Extreme Pivots for Faster Metric Indexes

Guillermo Ruiz[1], Francisco Santoyo[1], Edgar Chávez[2],
Karina Figueroa[1], and Eric Sadit Tellez[1]

[1] Universidad Michoacana de San Nicolás de Hidalgo, México
{gruiz,psantoyo}@dep.fie.umich.mx,
{karina,sadit}@fismat.umich.mx
[2] Universidad Nacional Autónoma de México
elchavez@matem.unam.mx

Abstract. Pivot tables are popular for *exact* metric indexing. It is well known that a large pivot table produces faster indexes. The rule of thumb is to use as many pivots as the available memory allows for a given application. To further speedup searches, redundant pivots can be eliminated or the scope of the pivots (the number of database objects covered by a pivot) can be reduced.

In this paper, we apply a different technique to speedup searches. We assign objects to pivots while, at the same time, enforcing proper coverage of the database objects. This increases the discarding power of pivots and in turn leads to faster searches. The central idea is to select a set of *essential* pivots (without redundancy) covering the entire database. We call our technique *extreme pivoting* (EP).

A nice additional property of EP is that it balances performance and memory usage. For example; using the same amount of memory, EP is faster than the List of Clusters and the Spatial Approximation Tree. Moreover, EP is faster than LAESA even when it uses less memory.

The EP technique was formally modeled allowing performance prediction without an actual implementation. Performance and memory usage depend on two parameters of EP, which are characterized with a wide range of experiments. Also, we provide automatic selection of one parameter fixing the other. The formal model was empirically tested with real world and synthetic datasets finding high consistency between the predicted and the actual performance.

1 Introduction

We are interested in the *proximity search problem*, where a metric space $(\mathcal{X}, d)$ is given. For a finite subset S of $\mathcal{X}$, and an element $q \in \mathcal{X}$, the goal is to find elements in S near q. One proximity query of interest is *range query*, given $r \geq 0$ we seek for $(q, r) = \{s \in S \mid d(s, q) \leq r\}$. It is also useful the notion of k *nearest neighbor query* (KNN), given an integer $k > 0$ find k nearest elements of S to q. KNN queries are equivalent to range queries if the search radius can be bounded. The ball centered at q with radius r is named the *query ball* (an analogous definition exists for KNN queries). The set $\mathcal{X}$ is often referred as the *universe* of objects, S is called the *database*, and q is called the *query object*.

The proximity search problem can be trivially solved with a sequential scan. This solution makes sense when the set S is small and the distance function d is cheap.

N. Brisaboa, O. Pedreira, and P. Zezula (Eds.): SISAP 2013, LNCS 8199, pp. 115–126, 2013.

For large datasets and/or expensive distance functions, metric indexing takes advantage of transitive properties of the metric to avoid a sequential scan. This problem has important practical applications in machine learning, pattern recognition, statistics, bioinformatics and textual and multimedia information retrieval, to name a few. There are several books and surveys (Chavez et al. [1], Samet [2], Zezula et al. [3]) describing with detail the different discarding rules, we will assume some familiarity of the reader with the subject.

There are two popular approaches to metric indexing, namely *pivot based indexes* and *compact partitions*. The later divides the data in spatially coherent regions, and at query time, regions without intersection with the query ball are discarded. The pivoting scheme consists in an implicit contractive mapping, where a distance $\delta(x, y)$ with the property $\delta(q, x) \leq d(q, x) \forall x \in \mathcal{X}$ is built using a set of objects (named pivots).

1.1 Related Work

For a long time, the main measure for performance comparison has been the number of distance computations to answer a query. The rationale behind this measure is because distances are the most expensive operation in a query. However, when measuring the actual time for answering a query it can be a surprise that a good index under this measure could be very slow when measuring the time. This may be led by a combination of a cheap distance and an expensive indexing structure.

AESA [4] stores $O(n^2)$ distances, and hence the construction cost is of the same order. At query time it performs a constant number of distance computations for a fixed database. However, they compute a linear number of arithmetic and logical operations at query time. A linear size restriction of the same idea is presented in LAESA [5]. The table of pivots comes from this initial idea.

