Boosting the Permutation Based Index for Proximity Searching *

Karina Figueroa[1] and Rodrigo Paredes[2]

[1] Facultad de Ciencias Físico-Matemáticas,Universidad Michoacana, México.
[2] Departamento de Ciencias de la Computación, Universidad de Talca, Chile.
karina@fismat.umich.mx, raparede@utalca.cl

Abstract. Proximity searching consists in retrieving objects out of a database *similar* to a given query. Nowadays, when multimedia databases are growing up, this is an elementary task. The permutation based index (PBI) and its variants are excellent techniques to solve proximity searching in high dimensional spaces, however they have been surmountable in low dimensional ones. Another PBI's drawback is that the distance between permutations cannot allow to discard elements safely when solving similarity queries.
In the following, we introduce an improvement on the PBI that allows to produce a better promissory order using less space than the basic permutation technique and also gives us information to discard some elements. To do so, besides the permutations, we quantize distance information by defining distance rings around each permutant, and we also keep this data. The experimental evaluation shows we can dramatically improve upon specialized techniques in low dimensional spaces. For instance, in the real world dataset of NASA images, our boosted PBI uses up to 90% less distances evaluations than AESA, the state-of-the-art searching algorithm with the best performance in this particular space.

Keywords: Permutation based index, Distance quantization, Proximity searching

1 Introduction

Nowadays, similarity searching has become an important task for retrieving objects in a multimedia database; with applications in pattern recognition, data mining and computational biology, to name a few. This task can be mapped into a metric space problem. A Metric Space is a pair $(\mathbb{X}, d)$, where $\mathbb{X}$ is a universe of objects, and d is a distance function $d : \mathbb{X} \times \mathbb{X} \rightarrow \mathbb{R}^+ \cup \{0\}$. The distance function is a *metric* if it satisfies, for all $x, y, z \in \mathbb{X}$, the following properties:

* This work is partially funded by National Council of Science and Technology (CONACyT) of México, Universidad Michoacana de San Nicolás de Hidalgo, México, and Fondecyt grant 1131044, Chile.

reflexivity $d(x, y) = 0$ iff $x = y$, symmetry $d(x, y) = d(y, x)$, and triangle inequality $d(x, y) \leq d(x, z) + d(z, y)$. The last one being useful to discard objects when solving similarity queries.

In practical applications, we have a subset $\mathbb{U}$ of n objects taken from the universe $\mathbb{X}$. So, the similarity searching problem can be defined as the problem of finding a small subset $\mathbb{S} \subset \mathbb{U}$ of the objects that are close to a given query q with respect to a particular metric function.

Basically, there are two types of queries: range query $(q, r)_d$ and k-nearest neighbor query $kNN(q)_d$. The first one retrieves all the objects within a given radius r measured from q, that is, $(q, r)_d = \{u \in \mathbb{U}, d(q, u) \leq r\}$. The second retrieves the k objects in $\mathbb{U}$ that are the closest to q. Formally, $|kNN(q)_d| = k$, and $\forall\ u \in kNN(q)_d, v \in \mathbb{U} \setminus kNN(q)_d$, $d(u, q) \leq d(v, q)$.

There are some indices to speed up similarity queries. One of them, the permutation-based index (PBI) [3] has a competitive performance in the hard case of high dimensional spaces. Oddly, in low dimensional spaces, this technique has poor performance when compared with, for instance, the pivot-based index (which is particularly well suited for low dimensional spaces, as reported in [4]). Another important drawback of the PBI is that it does not allow to discard objects when solving similarity queries.

Our contribution consists in granting the basic PBI the capability of safely discard objects. For this sake, we enhance the PBI with distance information in a convenient way, so that the increment of the space requirement is very small. Our technique allows to improve the retrieval of the PBI in low and medium dimensional metric spaces in a dramatic way. As a matter of fact, in the real world metric space of NASA images, we obtain the true answer of the $1NN(q)_d$ using 90% less distances evaluations than AESA, the best pivot index.

2 Related Work

2.1 Metric Space Indices

There are three kinds of indices for proximity searching in metric spaces, namely, pivot-based indices, partition-based indices and permutation-based indices. In [4], the reader can find a complete survey of the first two kinds.

