




# Extreme pivots: a pivot selection strategy for faster metric search

Guillermo Ruiz<sup>1</sup> · Edgar Chavez<sup>2</sup> · Ubaldo Ruiz<sup>3</sup> · Eric S. Tellez<sup>4</sup> 

Received: 12 December 2018 / Revised: 2 November 2019 / Accepted: 2 November 2019 /  
Published online: 16 November 2019  
© Springer-Verlag London Ltd., part of Springer Nature 2019

## Abstract

This manuscript presents the *extreme pivots* (EP) metric index, a data structure, to speed up exact proximity searching in the metric space model. For the EP, we designed an automatic rule to select the best pivots for a dataset working on limited memory resources. The net effect is that our approach solves queries efficiently with a small memory footprint, and without a prohibitive construction time. In contrast with other related structures, our performance is achieved automatically without dealing directly with the index's parameters, using optimization techniques over a model of the index. The EP's model is studied in-depth in this contribution. In practical terms, an interested user only needs to provide the available memory and a sample of the query distribution as parameters. The resulting index is quickly built, and has a good trade-off among memory usage, preprocessing, and search time. We provide an extensive experimental comparison with state-of-the-art searching methods. We also carefully compared the performance of metric indexes in several scenarios, firstly with synthetic data to characterize performance as a function of the intrinsic dimension and the size of the database, and also in different real-world datasets with excellent results.

**Keywords** Nearest neighbors search · Pivot-based metric indexes · Extreme pivots

---

✉ Eric S. Tellez  
estellezav@conacyt.mx; eric.tellez@infotec.mx

Guillermo Ruiz  
lgruiz@centrogeo.edu.mx

Edgar Chavez  
elchavez@cicese.mx

Ubaldo Ruiz  
uruiz@cicese.mx

<sup>1</sup> CONACyT-CentroGeo Aguascalientes, Aguascalientes, Mexico

<sup>2</sup> CICESE, Ensenada, Baja California, Mexico

<sup>3</sup> CONACyT-CICESE, Ensenada, Baja California, Mexico

<sup>4</sup> CONACyT - INFOTEC Centro de Investigación e Innovación en Tecnologías de la Información y Comunicación, Circuito Tecnopolo Norte #117 Col. Tecnopolo Pocitos II, C.P. 20313 Aguascalientes, Mexico

## 1 Introduction

Similarity search, often called proximity search, is a ubiquitous problem in computer science; it consists of retrieving objects similar to a given query from a collection; a given similarity function captures the similarity notion. A model for solving similarity search problems is the metric space model, where the similarity notion is made through a dissimilarity function, more precisely, a metric distance function, see [39]. The metric space model will be detailed in the next sections.

Another model for similarity search is that working for non-metric distance functions; these try to speed up searching taking advantage of relations found in collections beyond metric properties. They are useful in cases where metric properties cannot be guaranteed like distance functions being defined by domain experts based on their knowledge or by the necessity of using functions known to be non-metric like the Kullback–Leibler divergence or the dynamic time warping distance [33].

In particular, this manuscript focus on methods that work on general metric spaces. However, please notice that metric access methods are a large family of methods; they can range from those taking advantage of the precise representation of the data [29], to those working over general metric spaces [11].

Now, let us discuss the relevance of metric indexing. While the Naïve solution is easy to apply, i.e., exhaustively evaluate the entire collection to choose those items that satisfy the query, the use of this solution in large collections of thousands or millions of objects may be prohibitive. The metric space model is a well-studied mathematical framework for similarity search that provides the tools to speed up searching for similar objects in a collection; the standard procedure in the model is the creation of metric indexes, that is, data structures that take advantage of the underlying metric properties to speed up the search task.

The applications range is vast, and it includes the direct application to content-based information retrieval like those found in several multimedia search engines [32]; and a wide range of application domains ranging from audio and speech classification and analysis, matching of geometries and shapes, medical imaging, among others as mentioned in [33].

The *curse of dimensionality* (CoD), also known as *concentration of measure*, appears in intrinsically high-dimensional datasets with a net effect of limiting the applicability of indexing methods. It has been studied and documented in several places, for example in Shaft and Ramakrishnan [30], the authors prove that every index converges to exhaustive search as the dimension increases. On the survey of Chavez et al. [11], they introduced a definition for the dimension on metric spaces which was used and analyzed in Pestov [24–27] and in Volnyansky and Pestov [37]. The survey of Hjaltason and Samet [16] focused on practical index's performance, and how data's dimension affects it. Böhm et al. [2] showed some very nice examples of how our intuition can lead to wrong assumptions in high-dimensional spaces. It can be stated plainly as the practical impossibility to improve a sequential scan of the database with an index for intrinsically high-dimensional data.

In some applications, it is possible to use an approximate or probabilistic approach. In those methods, speed is traded for accuracy, and some errors are accepted as long as the answer can be obtained quickly; the accuracy of these is commonly measured as the quotient between the number of correctly retrieved objects and the number of requested nearest neighbors. Probabilistic methods aim at giving correct solutions most of the time (some relevant answers can be missed), while approximate solutions offer distance-based guarantees. For example, in  $\delta$ -dimensional spaces, it is possible to return an element within  $1 + \epsilon$  times the distance to the true nearest neighbor in time  $O(\delta(1 + 6\delta/\epsilon)^\delta \log n)$  [1].

## 1.1 Metric search problem

Let  $(\mathcal{X}, d)$  be a metric space; here,  $\mathcal{X}$  is the set of all valid objects (i.e., the *universe*), and  $d$  a metric function. As a metric, for any  $u, v, w \in \mathcal{X}$ , it has the following properties:

- Positivity,  $d(u, v) \geq 0$  and  $d(u, v) = 0 \iff u = v$ .
- Symmetry,  $d(u, v) = d(v, u)$ .
- Triangle inequality,  $d(u, w) + d(w, v) \geq d(u, v)$ .

We are interested in a finite subset  $S$  of  $\mathcal{X}$  (called the database). Given some query  $q \in \mathcal{X}$ , 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  $(q, r)_d = \{s \in S \mid d(s, q) \leq r\}$ . It is also useful for the notion of *k nearest neighbor* (KNN) queries; given an integer  $k > 0$ , find  $k$  nearest elements of  $S$  to  $q$ . KNN queries can be reduced to range queries using the optimal algorithm in [17]. This algorithm finds the radius  $r$  such that the query ball  $(q, r)_d$  only contains the nearest neighbor of  $q$ .

## 1.2 Contribution

This manuscript introduces the *extreme pivots* (EP), a technique that produces fast-and-small exact indexes in practice, which also performs well-theoretically. We believe that EP is a step forward in the search for efficient methods working on collections having millions of objects and represented in high intrinsic dimensions. Our contribution provides a formal method to obtain highly discriminant pivots with the assumption that we know the distribution of the queries, which can be approximated in practice using a sample of the database. Please note that this manuscript is an extended version of our previous work [28]; the new material goes from a revisited and extended state-of-the-art review, an in-depth analysis of the search cost, a new construction algorithm that improves the previous one, along with a vast experimental section performed with an improved methodology guided to show the statistical significance of comparing our indexes and other related state-of-the-art methods. Moreover, this manuscript includes an experimental analysis of the performance as a function of the database's size and also doubles the number of compared state-of-the-art methods.