Chavez et al. [1], proved that any pivot based metric index requires at least a logarithmic (on the database size) number of pivots (randomly selected from the database); however, the base of the logarithm depends on the intrinsic dimension of the database, needing larger indexes as the intrinsic dimension increases. This optimal number of pivots could not fit in main memory; hence, the rule of thumb is to use as much pivots as they fit.

Many of the pivots used are redundant if they are selected randomly. Hence, pivot selection became popular. Bustos et al. [6] present different strategies to this matter, showing that the proper selection of pivots is driven by datasets and queries. Remarkably, randomly chosen pivots can be good enough in any case. Finally, Celik [7,8] presents the priority vantage point method (Kvp). The Kvp is a structure where only the K most promising pivots are stored per object. It experimentally shows that the most promising pivots are those either near or far from the object.

Another approach consists in making a compact partition of the data. Here, two indexes are representative. The Spatial Approximation Tree (SAT) [9] is a metric tree where each node c is selected from S; c has $N(c)$ nodes, defined as $u \in N(c)$ if $d(u, c) < d(u, v)$, $v \in N(c)$, for all $u \in S \setminus \{c\}$. This procedure is recursively repeated for each $u \in N(c)$ with S as the set items in $S \setminus \{N(c) \cup \{c\}\}$ having u as its closer object in $N(c)$. Since $N(c)$ depends on the construction order, there are many $N(c)$. The author proposes to review S in the natural order imposed by the distance value

of each object to c. This index has good performance and no construction parameters (other than the order of the objects in $N(c)$); this simplicity of use makes the SAT a fair choice when there is not much knowledge of the database. Chavez and Navarro [10] present a robust and memory efficient alternative, dubbed as the List of Clusters (LC). The LC needs linear memory and up to quadratic construction time. At equality of memory, it remains unbeatable on datasets with very high intrinsic dimensionality.

2 Extreme Pivots

An Extreme Pivoting (EP) Index consists on a set of *pivot groups* (PG). Each group is a tuple $(\mathbb{P}, \alpha)$, where $\alpha > 0$ and $\mathbb{P}$ is a subset of S. For every $p \in \mathbb{P}$ there exists a set $A(p) \subset S$ such that for every $x \in A(p)$ we have that $|\mu_p - d(p,x)| \geq \alpha$. Here, μ_p is for the expected value of $d(u,p)$ for all $u \in S$. We must ensure that $\bigcup_{p\in\mathbb{P}} A(p) = S$ and $A(p_i) \cap A(p_j) = \emptyset$ for $i \neq j$; hence, both $\mathbb{P}$ and α define a partition of S. The proper selection of these parameters is deferred and discussed in upcoming sections. Let $p \in \mathbb{P}$, define $\overleftarrow{A}(p) = \{u \in A(p) \mid d(u,p) \leq \mu_p - \alpha\}$, and $\overrightarrow{A}(p) = \{u \in A(p) \mid d(u,p) \geq \mu_p + \alpha\}$ as depicted by Figure 1. For u in S, let $\mathsf{piv}(u)$ be the pivot such that $u \in A(p)$.

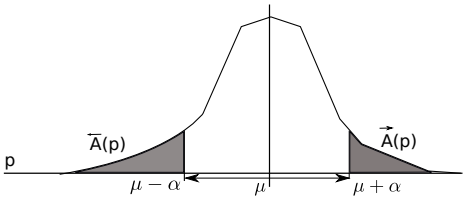


Fig. 1. The regions being controlled by α, from the perspective of pivot p

The elements of $A(p)$ will be called associates of p. Our construction depends critically on α, and in turn, the optimum value of α depends on the distribution of objects in the database. We will estimate α using some probabilistic assumptions, for now, please notice that when $\alpha = 0$, then just a pivot is required and our approach degrades to a pivot in a pivot table. If α is arbitrarily large, then $A(p)$ will be arbitrarily small. The parameter α governs more than just the size of associates of a pivot, as we will see.

Given an element u and a pivot p, let $a = Pr(u \in A(p))$, a is the probability of covering u with the pivot p. If $a \leq 1$ is a fixed value for a fixed database, then the number of pivots per group is $m = \frac{\ln(an+1)}{\ln(1/(1-a))}$.

