate our results, we remark that with $k = 10$, we need to compare only 0.4% to 0.6% of the database. If we use higher values of k , our index needs to review a larger fraction of the database, as expected; however, we note that the query performance is very mild (in fact, it is sublinear).

Table 1. Distance evaluations for k nearest neighbor queries using different values of k (columns) and number of permutants (rows)

Permutants \ kNN	10	100	1000
128	6511	35114	86878
256	5272	33258	86431
512	4654	30932	81827

5. Conclusions

In this paper we start by characterizing permutants and studying their discrimination power. Next, we propose an effective heuristic to select a good permutant candidate set. We also show empirical evidence that supports our technique. The experimental results show that our approach is very promising when retrieving objects from high dimensional spaces and also from real-world datasets.

In the future work, we are interested on compressing the permutant index through removing the middle of the permutations, as we state here. We are also very interested in how to characterize better the permutant set. Finally, the high variance permutant set of Fig. 1(b) is composed by permutants that

concentrate their occurrences in some specific positions of the permutation. In future work we also study how to we refine this permutant set.

Acknowledgements

We wish to thank the anonymous reviewers for many helpful comments. This work was partially supported by CONACyT (Mexico).

References

- [1] E. Chávez, G. Navarro, R. Baeza-Yates, and J. Marroquin, “Searching in metric spaces,” *ACM Computing Surveys (CSUR)*, vol. 33, no. 3, Sep. 2001, pp. 273–321.
- [2] G. Hjaltason and H. Samet, “Index-driven similarity search in metric spaces,” *ACM Transactions on Database Systems (TODS)*, vol. 28, no. 4, 2003, pp. 517–580.
- [3] H. Samet, “Foundations of Multidimensional and Metric Data Structures.” Morgan Kaufmann, 2006.
- [4] P. Zezula, G. Amato, V. Dohnal, and M. Batko, “Similarity Search – The Metric Space Approach,” Springer, *Advances in Database System*, vol. 32, 2006.
- [5] E. Chávez, K. Figueroa, and G. Navarro, “Effective proximity retrieval by ordering permutations,” *IEEE Trans. on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 30, no. 9, 2008, pp. 1647–1658.
- [6] B. Bustos, G. Navarro, and E. Chávez, “Pivot selection techniques for proximity searching in metric spaces,” *Pattern Recognition Letters (PRL)*, vol. 24, no. 14, 2003, pp. 2357–2366.
- [7] B. Bustos, O. Pedreira and N. R. Brisaboa, “A dynamic pivot selection technique for similarity search,” in *Wksp. on Similarity Search and Applications (SISAP’08)*, Abr. 2008, pp. 105–112.
- [8] P. Yianilos, “Data structures and algorithms for nearest neighbor search in general metric spaces,” in *Proc. ACM-SIAM Symp. on Discrete Algorithms (SODA’93)*, Jan. 1993, pp. 311–321.
- [9] L. Micó, J. Oncina, and E. Vidal, “A new version of the nearest-neighbor approximating and eliminating search (AESA) with linear preprocessing-time and memory requirements,” *Pattern Recognition Letters (PRL)*, vol. 15, 1994, pp. 9–17.
- [10] O. Pedreira and N. R. Brisaboa, “Spatial selection of sparse pivots for similarity search in metric spaces,” in *Conf. on Current Trends in Theory and Practice of Computer Science (SOFSEM’07)*, Springer, LNCS vol. 4362, 2007, pp. 434–445.
- [11] R. Fagin, R. Kumar, and D. Sivakumar, “Comparing top k lists,” *SIAM J. Discrete Mathematics (SIDMA)*, vol. 17, no. 1, 2003, pp. 134–160.
- [12] F. Falchi, M. Kacimi, Y. Mass, F. Rabitti, and P. Zezula, “Sapir: Scalable and distributed image searching,” in *SAMT (Posters and Demos)*, vol. 300, 2007, pp. 11–12.