## 1.3 Roadmap

This section introduces our contribution and also provides the necessary notation to follow up the rest of the manuscript. Section 2 summarizes the related state-of-the-art, while Sect. 3 is devoted to describe and analyze the extreme pivot (EP) technique. In Sect. 4, a practical implementation of EP based on pivot tables (called EPT) is presented. Section 5 gives an extensive benchmarking of our indexes and state-of-the-art exact metric indexes. We tried to compare our indexes under several setups of general interest. More precisely, we analyze the performance as a function of the dimension, the size of the dataset, and the memory usage. Finally, Sect. 7 concludes and summarizes our contribution.

## 2 Related work

The landscape of algorithms and discarding rules consists of a handful of key ideas; the interested reader can find books and surveys devoted to reviewing the algorithmic differences between indexes, for example, Chavez et al. [11], Samet [29] and Zezula et al. [39].

There are two popular approaches to metric indexing, namely *pivot-based indexes* and *compact partitions*. The pivot scheme consists in filtering the database  $S$  by using repeatedly the triangle inequality to bound the distance from an object to the query. A set of distinguished points  $\mathbb{P} = \{p_1, p_2, \dots, p_m\} \subseteq S$  (called the pivots) are used to define a filtering distance, always bounded from above by the original distance  $d$ . Let  $D(u, v) = \max_{1 \leq i \leq m} |d(u, p_i) - d(v, p_i)|$ . Using the triangle inequality, it is immediate  $D(u, v) \leq d(u, v)$ , and hence it implies  $(q, r)_d \subseteq (q, r)_D$ . All the points in  $(q, r)_D$  are compared with  $q$  using  $d$  to get  $(q, r)_d$ . More details can be found in [11].

A *compact partitioning* method arranges the database in spatially coherent, nonoverlapping regions. Most indexes of this kind are hierarchical, with a recursive rule as follows: A set of centers  $c_1, c_2, \dots, c_m \in S$  is selected per node such that every  $c_i$  is the center of a subtree  $T_i$ . The set of centers are used to partition the database such that each  $T_i$  is compact. (The sum of all items to a central object is minimized.) For example,  $u \in S$  is linked to the subtree  $T_i$  such that  $i = \arg \min_{1 \leq i \leq m} d(c_i, u)$ . The covering radius  $\text{cov}(c_i) = \max_{u \in T_i} d(c_i, u)$  is stored for each node. This construction is recursively applied. In a similar way, a query  $(q, r)_d$  is solved recursively starting from the root node: If  $d(q, c_i) \leq r$  then  $c_i \in (q, r)_d$ , and  $T_i$  must be explored if  $|d(q, c_i) - \text{cov}(c_i)| \leq r$ .

A popular complexity model has been the number of distance computations to answer a query. The rationale behind this measure is that distances are the most expensive operations in a query; however, an index optimized only to minimize the number of distance computations may have a search time (seconds) similar to an exhaustive evaluation of the whole dataset. This behavior is due to a combination of a simple distance function and an expensive indexing structure.

AESA [36] computes all distances between the objects in the database and stores them in a table. This method stores  $O(n^2)$  distances, and hence, the construction cost is of the same order. At query time, it performs a fixed number of distance computations for a database; it is also necessary to compute a linear number of arithmetic and logical operations. A linear restriction of the same idea is presented in LAESA [19], where a constant number of pivots are used, independently of the size of the database.

The number of pivots needed to keep the searching cost at a fraction of the size of the dataset depends on its *intrinsic dimension*. For searching methods, it is common to use the dimension notion of Chavez et al. [11]; here, the authors show the convenience of using  $\rho = \frac{\mu^2}{2\sigma^2}$  as a definition for the intrinsic dimension where  $\mu$  and  $\sigma$  are the mean and standard deviation of the histogram of distances in the database, that is,  $\rho$  measures the concentration around the mean; more specifically, high-dimensional datasets have large  $\mu$  values and small variances. This definition allowed them to prove that any pivot-based metric index requires at least a logarithmic (on the database size) number of pivots (randomly selected from the database) where the base of the logarithm depends on the intrinsic dimension  $\rho$  of the database. However, there is no universal way to measure the intrinsic dimensionality, and it may be dependent on the particular task; for example, Mirylenka et al. [20] use fractal dimension [35] to measure the complexity for classification problems.

As the intrinsic dimension increases, indexes need more pivots to remain competitive. However, using too many pivots increase the complexity of the index, and may lead to surpassing the cost of a sequential scan. Under this scheme, pivot selection is essential to reduce both the memory costs and the number of computed distances.

In the following paragraphs, we describe a number of paradigmatic strategies that summarize many of the available metric indexing approaches.

*Pivot Selection Strategies* Even when a random pivot's set is a good option for indexing, it has been demonstrated that pivot selection improves the overall performance. Up to the best of our knowledge, all pivot selection methods try to select good pivots *for all the items in the database*. Bustos et al. [5] introduced some pivot selection strategies. In particular, the authors propose an *incremental* selection strategy, which consists of taking a set of  $N$  candidate pivots and select the best one, say  $p_1$ . Then, from another set of  $N$  candidates, select the one  $p_2$  that makes  $\{p_1, p_2\}$  the best set of pivots among the options. This procedure is repeated until  $k$  pivots are selected. This technique needs to know in advance the proper value of  $k$ .

*The Sparse Spatial Selection (SSS)* It is a pivot-based method by Pedreira et al. [23] that automatically determines and selects the number of *essential* pivots to be used by a pivot table. SSS depends on the intrinsic dimension of the dataset and not in its size. Let  $d_{\max}$  be the maximum distance between any two objects in the database. The algorithm starts with an empty set and incrementally constructs the set of pivots as follows. An object becomes a new pivot if its distance to other pivots is higher than or equal to  $\varepsilon d_{\max}$  for a fixed  $0 < \varepsilon \leq 1$ . The goal is to choose the set of pivots to be well-distributed over space.

*The Priority Vantage Points (KVP)* Celik introduced the KVP in [6,7]; the author noticed that disregarding the number of pivots available, only some of them (the *good pivots*) contribute to the elimination of elements during a search. The author postulates that those pivots are the ones that are either close or far from the query; and therefore, each point is *visible* only by its  $k$  best pivots, and this parameter needs to be optimized. This strategy allows the KVP to use less memory than other pivot-based methods.