In this expression, m is a function of a (or more precisely, it depends on α). The proper value of a is tightly coupled with the searching performance (based on the discarding probability induced by a pivot group), which is estimated in the following paragraphs.

Given a query (q, r), the probability that u is not discarded by ℓ pivots is

$$Pr(|d(q,p_1) - d(u,p_1)| \leq r, \ldots, |d(q,p_\ell) - d(u,p_\ell)| \leq r).$$

In order to simplify the analysis, we suppose all objects to be described by independent identical distributed random variables (i.i.d.r.v). Therefore, the previous expression can be rewritten as follows

$$Pr(|d(q,p) - d(u,p)| \leq r)^{\ell}.$$

To bound the above probability we will use the distribution of distances from the pivot to all the database elements.

In addition to i.i.d.r.v. assumption we assume that probability functions are symmetric around the mean. Let X be the random variable such that $X(u) \sim d(\mathsf{piv}(u), u)$, and let Y be $Y(u) \sim d(u,v)$ for $u, v \in \mathcal{X}$. Thus, the probability function of X is defined in the extremes of Y, i.e., $X(u)$ corresponds to the shaded area in Figure 1, and $Y(u)$ matches the whole region.

Since we assume symmetry, then for every $u \in \overleftarrow{A}(\mathsf{piv}(u))$ there exists an element $v \in \overrightarrow{A}(\mathsf{piv}(v))$ such that $X(u) + X(v) = D$ with D a constant. We assume as well that for every element u there exists an element v such that $Y(u) + Y(v) = D$. Using these assumptions, $E[X] = E[Y] = D/2$, hence $E[X - Y] = 0$. Let σ_X^2 and σ_Y^2 be the variance of X and Y respectively.

Now, $Pr(|d(p,u) - d(p,q)| > r) = Pr(|X - Y| > r)$ and, using the Chebychev inequality[1] we have $Pr(|X - Y| > r) < (\sigma_X^2 + \sigma_Y^2)/r^2$.

The probability that a given u would not be discarded by its covering pivot $\mathsf{piv}(u)$, is $1 - Pr(|X - Y| > r) \geq 1 - (\sigma_X^2 + \sigma_Y^2)/r^2$.

The number of distances the algorithm would compute is

$$\text{cost} = m\ell + n\left(1 - Pr\left(|X - Y| > r\right)\right)^{\ell} \tag{1}$$

$$\geq m\ell + n\left(1 - \frac{\sigma_X^2 + \sigma_Y^2}{r^2}\right)^{\ell} \tag{2}$$

for a database of size n, and using m pivots for each of ℓ pivot groups. We obtained a lower bound for the average number of distances computed using this technique. We could minimize this last equation to get the optimum values for m and ℓ.

Fixing m, we obtain the optimal number of groups ℓ as

$$\ell^{\star} = \frac{\ln m/n - \ln\ln\left(1/s\right)}{\ln\left(s\right)}, \tag{3}$$

where $s = 1 - \frac{\sigma_X^2 + \sigma_Y^2}{r^2}$.

And using this optimal ℓ we can compute the minimum cost

$$\text{cost}^{\star} \geq \frac{m\left(\ln\frac{n}{m} + \ln\ln\left(1/s\right) + 1\right)}{\ln\left(1/s\right)} \tag{4}$$

$$= m\log_{1/s}\frac{n}{m} + o(m\ln\left(1/s\right)). \tag{5}$$

[1] For a random variable Z with mean μ_Z and variance σ_Z^2, $Pr(|Z - \mu_Z| > \epsilon) < \sigma_Z^2/\epsilon^2$.