Pivot-based indices consider a subset of objects $A = \{a_1, a_2, \ldots, a_{|A|}\} \subseteq \mathbb{U}$, called the pivots. These indices keep all the $n|A|$ distances between every object of the dataset $\mathbb{U}$ to all $a_i \in A$. Later, to solve a range query $(q, r)_d$, the pivot-based searching algorithms measure $d(q, a_1)$ and, by virtue of the triangle inequality, for every $u \in \mathbb{U}$ they lower bound $d(q, u) \geq |d(q, a_1) - d(u, a_1)|$. So, if $|d(q, a_1) - d(u, a_1)| > r$ then $d(q, u) > r$ and they discard u avoiding the computation of $d(q, u)$. Once they are done with a_1, they use a_2 to try to discard elements from the remaining set, and so on, until using all the pivots in A. The elements that still cannot be discarded at this point are directly compared with q.

There are many techniques based on this idea. Some of them try to reduce the memory requirements in order to get a small index, and others to reduce the

number of distance evaluations. These kinds of techniques have a competitive performance in low dimensional spaces. One can imagine that a low dimensional space is one that can be embedded in a uniformly distributed vector space whose dimension is lower than 16, *preserving* the relative distances among objects.

Among the pivoting techniques, AESA [10] excels in the searching performance. To do this, AESA considers that every object could be a potential pivot. So, it stores the whole matrix of distances between every object pair. The matrix requires $\frac{n(n-1)}{2}$ memory. Since, every object can operate as a pivot, the authors also define a scheme to sequencing the pivot selection. For this sake, they use an array $SumLB$ which accumulates the lower bounds of the distances between the query and every non-discarded object, with respect to all the previous objects in $\mathbb{U}$ used as pivots. Formaly, if it has previously selected i pivots, $SumLB(u) = \sum_{j=1}^{i} |d(q, a_j) - d(u, a_j)|$. So, the first pivot is an object chosen at random, and from the second pivot, AESA uses the non-discarded object that minimize $SumLB(u)$. AESA is the bottom line of exact algorithms. However, in several real-world scenarios, AESA is impractical to use, as its memory requirement is only plausible for reduced size dataset (up to tens of thousands).

Partition-based indices split the space into zones as compact as possible and assign the objects to these zones. To do this, a set of centers $\{c_1, \ldots, c_m\} \in \mathbb{U}$ is selected, so that each other object is placed in the zone of its closest center. These indices have a competitive performance in high dimensional spaces.

2.2 The Permutation Based Index (PBI)

Let $\mathbb{P} \subset \mathbb{U}$ be a subset of permutants of size m. Each element $u \in \mathbb{U}$ induces a preorder $\leq_u$ 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 $y, z \in \mathbb{P}$.

Let $\Pi_u = i_1, i_2, \ldots, i_m$ 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 $\mathbb{U}$ computes its preorder of $\mathbb{P}$ and associates it to a permutation which is stored in the index (PBI does not store distances). Thus, a simple implementation needs nm space. Nevertheless, as only the permutant identifiers are necessary, it is possible to compact several permutants in a single machine word.

The hypothesis of the PBI is that two equal objects are associated to the same permutation, while similar objects are, hopefully, related to similar permutations. So, if Π_u is similar to Π_q one expects that u is close to q.

At query time, the PBI search algorithm computes Π_q and compares it with all the permutations stored in the index. Subsequently, the dataset $\mathbb{U}$ is traversed in increasing permutation dissimilarity, comparing the objects in $\mathbb{U}$ with the query using the distance d of a particular metric space. Regrettably, PBI does not allow the discarding of objects at query time. Instead, a premature cut off in the reviewing process produces a probabilistic search algorithm (as the search algorithm reports the right answer to the query with some probability).