*The Spatial Approximation Tree (SAT)* The SAT [21] is a metric tree that creates a compact partition of the data. The root  $a$  (arbitrarily chosen) of the SAT is connected to its children  $N(a)$ , and the remaining elements of the database are assigned to the closest children of  $a$ . This process is repeated recursively with each child and its assigned elements. The set  $N(a)$  has the following two properties:

- (i) for all  $u, v \in N(a)$ , it follows that  $d(u, a) < d(u, v)$  and  $d(v, a) < d(u, v)$ , i.e., all items in  $N(a)$  are closer to  $a$  than each other.
- (ii) for all  $w$  not in  $N(a)$ , there exist  $z \in N(a)$  such that  $d(z, w) < d(z, a)$ , i.e., all items not in  $N(a)$  are closer to some item in  $N(a)$  rather than  $a$ .

For a given element  $c$ , multiple  $N(c)$  can be build. One way to do it is to take the closest element to  $c$ , call it  $x$ , and put it into  $N(c)$ . All the points closer to  $x$  than to  $c$  cannot be in  $N(c)$  (second property) so they are discarded. From the remaining elements, take the closest one to  $c$  and repeat until all the points not in  $N(c)$  are discarded. This construction of  $N(c)$  produces the SAT. Another way to get  $N(c)$  is to take the farthest point  $y$  (instead of the closest) and put it in  $N(c)$ , from the points not discarded by  $y$  take the farthest and repeat. The SAT produced in this way is called Distal SAT (DiSAT), recently described in [13].

*Vantage Point Tree* The VPT [38] is a binary tree where each element of the database is either a node or a leaf. It is constructed recursively from a random point  $p$  as the root; then, the median  $M$  is computed from remaining points, the points  $x$  such that  $d(p, x) < M$  go to the left side and the rest go to the right, and then,  $M$  is stored. For the left and right branches, the process is recursive until reaching a leaf with just one element. The search for elements at distance  $r$  or less from  $q$  is done starting from the root  $p$ ; if  $d(q, p) - r \leq M$ , the left side can be discarded; if  $d(q, p) + r < M$ , the right side will not contain an answer.

*The List of Clusters (LC)* Chavez and Navarro [10] introduced it as a robust and efficient memory index and works as follows. An element  $p$  is selected, and its  $m$  closer elements are the cluster of  $p$ . Then, from the remaining elements, another point is chosen, and its cluster is formed with the  $m$  closer elements. This procedure continues until there are no points left. The covering radius is stored in each cluster and it is used to solve the queries, i.e., if a query ball intersects with a cluster ball, then all elements in the cluster are checked to be part of the answer. The LC needs linear memory and nearly quadratic construction time for the useful combination of parameters.

*Disk-based Indexes* Prominent representatives of metric indexes working on the disk include the M-tree of Ciaccia et al. [14], a dynamic disk-based metric index. It can be seen as a compact partition index that takes similar decisions to B-Tree or R-Tree [15]. Skopal introduced the PM-Tree in [31]. The PM-Tree is an M-Tree enriched with a set of pivots. It significantly improves the search performance of the M-Tree, at the cost of extra complexity on the pivot operations and additional memory to store distance values to pivots. Another approach of secondary memory indexes is the M-Index (Novak and Batko [22]) which is a compact partition index stored in a *B+-Tree*-like structure. The idea is that centers of regions are assigned to integers, distant enough among them to store all items in each region. Then, each item is encoded to an integer based on its nearest center. This strategy reminds that of iDistance [18], which is designed for multidimensional data.

### 3 Extreme pivots

The EPT is an index based on pivots, but instead of finding ways of choosing good pivots, we focus on populating the pivots only with points that are likely to be discarded with that pivot. A pleasant consequence is that each pivot stores only a fraction of the database, reducing the amount of memory needed, which allows the use of more pivots enhancing the performance of the index.

In our previous work [28], we found that it is possible to achieve better search performances with low memory requirement if we paired each database's object with a pivot either close or far from it. In this contribution, we show a more in-depth analysis and also introduce a new variant of the construction algorithm. Regarding experimental results, this manuscript provides a more extensive experimental analysis including a set of statistical tests showing that performances of our previous and our current methods are statistically different and that the new one is significantly better for our benchmarking datasets. It is worth mentioning that this idea had been received with optimism and at least two variants were published in [12,34] which shows the potential of the technique and the need for a better understanding of the basics which make it work.

#### 3.1 A closer look at the pivot rule

Let  $\mathbb{P} = \{p_1, p_2, \dots, p_k\}$  be a set of pivots, the *pivot distance* is defined as

$$D(x, y) := \max_i \|d(x, p_i) - d(y, p_i)\| \quad (1)$$

where  $i = 1, \dots, k$ . The distance  $D$  is a lower bound of the original distance  $d(x, y)$ , i.e.,  $D(x, y) \leq d(x, y)$ , and hence

$$(q, r)_d \subseteq (q, r)_D.$$

Pivot-based methods use the pivot distance as a filter of the database: if a point is outside  $(q, r)_D$ , then it is outside  $(q, r)_d$ , and hence it can be discarded during the search. Normally, a linear scan of the database is done searching for the points that are not discarded. It is possible to use a tree or a data structure like the *spaghetti* [8] to find  $(q, r)_D$  in  $O(R \log(n))$  with  $R$  the size of  $(q, r)_D$ . The goal of a pivot selection algorithm is to reduce  $R$  to the size of  $(q, r)_d$ . For an analytical study of pivot tables, please refer to [11].

A closer look at the definition of the pivot distance is revealing. Since the maximum value over all pivots is used, then just one pivot contributes to decide that an element can be discarded. In other words, for every item in the database, only one pivot (that depends on the query) is responsible for filtering. With this observation in mind, it makes sense that for each element of the database, we seek for the pivot, increasing the chance of filtering the element for any given query. In this work, we follow this approach, and we propose a new technique which is described below.

In a pivot table each pivot *covers* the entire database; in other words, all the distances to the database objects are stored per pivot. From the observation above, we anticipate that each pivot serves only a few objects and most of the distances are idle, without being used which leads to an inefficient usage of the memory and a low probability that the index occupies a place in faster memory (e.g., RAM or the processor’s cache). The latter perspective implies slow searches.