This expression is similar to the cost obtained by Chavez et al. [1] for randomly selected pivots; i.e., $cost \geq \ln n + \ln\ln(1/t) + 1/\ln(1/t)$, with $t = 1 - 2\sigma_Y^2/r^2$. This expression gets an optimal number of pivots,

$$k^\star = \frac{\ln n + \ln\ln(1/t)}{\ln(1/t)} \tag{6}$$

$$= \log_{1/t} n + o(\ln(1/t)). \tag{7}$$

Using random pivots (as analyzed by Chavez et al), the query cost depends on the database and the query radius, that is why the only way to improve the search speed is to increment ℓ. Our analysis also depends of σ_X, a parameter that we set at construction time, adjusting α. Please remember that $\sigma_X^2 = E[(d(u,p) - \mu_p)^2]$, and by construction $|d(u,p) - \mu_p| \geq \alpha$, thus $\sigma_X^2 \geq \alpha^2$. So, we have the chance to make adjustments to get better results on average, once we know the database. Also, in our cost equation, we can set the ℓ and m parameters (m depends on α) at construction time, ℓ can control the memory needed by the index. A greater m will reduce the probability of not discarding an element, even on fixed memory setups.

3 EP Table

A simple implementation of EP is a table. A set of ℓ pivot groups will be called an *EP Table*. Each group is computed as follows given the number of pivots m, and the number of instances (groups) ℓ. The construction consists on creating ℓ groups using Algorithm 1, which was sketched and analyzed in the previous section.

Algorithm 1. Randomized construction of the EP Table

Input: The input database $S = \{u_1, \ldots, u_n\}$, and the number of pivots m
Output: The set of pivots P, and the array g of n tuples $(\mathsf{piv}(u), d(u, \mathsf{piv}(u)))\ \forall u \in S$.

1: Select a random pivot p_1, $P \leftarrow P \cup \{p_1\}$
2: Initialize $g[1,n] = (p_1, d(u_1,p_1)), (p_1, d(u_2,p_1)), \ldots, (p_1, d(u_n,p_1))$
3: **for** $i = 2$ to m **do**
4: Select p_i randomly from S, $P \leftarrow P \cup \{p_i\}$
5: **for** $j = 1$ to n **do**
6: $g[j] = (p_i, d(u_j,p_i))$ **if** $|d(u_j,p_i) - \mu| > |d(u_j, \mathsf{piv}(u_j)) - \mu|$.
7: **end for**
8: **end for**

3.1 Optimizing α

In the previous construction, all parameters are assumed fixed. We can optimize the parameters using the model and the analysis described previously. The optimal α is achieved maximizing the probability of discarding an object u, which is approximated by $1 - (\sigma_X^2 + \sigma_Y^2)/r^2$. Using this expression, we observe that $\sigma_X = \sqrt{r^2 - \sigma_Y^2} \geq \alpha$.

This σ_X value implies that for high intrinsic dimensional datasets, α will be large, and it will not be useful (because it will produce a very large m). In this case, a suboptimal α value can be used and the whole performance could be improved using several pivot groups, i.e., increasing ℓ.

A better option consists in fixing ℓ controlling how much memory is used by the metric index. Once fixed ℓ, we can approximate the optimal m numerically. For this purpose we use Expression 2 as detailed in Algorithm 2. Here, the idea is to be incrementing m by one, and stop the algorithm whenever the derivative of Expression 2 becomes zero or positive. This procedure will create a single group, so it must be called ℓ times.

Algorithm 2. Numerically optimized construction of the EP-Table

Input: The input database $S = \{u_1, u_2, \cdots, u_n\}$, and the number of groups ℓ.
Output: The set of pivots P, and the array g of n tuples $(\mathsf{piv}(u), d(u, \mathsf{piv}(u)))\ \forall u \in S$.

1: Estimate σ_Y^2 and r^2.
2: Define $prev \leftarrow n$.
3: Select a random pivot p_1, $P \leftarrow P \cup \{p_1\}$
4: Initialize $g[1, n] = (p_1, d(u_1, p_1)), (p_1, d(u_2, p_1)), \cdots, (p_1, d(u_n, p_1))$
5: Define $m \leftarrow 1$
6: Compute $cost_1 = m\ell + n(1 - (\sigma_X^2 + \sigma_Y^2)/r^2)^\ell$. {At any step, σ_X^2 is computed with the current tuples in g}.
7: **while** True **do**
8: Select p_i randomly from S, $P \leftarrow P \cup \{p_i\}$
9: **for** $j = 1$ to n **do**
10: $g[j] = (p_i, d(u_j, p_i))$ **if** $|d(u_j, p_i) - \mu| > |d(u_j, \mathsf{piv}(u_j)) - \mu|$.
11: **end for**
12: $m \leftarrow m + 1$.
13: Update σ_X^2 with the current tuples in g.
14: $cost_i = m\ell + n\left(1 - (\sigma_X^2 + \sigma_Y^2)/r^2\right)^\ell$.
15: **if** $cost_i \geq cost_{i-1}$ **then**
16: stop loop
17: **end if**
18: **end while**