There are many similarity measures between permutations. One of them is the L_p family of distances [5], that obeys Eq. (1).

$$L_p(\Pi_u, \Pi_q) = \sum_{j=[1,|\mathbb{P}|]} |\Pi_u^{-1}(i_j) - \Pi_q^{-1}(i_j)|^p \quad (1)$$

With $p = 1$, we obtain *Spearman Footrule* (S_F) and for $p = 2$ *Spearman Rho* (S_ρ). For example, let $\Pi_u = (42153)$ and $\Pi_q = (32154)$ be the object $u \in \mathbb{U}$ and query $q \in \mathbb{X} \setminus \mathbb{U}$ permutations, respectively. Thus $S_F(\Pi_u, \Pi_q) = 8$, $S_\rho(\Pi_u, \Pi_q) = 32$. As reported in [3], S_F has a good performance with less operations than the others.

There have been several works trying to improve the PBI's performance. In [7, 1]. Some variants have been proposed with the aim of reducing the space required by the permutations [2, 8, 9]. To do so, instead of using the whole permutation, just considering some of its portions. However, these technique lose precision in the query retrieval.

In general terms, all of the PBI's variants are designed for high dimensional spaces and none of them can prune the dataset when solving similarity queries.

2.3 Distance Quantization

A simple way to reduce the memory requirement when representing the distances among objects is to quantize them. Of course, there is a tradeoff between memory requirement and precision. However, there are some cases where the quantization is effective. For instance, in BAESA [6], the authors quantize the whole distance matrix of AESA [10]. Using only four bits per distance, eight times less space than AESA, BAESA needs just 2.5 times the distance computations of AESA.

3 Our Contribution

Essentially, we introduce a novel modification into the PBI. It consists in enhancing it with quantized distance information. Since we now have distance data, we can safely discard objects, and this dramatically improves the performance of the PBI search algorithm in low and medium metric spaces.

In the following, we describe the modification of the PBI and also how to adapt the PBI searching algorithm in order to benefit from the extra data.

3.1 Enhancing PBI with Distance Quantization

For each object $u \in \mathbb{U}$, the index stores not only its permutation, but also quantized distance information. To do that, for each permutant $p \in \mathbb{P}$, we define ζ concentric zones $z_1, \ldots, z_\zeta$ limited by $\zeta - 1$ radii $r_1, \ldots, r_{\zeta-1}$ and two extra values $r_0 = 0$ and $r_\zeta = \infty$ (for limiting the first and last zone). So, each object $u \in \mathbb{U}$ is located in the zone z_i where it satisfies the relation $r_{i-1} < d(u, p) \leq r_i$.

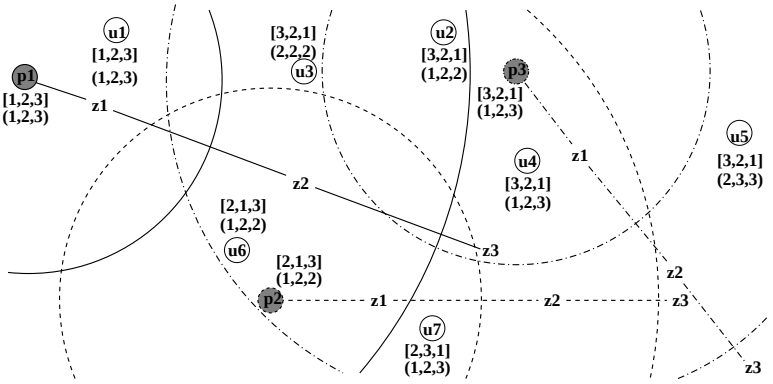


Fig. 1. Example of our technique, permutations and zones for every object.

To compute the zones, we do not need extra distance computations. In fact, we compute them during the calculation of the permutation. We call Π_u the permutation for u and Z_u the zones information for object u. So, for the permutant in the j-th cell of the permutation, $Z_u(j)$ indicates in which of its zones the object u lies. Hence, every object has its permutation and new extra information, the zone where it belongs according to every permutant.