### 3.2 The extreme approach

Instead of finding the best set of pivots  $\mathbb{P}$ , the extreme pivots (EP) approach tries to associate each item  $u$  to a good (maybe the best) available pivot  $\text{piv}(u)$  for that object. As was pointed out before, the more distant pivot from mean is used to discard the elements, thus for each  $u \in S$ , EP will try to select the pivot  $\text{piv}(u) = p$  such that

$$|d(u, \text{piv}(u)) - d(q, \text{piv}(u))| \text{ is maximized.}$$

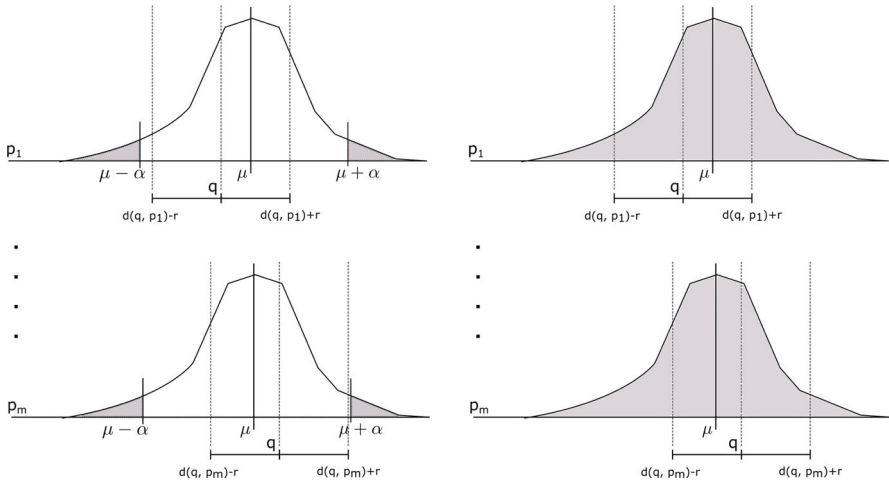
During construction time, we know  $(u, \text{piv}(u))$ , but since we do not have access to the queries, we cannot know exactly  $d(q, \text{piv}(u))$ , so the best thing we can do is estimate it. Assuming that  $q$  follows the same distribution of the data,  $d(q, p)$  is on average  $\mu_p$  of the distances from the pivot  $p$  to every object. That is,  $|d(u, p) - \mu_p|$  is maximized when  $u$  and  $p$  are either close or far from each other. Note that other pivot selection strategy can benefit from EP since we are assigning to each object the pivot most likely to discard it.

A formal definition of the above intuition follows. A *Pivot Group* (PG)  $\mathbb{P}$  is a subset of  $S$ , and its elements are called *pivots*. Every  $p \in \mathbb{P}$  has a set  $A(p) \subset S$  called the associates of  $p$  with the following properties:

- Every  $x \in A(p)$  is such that  $|\mu_p - d(p, x)|$  is maximum for all pivots in  $\mathbb{P}$  where  $\mu_p$  is the expected value of  $d(u, p)$  for all  $u \in S$ . Ties are broken arbitrarily.
- $\cup_{p \in \mathbb{P}} A(p) = S$ .
- $A(p_i) \cap A(p_j) = \emptyset$  for  $i \neq j$  for  $i, j = 1, \dots, k$ .

In order to have a mental picture and a better understanding of this process, note that the associated sets define a partition of  $S$  which induces an equivalence relation. As the number of pivots increases also does the number of equivalence classes. A bigger number of equivalence classes imply a smaller number of elements contained in each one of them.

Figure 1 illustrates the central idea of EP and its differences with other pivot-based approaches. In EP, each item  $u$  is associated with a single pivot  $\text{piv}(u)$ , and as was mentioned before,  $d(u, \text{piv}(u))$  is apart from the mean  $\mu$  of the distances from  $\text{piv}(u)$  to all



**(a)** An Extreme Pivot. From the perspective of the pivot  $p$ , the ranges  $[0, \mu - \alpha]$  and  $[\mu + \alpha, \infty)$  are populated with items from the datasets, the interior range is empty. **(b)** The behavior of other pivot tables. Each pivot sees the entire dataset no matter which pivot selection strategy is used.

**Fig. 1** Solving  $(q, r)$  with EP and other pivot tables; here,  $\mu$  is the expected value of each random variable  $p_i$  (pivot) and  $\alpha$  is a exclusion threshold of EP

elements in  $S$ . So, we can say that  $|d(u, \text{piv}(u)) - \mu| \geq \alpha$  for some value  $\alpha$ ; therefore,  $\alpha$  defines an exclusion threshold around the mean  $\mu$  for EP. Consider the query  $(q, r)$ , the expected value of  $d(q, \text{piv}(u))$  will be close to  $\mu$  (see Fig. 1a). Since  $d(u, \text{piv}(u))$  will be far from  $\mu$ , the intersection of the query ball and the intervals containing elements associated with  $\text{piv}(u)$  will be small.

Different from EP, in the other pivot-based approaches (no matter the pivot selection technique), each pivot has a full view of the dataset (see Fig. 1b). Since we consider that queries follow the same distribution,  $d(q, p)$  is close to the mean for most pivots. Therefore, a pivot table needs to contain a pivot with a low populated range, i.e., far from the mean, for each possible query to improve its filtering capabilities. For that reason, usually, a large pivot set is needed. However, note that the pivot table needs  $O(kn)$  memory where  $k$  is the number of pivots. Nonetheless, a large pivot set increase the search time since intersection's cost of the ranges is proportional to  $k$  in the worst case.

As other pivot-based techniques, in EP, the number of pivots can be large, depending on the dimensionality and size of the dataset. Nevertheless, due to the mentioned selection, each object in the database is covered by exactly *one* pivot, and hence it uses  $O(n)$  machine words, i.e., it is equivalent to a single pivot in a pivot table. The discarding power of EP can be improved using several independent pivot groups; therefore, elements are covered with more than one pivot. In this work, we propose a simple implementation of several PG using tables (EPT). We will describe it in detail in Sect. 4.

### 3.3 Bounding parameters in PG

We know that each pivot has its best-associated elements. For simplicity, let us suppose that all pivots follow an identical distribution. Therefore, let  $a$  be the probability of the event in



which an element is associated with a given pivot, that is, the points in  $A(p)$  are either close or far from  $p$ . We can assume that for every  $u \in A(p)$ , we have that  $|d(p, u) - \mu_p| > \alpha$  for every pivot  $p$ .

We present a relation between  $a$  and  $\alpha$  in the next theorem. Notice how  $a$  is closely related to the expected query set.

**Theorem 1** *Let  $X$  be the random variable describing the distance between a pivot  $p$  and an indexed item  $u$  associated with  $p$ , i.e.,  $X(u) \sim d(u, p)$ ; then, the probability of  $u$  being associated with  $p$  is  $a = \Pr(u \in A(p)) < \frac{\sigma_X^2}{\alpha^2}$ , where  $\sigma_X^2$  is the variance of  $X$  and  $\alpha$  the exclusion threshold around the mean  $\mu_X$ .*

**Proof** Using the Chebyshev’s inequality,<sup>1</sup>  $\Pr(|X - \mu_X| > \alpha)$  is upper bounded by  $\frac{\sigma_X^2}{\alpha^2}$ . Since  $|d(p, u) - \mu_p| > \alpha$  for every  $u \in A(p)$ , we have

$$\begin{aligned} a &= \Pr(u \in A(p)) < \Pr(|d(p, u) - \mu_p| > \alpha) \\ &= \Pr(|X - \mu_X| > \alpha) \end{aligned}$$

and then  $a < \frac{\sigma_X^2}{\alpha^2}$ . The proof is complete. □

This theorem shows that for a given  $X$  (fixed by the dataset and query), we can adjust  $a$  to obtain the necessary  $\alpha$  such that associated items escape from the zone around the mean from their pivot’s point of view. However, highly concentrated distributions produce small values for  $a$  (large  $\alpha$ , Fig. 1a); then, many pivots are needed. Notice that a high number of pivots do not produce memory problems in EP, yet it may increase the index’s search cost. Based on Theorem 1, the number of necessary pivots  $m$  is lower bounded as:

$$m > \frac{\alpha^2}{\sigma_X^2}, \text{ since } m = 1/a. \tag{2}$$

Please note that  $m$  could be independent of  $n$ , yet dependent on the data and query set distribution. This bound of  $m$  is highly informative, yet it does not provide a proper method to upper bound  $m$ . In the next section, we introduce the EP Table (EPT), a practical implementation of EP. The construction of EPT is not based on computing the value of  $m$  directly; it tries to optimize online the expected search cost as a function of  $m$  (detailed below). This scheme is just a possible shift in the discussion given above.