It is important to notice that this procedure depends heavily on the estimated values σ_Y^2 and r^2. Also, for real world databases the i.i.d.r.v. assumption can be far from true. For this reason, we add a damping constant $\beta \leq 1$ for the discarding probability in lines 6 and 14, resulting on $cost_i = m\ell + n\left(1 - \beta(\sigma_X^2 + \sigma_Y^2)/r^2\right)^\ell$. The precise value of β is dependent on how much both the database and the query set disparate from the i.i.d.r.v. assumption.

4 Experimental Performance

In this section we present the performance as a function of the dimension, and for different standard databases used by the community. As usual, vector spaces are indexed

without using the coordinates. Results are reported in both time and the fraction of the database revised for a given query.

— **Nasa** This database is a collection of 40150 vectors of dimension 40 obtained from the SISAP project (`http://www.sisap.org`). It uses L_2 as distance function.
— **Colors** The second benchmark is a set of 112682 color histograms (112-dimensional vectors) from SISAP, under the L_2 distance.
— **CoPhIR-1M** We use a subset of the CoPhIR database, of one million objects selected from the CoPhIR project [11]. Each object is a 208-dimensional vector and we use the L_1 distance.
— **RVEC** We generate random vectors in the unitary cube, in five dimensions 4, 8, 12, 16 and 20.

Each plot represents 256 nearest neighbor queries. The query object was not indexed. We used four searching algorithms as baseline for comparison.

1. The *Sequential* scan to bound the searching time when the dimension is large.
2. The *LAESA*, the standard pivot table.
3. The *List of Clusters* (LC) [10], which until now it holds the best performance for equality of memory (on the right setup).
4. The fourth baseline is a version of Navarro's *SAT* [9] built using a random order. The SAT is probably the best index having no parameters.

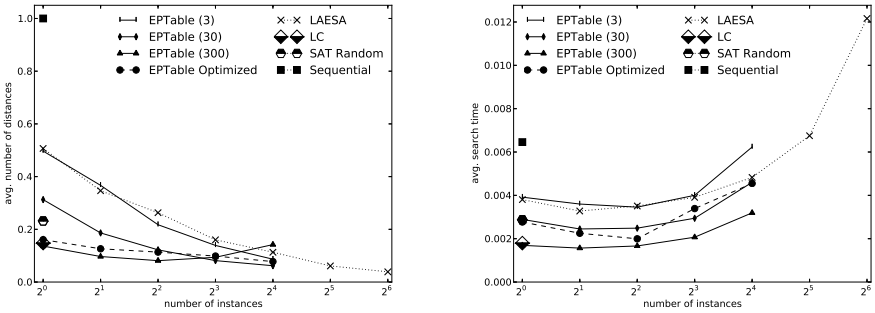
We show that with a simple tweak (essentially adjusting the number of groups) we can be faster than LAESA using as much as 64 pivots, for a fraction of the memory usage, or several times faster than LAESA using the same index size. The LC is reported as the best setup for the given dataset among the bucket sizes $16, 32, 64, 128, 256, 512$, 1024, and 2048. We tested EP Table with $3, 10, 30, 100, 300$, and 1000 pivots per instance, and up to 16 instances; we fix $\beta = 0.8$ (Section 3.1) in the numerically optimized EP Table in order to diminish the effect of the i.i.d.r.v. assumption. The optimal value of β can be very tricky, however, values around 1 produce good indexes without reducing the performance on well known distributions, and 0.8 seems to be simply enough for most setups. A detailed study on β is beyond the scope of this paper.