Geometrically, for each $p_i \in \mathbb{P}$, this can be seen as partitioning the space with ζ distance rings centered in p_i. Fig. 1 illustrates the idea, where the dataset is $\mathbb{U} = \{p_1, p_2, p_3, u_1, \ldots, u_7\}$ and the first three objects are the permutants ($\mathbb{P} = \{p_1, p_2, p_3\}$). Each one has 3 zones (which are denoted by $z1, z2, z3$). For each object in $\mathbb{U}$, we show its permutation Π_u in the sequence closed by [ ] and its zone information Z_u in the one closed by ( ). For example, for u_6, its permutation is $[2, 1, 3]$ and for permutants p_2, p_1 and p_3, u_6 belongs to zones 1, 2 and 2. Notice that with our proposal, now we can figure out whether two objects with equal permutations are close or not.

Our proposal has two significant advantages, namely, (i) now we can discard some elements without compute their distances directly, and (ii) we improve the prediction power of the PBI.

In terms of space, besides the table of permutations, we need to store the distance information. Each object needs $m\lceil \log_2 m \rceil$ bits for the permutation (remember that $m = |\mathbb{P}|$) and also needs to store the zones. In order to represent the m zones (one for each permutant), we need $m\lceil \log_2 \zeta \rceil$ bits (where ζ is the number of zones). Finally, for each zone of a permutant, we need a float number to store the radius. This counts for $m\zeta 32$ bits. Adding up for the n objects we obtain $nm(\lceil \log_2 m \rceil + \lceil \log_2 \zeta \rceil) + \zeta m 32$ bits.

For the sake of determining an equivalence in space requirement between permutants with or without zones, we follow this rationale. The standard PBI uses $nm\lceil \log_2 m \rceil$ bits. The extra memory used by the zones allows to add more permutants to the plain PBI, but is not enough to double the number of permutants. Assuming that m is a power of 2, any extra permutant forces to use

another bit. We also assume that ζ is a power of 2. So, in the numerator of Eq. (2), we have the space used by the PBI with zones, and in its denominator, the space used by a plain PBI with m^* permutants, where $m^* \in (m, 2m)$.

$$\frac{nm(\log_2 m + \log_2 \zeta) + \zeta m32}{nm^*(\log_2 m + 1)} = \frac{m(\log_2 m + \log_2 \zeta)}{m^*(\log_2 m + 1)} + \frac{\zeta m32}{nm^*(\log_2 m + 1)} \quad (2)$$

The second term is $O\left(\frac{\zeta}{n \log_2 m}\right)$, so is negligible. The first term is $\frac{m(\log_2 m + \log_2 \zeta)}{m^*(\log_2 m + 1)}$ $= \frac{m}{m^*}\left(1 + \frac{-1 + \log_2 \zeta}{\log_2 m + 1}\right)$. To do a fair memory comparison, we set this term to 1. So, the number of equivalent permutants m^* for m permutants and ζ zones is

$$m^* = m\left(1 + \frac{-1 + \log_2 \zeta}{\log_2 m + 1}\right) \quad (3)$$

In the rest of this section, we show how to use the PBI index enhanced with the quantized distance information during the query time. For shortness, we call this index the PZBI.

Object discarding In order to discard objects in the new PZBI, we adapt the pivot excluding criterion. To do so, after computing the distance from q to all the permutants in order to produce Π_q, we compute the zones where q belongs (Z_q) and the zones where the query ball $(q, r)_d$ intersects. For this sake, we manage two arrays FZ and LZ. For each permutant, in FZ and LZ we store the first and last zone, respectively, where the query ball intersects. Therefore, given the query radius r, for each permutant $p \in \mathbb{P}$, FZ_p and LZ_p store the number of the zone that contains $d(p, q) - r$ and $d(p, q) + r$, respectively.

With this, when we are reviewing the objects, for each permutant $p \in \mathbb{P}$ we discard, by virtue of the triangle inequality, every element in $\mathbb{U}$ that belongs to any zone that is not in the range $[FZ_p, LZ_p]$. This allows discarding some elements without performing the direct comparison with the query.

Note that we can simulate $kNN(q)_d$ queries with range queries whose initial radius is ∞, and that its radius reduces to the final one as long as the search process progresses.