### 4 Extreme pivot table

Pivot Groups can be used to construct independent indexes; the intersection of all the independent outputs defines the final set of candidates. Each pivot in a group can only be used to discard its associates. For simplicity, we handle the index as a table, although sublinear implementations for pivots are available. An *Extreme Pivot Table* (EPT) is a collection of  $\ell$  pivot groups. Note that each point has  $\ell$  associate sets; this corresponds to each point having  $\ell$  pivots. So, when we talk about  $m$  pivots, we refer to  $m$  pivots per PG, and when we say  $\ell$  pivots, we mean pivots per point of the database.

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

### 4.1 Data structure and search algorithm

An EPT is composed of  $\ell$  PG's. Each PG is a list of tuples  $(\text{piv}(u), d(u, \text{piv}(u)))$ . More precisely, we have the following matrix:

$$T = \begin{bmatrix} (P_{1,1}, D_{1,1}) \dots (P_{1,n}, D_{1,n}) \\ \vdots \\ (P_{\ell,1}, D_{\ell,1}) \dots (P_{\ell,n}, D_{\ell,n}) \end{bmatrix}$$

where  $P_{i,j}$  stores  $\text{piv}_i(u_j)$  and  $D_{i,j}$  stores  $d(u_j, \text{piv}_i(u_j))$ . Since each Pivot Group (row) is independent,  $\text{piv}$  should be parameterized with the PG identifier to properly distinguish among the available pivot groups.

The other typical pivot tables store  $T$  just using  $D$ . So, EPT will use  $\log(m) + 1$  extra bits per item which are non-significant for most applications as the number of pivots  $m$  is fixed for a fixed data and query set. Notice that  $m$  can be fixed as the maximum  $m$  among all the pivot groups without significant memory waste.

Once  $T$  is constructed, a query  $(q, r)$  is solved using Algorithm 1. Every point in the database is tested if one of its  $\ell$  pivots can discard it using the pivot rule; if it cannot be discarded, a direct comparison with the query is made to see if it is part of the result. A k- $\text{nn}$  query is solved similarly, yet using min-priority queues of maximum size  $k$ , which means that the furthest item will be discarded whenever a closer object is found. To solve k- $\text{nn}$  queries, Algorithm 1 replaces  $r$  by the currently known covering radius  $\text{cov}$  of the result set. Formally,  $\text{cov}(R)$  is evaluated to the furthest distant in  $R$  if  $|R| \geq k$  and  $\infty$  otherwise. To make this algorithm's transformation clearer, we append the pair distance value and object to the result set (line 14). This way,  $R$  looks like a priority queue.

---

#### Algorithm 1 The search algorithm for the EP Table

---

**Input:** The database  $S$ , a query  $(q, r)$ , and the EP Table  $T$  of size  $\ell \times n$  with  $d(u, \text{piv}_i(u))$  for  $1 \leq i \leq \ell$  and every  $u \in S$

**Output:** The result set  $R$

- 1: Let  $R$  be an empty set
  - 2: Let  $H : S \rightarrow R$  be a sparse table implemented with a dictionary to cache pivot's distances to  $q$ , i.e.,  $H(p) = d(p, q)$ . For the sake of simplicity, if  $H(p)$  is not cached, then it is computed and stored in  $H(p)$ .
  - 3: **for**  $u \in S$  **do**
  - 4:   Let *review* be true.
  - 5:   **for**  $i = 1$  to  $\ell$  **do**
  - 6:     **if**  $|H(\text{piv}_i(u)) - d(u, \text{piv}_i(u))| > r$  **then**
  - 7:       Set *review* to false.
  - 8:     **break for**
  - 9:   **end if**
  - 10: **end for**
  - 11: **if** *review* **then**
  - 12:   Let  $d^* = d(u, q)$
  - 13:   **if**  $d^* \leq r$  **then**
  - 14:      $R \leftarrow R \cup \{(d^*, u)\}$
  - 15:   **end if**
  - 16: **end if**
  - 17: **end for**
-

### 4.2 Probability of discarding

Given a query  $(q, r)$ , the probability that an element  $u$  is not discarded by  $\ell$  pivots is

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

To simplify the analysis, we assume all objects to be described by independent identically distributed random variables (i.i.d.r.v). Therefore, the previous expression is rewritten as follows:

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

In order to predict the above probability, we will use the distribution of distances from the pivots to all the database elements. First, we will see the probability that a pivot would discard one of its associates.

For pivot  $p$ , let  $X$  be the random variable such that  $X(u) \sim d(u, p)$  if  $u \in A(p)$  and  $X(u) \sim 0$  otherwise, that is because  $p$  only has information for  $A(p)$ . Similarly, let  $Y$  be the random variable such that  $Y(q) \sim d(p, q)$ . The variable  $Y$  will represent the distribution of  $q$ . Typically,  $X$  and  $Y$  have a normal distribution so, let us assume they are distributed normally with mean  $\mu$ ,  $E[X] = E[Y] = \mu$ . We divide the domain of their histograms into intervals of length  $r$ . Let  $I_i$  be the interval  $[ir, (i + 1)r)$  for  $i \in \mathbb{N}$ . We are going to associate an element  $u$  with the distance  $d(p, u)$  and say that an element  $u$  is in  $I_i$  if  $d(p, u) \in I_i$ . It is convenient to express the mean in terms of the searching radius. It is safe to assume  $\mu > r$  because otherwise the query is not selective at all and the entire database would be the answer to the query. Let  $\mu = rc$  with  $c > 0$ .

$$\begin{aligned} \Pr(|d(q, p) - d(u, p)| > r) &= \Pr(u \text{ is discarded}) \\ &= \sum_{i=0}^K \Pr(u \text{ is discarded} | q \in I_i) \Pr(q \in I_i) \\ &\quad + \Pr(u \text{ is discarded} | q \in [(K + 1)r, \infty)) \Pr(q \in [(K + 1)r, \infty)) \text{ for every } K \in \mathbb{N}. \end{aligned}$$

Remember that the distance from any  $u$  in  $A(p)$  to  $\mu$  is bigger than  $\alpha$ , we will also assume that  $\mu - \alpha = br$  for some  $b > 0$ . In other words, the covering radius of each object is expressed in terms of  $b$  times  $r$ .