The algorithms were implemented in C# with the Mono framework[2]. Algorithms and indexes are available as open source software in the *natix* library[3]. All experiments were executed in a 4x quadcore Intel Xeon 2.40 GHz workstation with 32GiB of RAM, running CentOS Linux without exploiting the multicore architecture.

4.1 Performance of Our Indexes per Database

Figure 2 shows the performance for the Nasa database. The left side of the figure presents the performance as the fraction of the database revised. The EP Table is the best option, specially in setups with few instances and few pivots per instance. As shown in Figure 2(b), the search speed decreases when either the number of instances or the number of pivots per instance is large. This slowdown is because the internal computation

[2] `http://www.mono-project.org`
[3] `http://github.com/sadit/natix/`



(a) Average ratio of evaluated distances

(b) Average search time

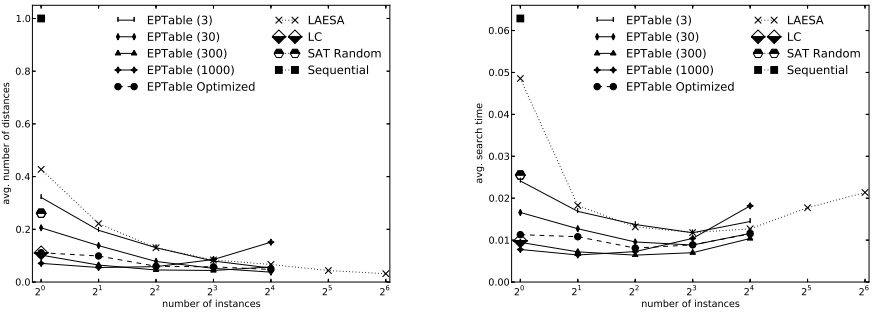
Fig. 2. Performance comparison of the nearest neighbour search in the Nasa database. We show curves only for $m = 3, 30$, and 300 in order to simplify the analysis of the figure.

increases. Please notice that the performance of the numerically optimized EP Table is very close to the best setups (on both review and real time).

In practical applications we need to optimize both memory and real query time, not the amount of computed distances. At some point, increasing the number of instances to reduce the fraction of the database revised, slowdowns the query time. Please notice that the EP Table remains as the fastest index, faster than our baselines. The speed difference is especially noticeable in setups with few instances and many pivots.

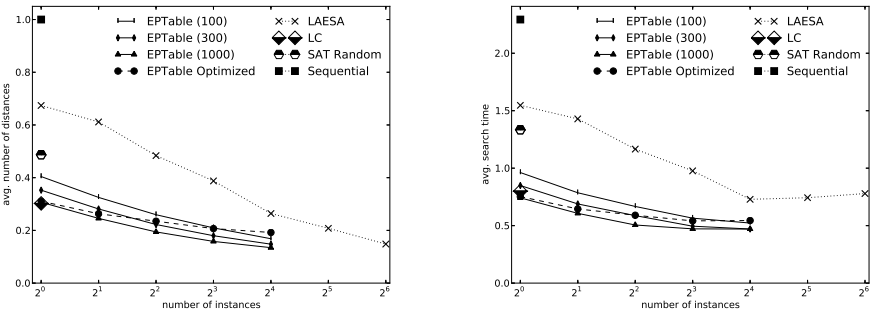
The performance of Colors is similar to the one of Nasa when the instance is numerically optimized. Figure 3(a) compares the fraction of the database revised. Notice that EP Table with few instances and several pivots per instance produce very fast indexes, surpassing the performance of LAESA, LC, and SAT. The real time, Figure 3(b) shows a similar performance for real time; however, the performance degrades for a large number of instances and pivots since the distance function is not very expensive and the internal cost becomes of matter.