Improving the Distance Permutation Since we have more information per object (its permutation and corresponding zones), we can use the zone information so as to improve the reviewing order when solving a similarity query. We propose several strategies as follow. Let us define $Z_D(Z_u, Z_v) = \sum_{j=[1,|\mathbb{P}|]} |Z_u(j) - Z_v(j)|$. Z_D accumulates the sum of absolute values of differences in the zone identifiers between the objects u and $v \in \mathbb{X}$ for all the permutants. With this, we define the following variants:

- **PsF**: Just computing Spearman Footrule S_F as the plain PBI, see Eq. (1).
- **SPZ**: It is the sum of $S_F(\Pi_u, \Pi_q)$ and $Z_D(Z_u, Z_q)$.

– **PZ**: We compute $S_F(\Pi_u, \Pi_q)$ and $Z_D(Z_u, Z_q)$, separately. Next, we sort by S_F in increasing order and use Z_D to break ties.
– **ZP**: Analogous to **PZ**, but we sort by $Z_D(Z_u, Z_q)$ and break ties with $S_F(\Pi_u, \Pi_q)$.

Computing Radius Another important issue to define is how to establish the radii of the concentric zones. Every radius can be asigned as follow:

– **Uniformly distributed by distances** (REQ). We obtain a sample of objects and compute their distance towards the permutant. This gives us a good approximation of the distribution of distances with respect to that permutant. Let us call r_{max} and r_{min} the maximum and minimum distance computed in the sample for that permutant. Then, we basically use $(r_{max} - r_{min})/\zeta$ as the increment between one radius to the next.
– **Uniformly distributed by elements** (EEQ). Once again we obtain a sample of objects and compare them with the permutant. Later, we sort the distances and select points taken at regular intervals (a fraction $\frac{1}{\zeta}$ of the sample size).

4 Experimental Evaluation

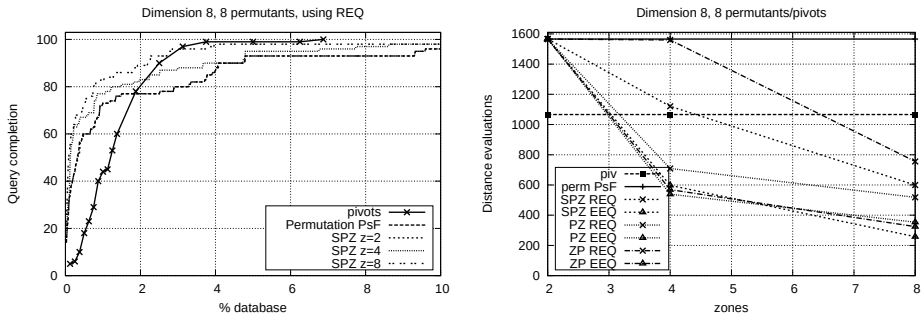
We tested our contribution with two kinds of databases: uniformly distributed vectors in the unitary cube of dimension in [8,32] and NASA images. The first one allows us to know the performance of the search algorithm and how the parameters can change the results. The second one gives us a good idea of how our technique works in real life.

4.1 Unitary Cube

We use a dataset with 80,000 vectors uniformly distributed in the unitary cube using the Euclidean distance. We tested $1NN(q)_d$ queries with a query set of size 100 in dimension 8 to 32.

In Fig. 2, we show the performance of the PZBI using 8 permutants in dimension 8, with REQ parameter. In Fig. 2(a), the y-axe shows the average of the percentage of elements retrieved by the $1NN(q)_d$, as we review an increasing portion of the database (x-axe). Fig. 2(b) shows the impact of the number of zones in the performance of the PZBI. Notice that we can process $1NN(q)_d$ queries faster than pivot-based algorithm. For instance, with 8 zones and EEQ, PZBI requires up to 81% less distance evaluations. Pivots are represented with a horizontal line just to allow a visual comparison of the results. In this plot, we can notice that the strategy SPZ is better than PZ or ZP.

In Fig. 3, we use dimensions 8, 16 and 32. We show that the PBI is beated by the PZBI using 8 zones and SPZ in low dimensional spaces. Notice that the PBI becomes better in high dimension. Pivot-based algorithm in dimension 16 is out the plot with almost 54,000 distances. In this figure we use m^* permutants



(a) Query completion. (b) Distance evaluation.