$$\begin{aligned} \Pr(u \text{ is discarded}) &= 2 \sum_{i=0}^{c-1} \Pr(u \text{ is discarded} | q \in I_i) \Pr(q \in I_i) \\ &= 2 \sum_{i=0}^b \Pr(u \text{ is discarded} | q \in I_i) \Pr(q \in I_i) \\ &\quad + 2 \sum_{i=b+1}^{c-1} \Pr(u \text{ is discarded} | q \in I_i) \Pr(q \in I_i). \end{aligned}$$

Note that  $\Pr(u \text{ is discarded} | q \in I_i) = 1$  when  $q$  is in  $[b(r + 1), \mu]$ , so

$$\begin{aligned} \Pr(u \text{ is discarded}) &\geq 2 \left( \frac{1}{2} + \int_0^{\mu-\alpha-3r} f_X(x) dx \right) \int_0^{\mu-\alpha+r} f_Y(x) dx \\ &\quad + 2 \int_{\mu-\alpha+r}^{\mu} f_Y(x) dx \end{aligned}$$

Let us see an example of applying the last inequality. In this assumption,  $Y$  distributes normally, so the values of the above integrals can be found in the literature. Let  $\sigma_Y^2$  be its variance, and  $\alpha = \sigma_Y + r$ . We have

$$\begin{aligned} \Pr(u \text{ is discarded}) &\geq 2 \left( \frac{1}{2} + \int_0^{\mu - \sigma_Y - 4r} f_X(x) dx \right) (0.16) + 2(0.34) \\ &\geq 2 \left( \frac{1}{2} \right) (0.16) + 2(0.34) = 0.84 \end{aligned}$$

With the last parameters, we have a probability of discarding bigger than 0.84.<sup>2</sup> *An insightful analysis.* The discarding problem can also be stated in a less precise but more insightful way as follows:

$$\Pr(|d(p, u) - d(p, q)| > r) = \Pr(|X - Y| > r)$$

then, using the Chebyshev’s inequality, 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  $\text{piv}(u)$  is

$$1 - \Pr(|X - Y| > r) \geq 1 - (\sigma_X^2 + \sigma_Y^2)/r^2.$$

This expression directly relates the distribution of the dataset, the pivots as are seen by its associated pivots, and the query set. Recall that we can always adjust  $\sigma_X^2$  increasing or decreasing  $\alpha$ ; which is also linked to the number of extreme pivots, as it is detailed below.

### 4.3 Search cost

The ultimate measure for efficiency is the search cost. In a first approach, we use the model of complexity where only the number of distance computations is counted. We discuss later the side computations since, in our approach, we can maintain them small (sublinear) as well.

Let  $s$  be the probability that an element  $u$  is not discarded. Then, the algorithm computes the following distances.

$$\text{cost} = m\ell + ns^\ell, \tag{3}$$

for a database of size  $n$ , where  $m$  is the number of pivots in the PG and  $\ell$  the number of groups. Notice that the number of groups corresponds with the number of extreme pivots seen per item, so, at this point, it is correct to say that each item is associated with  $\ell$  pivots. Since  $m$  is fixed for a given dataset (Expression 2), we focus on determining  $\ell$ . We have

$$\frac{\partial \text{cost}}{\partial \ell} = m + ns^\ell \ln(s)$$

then

$$\begin{aligned} m + ns^\ell \ln(s) &= 0 \\ \ell^* &= \frac{\ln m/n - \ln \ln(1/s)}{\ln(s)}. \end{aligned} \tag{4}$$

---

<sup>2</sup> The precise values 0.16 and 0.34 can be numerically computed or retrieved in traditional pre-computed tables for the normal distribution.

Using this optimal  $\ell$ , we can compute the minimum cost

$$\begin{aligned}
 \text{cost}^* &= m\ell^* + ns^{\ell^*} \\
 &= \frac{m \ln(\frac{m}{n}) - m \ln \ln 1/s}{\ln s} + ns^{\frac{\ln \frac{m}{n} - \ln \ln 1/s}{\ln s}} \\
 &= \frac{m \ln(\frac{m}{n}) - m \ln \ln 1/s}{\ln s} + n \left( \frac{-m}{n \ln s} \right) \\
 &= \frac{m \ln(\frac{m}{n}) - m \ln \ln 1/s - m}{\ln s} \\
 \text{cost}^* &= \frac{m \left( \ln \frac{n}{m} + \ln \ln (1/s) + 1 \right)}{\ln (1/s)} \tag{5}
 \end{aligned}$$

$$= m \log_{1/s} \frac{n}{m} + o(m \ln (1/s)). \tag{6}$$

This expression is related to the cost obtained by Chavez et al. [11] for randomly selected pivots, i.e.,

$$\text{cost} \geq (\ln n + \ln \ln (1/t)) / \ln (1/t), \tag{7}$$

with  $t = 1 - 2\sigma_X^2/r^2$ . This expression gets an optimal number of pivots,

$$k^* = \frac{\ln n + \ln \ln(1/t)}{\ln(1/t)} \tag{8}$$

$$= \log_{1/t} n + o(\ln(1/t)). \tag{9}$$

The equations looks similar; however, the main difference is that the search cost of random pivots (in Chavez et al. [11]) depends on the database and the query radius, and hence the only way to improve the search cost is by increasing the number of pivots. On the other hand, the cost of extreme pivots also depends on  $\sigma_X$ , adjusted through  $\alpha$ , a parameter that helps to adapt the index to the database and a set of queries of interest. Please remember that the variance  $\sigma_X^2 = E[(d(u, p) - \mu_p)^2]$ , and by construction  $|d(u, p) - \mu_p| \geq \alpha$ , thus  $\sigma_X^2 \geq \alpha^2$ ; see Sect. 4.2 for details. Therefore, we have the chance to make adjustments to get better results on average, by knowing how the database is distributed. In our cost equation, we also can set the  $\ell$  and  $m$  parameters ( $m$  depends on  $\alpha$ ) at construction time. Notice that  $\ell$  controls the amount of memory needed by the index, and  $m$  the discarding probability.

The index will have a better performance with higher values of  $\ell$  at the cost of memory usage, regarding distance evaluations. A large  $m$  increases the probability of a pivot discarding a given element, without increasing the index size. With  $m$  also increases the number of distances computed between the pivots and the query ( $m \times \ell$ ), and therefore, we need to adjust  $m$  accurately; each dataset has optimal values for these parameters, and they must be found or approximate to take advantage of extreme pivots. In the rest of this section, we aboard the problem of adapting an EPT to a given dataset.

### 4.4 Construction algorithms

Now, we show two algorithms creating EPT indexes. The first algorithm corresponds to a first approximation presented in [28]. It is convenient and correct, but it uses an extra parameter  $0 < \beta \leq 1$  to overcome practical issues with the estimation of the needed statistics. The second one (Algorithm 3) is part of the contribution of this manuscript and replaces  $\beta$  for

a more intuitive and self-explained parameter, *window*. The central idea of *window* is to provide a way to optimize the average performance.