The last real world database is CoPhIR-1M. Figure 4 shows how LAESA and EP Table shows a decreasing tendency as the number of instances increases. The optimum number of instances is not reached, because the database is large and the distance function is more expensive. However, both indexes increase the searching time at some point. Compared to SAT, EP Table is faster even using a single instance. As compared with LC, EP Table is faster, and equally faster with a single instance, however the LC requires more than 15 thousand centers, and EP Table is fixed with 1000 pivots, which means that EP Table is faster to construct (this is really important on databases with a costly distance function). In any case, it is possible to allow the EP Table to use more pivots per group to achieve faster indexes. In CoPhIR-1M, as in other databases, the numerically optimized EP Table is very close to the best setups of EP Table, however it decreases its performance as the number of instances increase. This can be an effect of our i.i.d. assumption. In contrast, the performance of the optimized EP Table in synthetic databases (e.g. RVEC-20) easily surpass LAESA (see Figure 5). This last fact is



(a) Average ratio of evaluated distances (b) Average search time

Fig. 3. Performance comparison of the nearest neighbour search in the Colors database. We show curves only for $m = 3, 30, 300$, and 1000 in order to simplify the analysis of the figure.



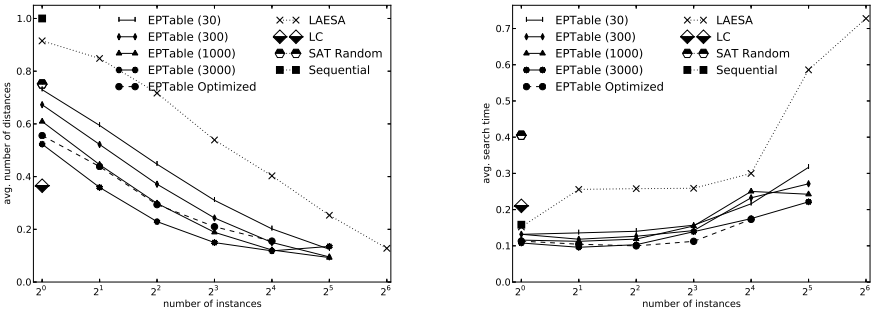
(a) Average ratio of evaluated distances (b) Average search time

Fig. 4. Performance comparison of the nearest neighbor search in the CoPhIR-1M database. We show curves only for $m = 100, 300$, and 1000 in order to simplify the analysis of the figure.

not surprising, since all the objects were generated exactly with the same random process as we have modeled. It is important to notice that for this high intrinsic dimensional dataset with a cheap distance function, our EP Table is faster than other indexes.

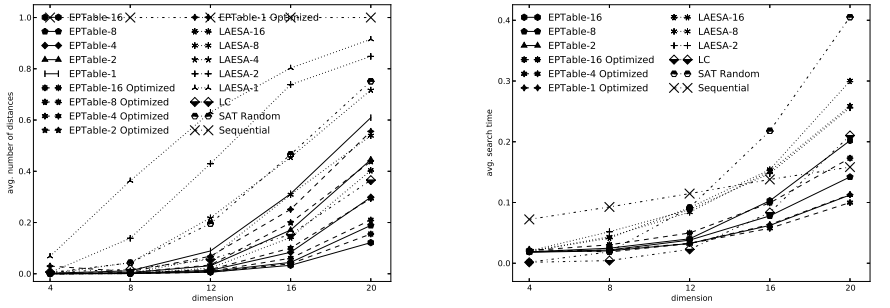
4.2 The Effect of the Dimension on the Search Performance

The curse of dimensionality stands for the odd situation where a (clever) index is slower than a plain sequential scan of the data, it is well documented in the literature [1]. The last experiment consist in testing how the indexes handle the dimensionality. We used random vectors of several dimensions in RVEC-*. Figure 6(a) shows the fraction of the database revised. Each point of EP Table represents the best performance setup in that dimension. This setup consists on six different number of pivots per instance $(3, 10, 30, 100, 300$, and $1000)$. In the same figure, the *optimized* versions of EP Table



(a) Average ratio of evaluated distances (b) Average search time

Fig. 5. Performance comparison of the nearest neighbour search in the RVEC-20 database. We show curves only for $m = 30, 300, 1000$, and 3000 in order to simplify the analysis of the figure.



(a) Average ratio of evaluated distances (b) Average search time

Fig. 6. Performance comparison of the nearest neighbor search in terms of the dimensionality (random vectors). On the right side, we ommit some curves in order to improve the readability.

shows the performance for the index numerically optimized for the desired number of instances, see Section 3.1. Also, the figure shows the performance for LAESA with several pivots (1, 2, 4, 8, and 16), implying that both LAESA and EP Table use about the same amount of memory.