Fig. 2. Searching performance in the space of vectors of dimension 8, using 8 permutants/pivots.

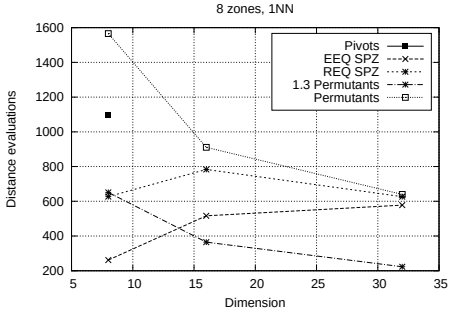


Fig. 3. Searching performance in the space of vectors, with 8 zones, using SPZ for dimensions 8, 16 and 32.

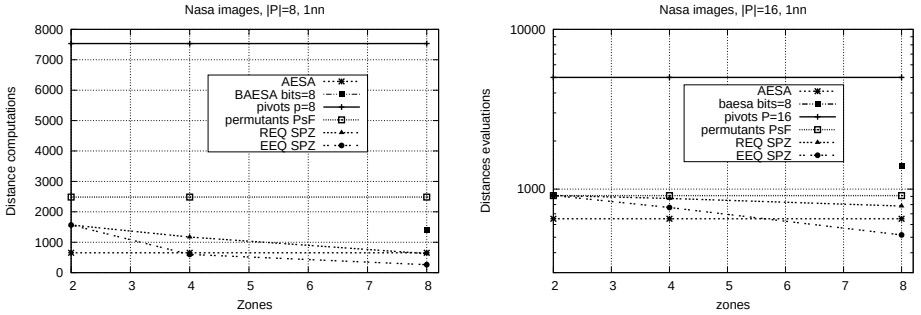
and we show that in high dimension is better to use more permutants, however, in low dimension, is better to split the space with zones.

4.2 NASA images

We use a dataset of 40,150 20-dimensional feature vectors, generated from images downloaded from NASA[3], where duplicated vectors were eliminated. We also use Euclidean distance.

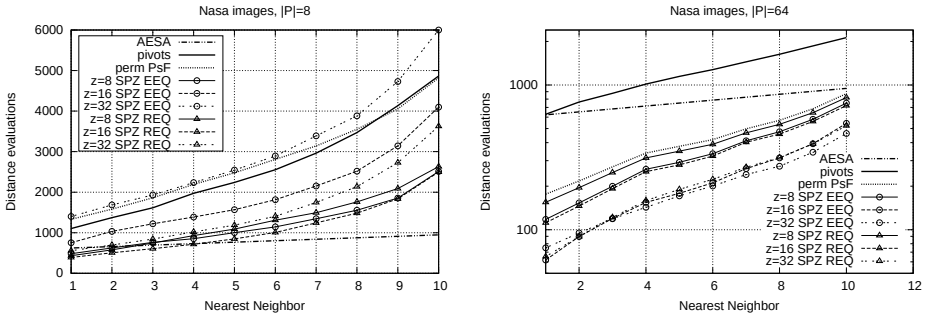
In Fig. 4, we use 8 and 16 permutants with 8 zones (Figs. (a) and (b), respectively). Notice that using the strategy SPZ we have an excellent improvement in the distance evaluations. In this figure, we are comparing our results with AESA (see Section 2.1) and its quantized version BAESA (see Section 2.3). Notice that our PZBI needs $\frac{1}{3}$ of the distance computations used by AESA, as can be seen when using 8 zones, EEQ space partitioning and SPZ. AESA and pivots are represented with an horizontal line in order to simplify the visual comparison.

[3] at `http://www.dimacs.rutgers.edu/Challenges/Sixth/software.html`



(a) $m = 8$ and $1NN(q)_d$. (b) $m = 16$ and $1NN(q)_d$.

Fig. 4. Nasa image dataset. Changing the number of zones and the impact on the selection of REQ or EEQ.



(a) $kNN(q)_d$ queries, $m = 8$ permutants. (b) $kNN(q)_d$ queries, $m = 64$ permutants.

Fig. 5. NASA image dataset. $kNN(q)_d$ queries, varying k.