The number of pivot groups and pivots per group,  $m$  and  $\ell$ , can be optimized using the model and the analysis described above. The optimal  $\alpha$  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  $\sigma_X = \sqrt{r^2 - \sigma_Y^2} \geq \alpha$ . For high intrinsic-dimensional datasets,  $\sigma_X$  becomes high; therefore, it can become useless for searching since the tightly linked  $m$  is a major term in the search cost. In this case, a suboptimal  $\alpha$  can be used, and the overall performance could be improved just increasing the number of pivot groups. The same strategy is useful when the computation of the distance function is too expensive.

The total amount of memory used can be set by fixing  $\ell$ . Once fixed, we can approximate the optimal  $m$  numerically. For this purpose, we use Eq. 3 as detailed in Algorithm 2. Here, the idea is to increment  $m$  by one at each step, and stop whenever the derivative of Expression 3 becomes zero or positive. We need to apply the above procedure  $\ell$  times since it only creates a single group.

---

**Algorithm 2** Numerically optimized construction of the EP Table; here,  $\beta$  controls the variance.

---

**Input:** The input database  $S = \{u_1, u_2, \dots, u_n\}$ , and the number of groups  $\ell$ .

**Output:** The set of pivots  $P$ , and the array  $g$  of  $n$  tuples  $(\text{piv}(u), d(u, \text{piv}(u))) \forall u \in S$ .

- 1: Estimate  $\sigma_Y^2$  and  $r^2$ .
  - 2: Select a random pivot  $p_1, P \leftarrow P \cup \{p_1\}$
  - 3: Initialize  $g[1, n] = (p_1, d(u_1, p_1)), (p_1, d(u_2, p_1)), \dots, (p_1, d(u_n, p_1))$
  - 4: Define  $\beta = 0.8$
  - 5: Let  $m \leftarrow 1$
  - 6: Let  $prev = m\ell + n(1 - \beta(\sigma_X^2 + \sigma_Y^2)/r^2)^\ell$
  - 7: **while** True **do**
  - 8:    $m \leftarrow m + 1$
  - 9:   Select  $p_m$  randomly from  $S, P \leftarrow P \cup \{p_m\}$
  - 10:   **for**  $j = 1$  to  $n$  **do**
  - 11:      $g[j] \leftarrow (p_m, d(u_j, p_m))$  **if**  $|d(u_j, p_m) - \mu| > |d(u_j, \text{piv}(u_j)) - \mu|$
  - 12:   **end for**
  - 13:   Update  $\sigma_X^2$  with the current tuples in  $g$
  - 14:    $cost \leftarrow m\ell + n \left(1 - \beta(\sigma_X^2 + \sigma_Y^2)/r^2\right)^\ell$ .
  - 15:   **if**  $cost \geq prev$  **then**
  - 16:     **stop** loop
  - 17:   **end if**
  - 18:    $prev \leftarrow cost$
  - 19: **end while**
- 

It is important to notice that this procedure depends on the estimated values of  $\sigma_Y^2$  and  $r^2$ . Also, for real-world databases, the i.i.d.r.v. assumption can be far from true. In a preliminary version of this paper [28], we introduced a constant  $\beta \leq 1$  to compute the discarding probability in lines 6 and 14. The precise value of  $\beta$  depends on how much both the database and the query set differ from the i.i.d.r.v. assumption.

In this paper, we introduce a new construction algorithm, not based on  $\beta$ . We use a more explicit manipulation of the data variance. The central idea is to consider the cost function as a *noisy signal* that should be smoothed. So, instead of doing an explicit modification of the cost like  $\beta$ , in this manuscript, we use the average of *window* evaluations. Algorithm 3 shows the new construction algorithm. Any value *window*  $> 1$  will produce a smoother signal (line 5),

**Algorithm 3** Numerically optimized construction of the EP Table using signal processing ideas to handle the variance of the data.

**Input:** The input database  $S = \{u_1, u_2, \dots, u_n\}$ , and the number of groups  $\ell$ , the *window* size

**Output:** The set of pivots  $P$ , and the array  $g$  of  $n$  tuples  $(\text{piv}(u), d(u, \text{piv}(u))) \forall u \in S$ .

```

1: Estimate  $\sigma_Y^2$  and  $r^2$ 
2: Let  $prev \leftarrow n$ , and  $m \leftarrow 0$ 
3: while True do
4:    $cost \leftarrow 0$ 
5:   for 1 to window do
6:      $m \leftarrow m + 1$ 
7:     Select  $p_m$  randomly from  $S$ ,  $P \leftarrow P \cup \{p_m\}$ 
8:     if  $m = 1$  then
9:        $g[1, n] \leftarrow (p_1, d(u_1, p_1)), (p_1, d(u_2, p_1)), \dots, (p_1, d(u_n, p_1))$ 
10:    else
11:      for  $j = 1$  to  $n$  do
12:         $g[j] = (p_m, d(u_j, p_m))$  if  $|d(u_j, p_m) - \mu| > |d(u_j, \text{piv}(u_j)) - \mu|$ 
13:      end for
14:    end if
15:    Update  $\sigma_X^2$  with the current tuples in  $g$ 
16:     $cost \leftarrow cost + m\ell + n \left(1 - (\sigma_X^2 + \sigma_Y^2)/r^2\right)^\ell$ 
17:  end for
18:  if  $cost/window - prev \geq 0$  then
19:    stop loop
20:  end if
21:   $prev \leftarrow cost/window$ 
22: end while

```

but values from 4 to 32 works well for all our datasets (see Sect. 5.1). Notice that this strategy can add  $window \times \ell$  unnecessary pivots, which can be a problem if  $window \times \ell$  is large (thousands); this effect will not appear when *window* is small. Without loss of generality, the algorithm assumes that database length is a multiple of *window*.

### 5 Performance

The definitive test for a metric index is the comparison against state-of-the-art indexes in a standard testbed. The rest of this section is dedicated to perform an extensive empirical analysis using real-world and artificial datasets commonly used in the literature, described below.

- Nasa This database is a collection of 40, 150 vectors of 20 coordinates obtained from the SISAP project (<http://www.sisap.org>). It uses  $L_2$  as distance function. A single search is completed on 0.014 s.
- Colors The second benchmark is a set of 112,682 color histograms (112-dimensional vectors) from SISAP, under the  $L_2$  distance. Each query needs 0.165 s to be solved.
- Gutenberg It is a dictionary of 2,178,587 words extracted from the Gutenberg project (<http://www.gutenberg.org>). Several languages were considered and mixed. The Levenshtein’s distance was used to measure the proximity between objects. Exhaustive verification of all items in the dictionary needs close 16.245 s to solve a single query.
- CoPhIR-1M A one million subset of the Cophir dataset [3]. Each object is a 208-dimensional vector where each coordinate is a small integer, and it uses  $L_1$  as distance. Each query needs 2.052 s to be completed.