It is remarkable that EP Table outperforms all indexes, and it rapidly surpasses the performance of LAESA. Please also notice that EP Table is better in the majority of the setups. Also, notice that SAT rapidly degrades its performance with increasing dimension. Remarkably, the LC is quite robust (please remember that we show only the best setup for each dimension); however, EP Table surpasses both SAT and LC in most configurations. The numerically optimized EP Table shows a similar performance than the best non optimized EP Table and surpasses the performance of LAESA, SAT, and LC (even on a single instance and large dimensions). These experimental results verify the closeness of our theoretical analysis to the real performance.

Measuring the query time (Figure 6(b)), EP Table (and the numerically optimized version) produces the fastest indexes, specially in large dimensions. However, the order of the time curves is not the same than the order found measuring the amount of evaluated distances. This also remarks that minimizing the number of distances not necessarily produces faster indexes.For example, in this experiment, the faster indexes are those having a medium number of instances.

5 Conclusions

We presented EP Table, a new index for proximity searching. It can be seen as a generalization of the pivot based indexes. The indexing technique incorporates a model which proved to be accurate if the distance distribution from the objects to the pivots is known. This has been proved with uniform multidimensional synthetic data, and we postulate that once characterized the distance distributions, the same performance boost exhibited with the synthetic database will be attained with real world databases. The resulting index may have a fixed number of instances, with each instance using essentially one machine word per database element. We have shown that EP Table surpasses the performance of both pivot tables and compact partitioning indexes.

We are working on incorporating the cost of the distance function and the size of the database in the analysis and the pivot optimization. Also, the β parameter needs a deeper study to fully understand its functionality and capabilities, this is one of our current research trends. Finally, we are studying how to substitute the table search with faster structures using the internal structure of the partition to speed up searches.

References

1. Chávez, E., Navarro, G., Baeza-Yates, R., Marroquín, J.L.: Searching in metric spaces. ACM Comput. Surv. 33(3), 273–321 (2001)
2. Samet, H.: Foundations of Multidimensional and Metric Data Structures, 1st edn. The Morgan Kaufman Series in Computer Graphics and Geometic Modeling. Morgan Kaufmann Publishers, University of Maryland at College Park (2006)
3. Zezula, P., Amato, G., Dohnal, V., Batko, M.: Similarity Search - The Metric Space Approach, 1st edn. Advances in Database Systems, vol. 32. Springer (2006)
4. Vidal Ruiz, E.: An algorithm for finding nearest neighbours in (approximately) constant average time. Pattern Recognition Letters 4, 145–157 (1986)
5. Micó, M.L., Oncina, J., Vidal, E.: A new version of the nearest-neighbour approximating and eliminating search algorithm (aesa) with linear preprocessing time and memory requirements. Pattern Recogn. Lett. 15, 9–17 (1994)
6. Bustos, B., Navarro, G., Chávez, E.: Pivot selection techniques for proximity searching in metric spaces. Pattern Recognition Letters 24(14), 2357–2366 (2003)
7. Celik, C.: Priority vantage points structures for similarity queries in metric spaces. In: Shafazand, H., Tjoa, A.M. (eds.) EurAsia-ICT 2002. LNCS, vol. 2510, pp. 256–263. Springer, Heidelberg (2002)
8. Celik, C.: Effective use of space for pivot-based metric indexing structures. In: SISAP 2008: Proceedings of the First International Workshop on Similarity Search and Applications (sisap 2008), pp. 113–120. IEEE Computer Society, Washington, DC (2008)

9. Navarro, G.: Searching in metric spaces by spatial approximation. The Very Large Databases Journal (VLDBJ) 11(1), 28–46 (2002)
10. Chávez, E., Navarro, G.: A compact space decomposition for effective metric indexing. Pattern Recogn. Lett. 26, 1363–1376 (2005)
11. Bolettieri, P., Esuli, A., Falchi, F., Lucchese, C., Perego, R., Piccioli, T., Rabitti, F.: CoPhIR: a test collection for content-based image retrieval. CoRR abs/0905.4627v2 (2009)