In Fig. 5, we test $kNN(q)_d$ queries, varying $k \in [1, 10]$ (that is, increasing the size of the query answer), the number of zones $\zeta \in [8, 32]$, with two different size of permutants, $m = |\mathbb{P}| = 8$ and 64. In Fig. 5(a), we notice that using $m = 8$ permutants, we compute less than 50% of distance evaluations of AESA for $1NN(q)_d$ and just 3 times or $10NN(q)_d$ using significantly less space in the index. We also notice that with just 8 permutants and 8 zones, we can retrieve up to $4NN(q)_d$ faster than AESA. On the other hand, using $m = 64$ permutants with 32 zones (5 bits per element) we are computing just 25% of the pivot technique even in the hardest case.

In Fig. 5(b), we use 64 permutants and we have a better performance. For example, with 32 zones, 64 permutants and REQ we use just the 10% of distance of AESA, that is to say 90% less than the reference-algorithm for metric spaces.

In both plots (Figs. 4 and 5), our technique beats AESA. This is really surprising, as AESA used to be the lower bound of searching in metric spaces.

5 Conclusions and Future Work

The basic Permutant Based Index (PBI) has shown an excellent performance in high dimensional metric spaces. Its main drawback is that it does not allow to safely discard objects when solving similarity queries. Our contribution consists in granting the basic PBI the capability of safely discard objects. For this sake, we enhance the PBI with distance information in a quantized way.

Our technique allows to improve the retrieval of the PBI in low and medium dimensional metric spaces in a dramatic way.

In order to illustrate the benefits of our technique, we can say that in the real world metric space of NASA images PBI with quantized distance is capable to beat AESA search algorithm. As a matter of fact, with 32 zones and 64 permutants we use just the 10% of distance evaluation of AESA, that is to say 90% less than the reference-algorithm for metric spaces.

Future Work We plan to develop a searching algorithm to efficiently solve k-nearest neighbor queries based on our PBI with quantized distances.

Another trend is to develop efficient mechanism to avoid the sorting of the whole set of non-discarded objects.

References

1. Amato, G., Esuli, A., Falchi, F.: Pivot selection strategies for permutation-based similarity search. LNCS. Similarity Searching and Applications 8199, 91–102 (2013)
2. Amato, G., Savino, P.: Approximate similarity search in metric spaces using inverted files. In: Proc. 3rd Intl. Conf. on Scalable Information Systems. pp. 28:1–28:10. ICST (2008)
3. Chávez, E., Figueroa, K., Navarro, G.: Effective proximity retrieval by ordering permutations. IEEE Trans. on Pattern Analysis and Machine Intelligence (TPAMI) 30(9), 1647–1658 (2009)
4. Chávez, E., Navarro, G., Baeza-Yates, R., Marroquín, J.: Searching in metric spaces. ACM Computing Surveys 33(3), 273–321 (Sep 2001)
5. Fagin, R., Kumar, R., Sivakumar, D.: Comparing top k lists. SIAM J. Discrete Math. 17(1), 134–160 (2003)
6. Figueroa, K., Fredriksson, K.: Simple space-time trade-offs for AESA. In: Proc. 6th Workshop on Efficient and Experimental Algorithms (WEA'07). pp. 229–241. LNCS 4525, Springer–Verlag (2007)
7. Figueroa, K., Paredes, R.: An effective permutant selection heuristic for proximity searching in metric spaces. In: Proc. 6th Mexican Conf. on Pattern Recognition (MCPR'14). pp. 102–111. LNCS 8495, Springer (2014)
8. Figueroa, K., Paredes, R.: Compact and efficient permutations for proximity searching. In: Proc. 4th Mexican Conf. on Pattern Recognition (MCPR'12). pp. 207–215. LNCS 7329, Springer (2012)
9. Mohamed, H., Marchand-Maillet, S.: Quantized ranking for permutation-based indexing. LNCS. Similarity Searching and Applications 8199, 103–114 (2013)
10. Vidal, E.: An algorithm for finding nearest neighbors in (approximately) constant average time. Pattern Recognition Letters 4, 145–157 (1986)