- RVEC We generated random vectors in the unitary cube, in four dimensions 8, 12, 16, and 20, each one with one million unique items. Also, to study the effect of the database's length, we create five datasets, of dimension sixteen, containing  $10^5$ ,  $3 \times 10^5$ ,  $10^6$ ,  $3 \times 10^6$ , and  $10^7$  items, respectively. Each database is dubbed as RVEC- $\{8, 12, 16, 20\}$  for the one million case, and RVEC-16-SIZE, for the variable length dataset. We used the  $L_2$  distance with the RVEC dataset.

Table 1 lists the characteristics of each benchmark. It shows the intrinsic dimension computed as  $\rho = \mu^2/2\sigma^2$ , as described in [11]. Other characteristics are listed like the maximum, the mean, and the standard deviation of the distance function distribution. Notice how synthetic RVEC characteristics are smoothly varying with respect to the intrinsic dimensionality and how the same characteristics are quite different in real-world datasets. These different values encourage testing in both synthetic and real datasets since synthetic ones are easy to extrapolate but unrealistic, while any result on real ones is too dependent on the database itself.

Notice that our study is placed in a general metric space model, that is, every search method uses the distance as a black box; in other words, the methods cannot take advantage of the underlying structure of the objects. This restriction allows for supporting any metric distance function and enables the creation of generic search methods without taking care of a particular application domain.

Each plot represents an average of 30 independent builds of every index. In each index, we report an average of 256 nearest neighbor queries. Query objects were not indexed. In the Gutenberg case, the query set consists of 512 words randomly selected from the dataset inserting, deleting, or replacing a unique character at a random position. This edit operation ensures that a neighbor exists at a distance 1 and that it is not part of the original dataset.

We used several canonical and state-of-the-art metric indexes as the baseline for the comparison. We avoid the comparison of secondary memory indexes since our index is designed for RAM. The list of methods compared in our experimental section is:

1. Our EPT and EPT\* indexes.
2. The *Sequential* or *exhaustive* scan to bound the searching time when the dimension is large.
3. The *LAESA*, the standard pivot table. We allow it to vary the number of pivots, which impacts directly in memory usage.
4. A third baseline is the incremental selection of pivots by [5] (dubbed as *BNC*).
5. The Spatial Selection of Sparse Pivots (*SSS*) of [23].
6. The *K Vantage Pivots (KVP)* by [7].
7. The *List of Clusters (LC)* [10]. It holds excellent search times using the right setup when compared with indexes using the same amount of memory. Since LC cannot improve adding more memory, we select the best setup among a list of bucket sizes (detailed by each experiment). The search speed of LC is paid with a costly construction step.
8. The *SAT* and the *Distal SAT (DiSAT)* [13,21]. Both methods are parameterless metric indexes.
9. The *VPT* [38], a well-known parameterless index.

We report two versions of the EPT construction algorithm. The figures labeled these indexes as EPT and EPT\*, respectively, for Algorithms 2 and 3. Since both are the same index with two different optimizing algorithms, we name both as EPT whenever the context allows. We provide statistical tests to prove that both algorithms produce different indexes.

For the majority of our experiments, the indexes use the following set of parameters. EPT\* and EPT use  $\ell = 2, 4, 8, 16$ , and 32 groups; BNC and LAESA use 2, 4, 8, 16, 32,



**Table 1** Characteristics of our benchmarks. The intrinsic dimension is measured as  $\mu^2/2\sigma^2$  as defined in [11], where  $\mu$  and  $\sigma$  are the mean and variance of the distance function distribution of the dataset. The size of RVEC-16 dataset varies from  $10^5$  to  $10^7$  along the experimental section

| Name      | Size      | Explicit dimension | Distance function | Intrinsic dimension | $d_{\max}$ | $\mu$ | $\sigma$ |
|-----------|-----------|--------------------|-------------------|---------------------|------------|-------|----------|
| Nasa      | 40,150    | 20                 | $L_2$             | 3.69                | 2.82       | 1.469 | 0.446    |
| Colors    | 112,682   | 112                | $L_2$             | 8.59                | 1.38       | 0.302 | 0.032    |
| Gutenberg | 2,178,587 | –                  | Lev. dist.        | 0.74                | 30         | 9.069 | 2.485    |
| CoPhIR    | $10^6$    | 208                | $L_1$             | 19.31               | 32,682     | 0.357 | 0.096    |
| RVEC-8    | $10^6$    | 8                  | $L_2$             | 20.21               | 2.19       | 0.509 | 0.112    |
| RVEC-12   | $10^6$    | 12                 | $L_2$             | 29.78               | 2.55       | 0.546 | 0.096    |
| RVEC-16   | $10^6$    | 16                 | $L_2$             | 38.21               | 2.81       | 0.580 | 0.087    |
| RVEC-20   | $10^6$    | 20                 | $L_2$             | 45.39               | 2.95       | 0.607 | 0.082    |

and 64 pivots. SSS uses  $\varepsilon = 0.3, 0.4,$  and  $0.5$  with a maximum of 512 pivots. (The precise number of pivots depends on  $\varepsilon$  and the properties of the database.) KVP uses  $k = 2, 4, 8, 16,$  and  $32$  selected from 1024 pivots. LC uses the best setup selected among  $n/m = 1024, 512, 256, 128,$  and  $64$ . SAT, DiSAT, and VPT are not affected by parameters.

Our construction algorithms need to estimate the query set statistics. So, we assume that queries have the same distribution as that of the objects; this is a fair assumption. We emphasize that the actual query set is unknown by the optimization process of the index. Please notice that our indexes will always retrieve the correct answers, but in the case of having the actual distribution of the query set, our indexes can be better tuned.

We provide an open source implementation of all presented algorithms in C# in the *natix* library.<sup>3</sup> All experiments were executed in a  $32\times$  Intel Xeon 2.60GHz workstation with 64GiB of RAM, running Linux / CentOS 6.6 without exploiting the multicore architecture for search experiments.

The search performance is the average of 30 independent runs over each query set to reduce the variation found due to stochastic decisions in several algorithms. Also, these 30 runs are used to compute statistical significance among indexes. Finally, we report the search performance with two unitless measures, more precisely, the *search cost* and the *speedup*. The first one is the average number of distances evaluated to solve a query normalized with the number of items in the dataset, that is,

$$\text{search cost} = \frac{\text{number of distance evaluations}}{\text{the size of the dataset}}. \quad (10)$$

The second score measures the relative improvement, regarding clock time, as compared with the sequential search, which is formulated as follows:

$$\text{speedup} = \frac{\text{search time needed by the sequential scan}}{\text{search time spent by the index}}. \quad (11)$$

Note that *search time* is measured in clock time (seconds). Finally, since our comparison includes indexes having different parameters, we parameterize the performance with the index's memory, which is an image of how these parameters impact the index's complexity. This test gives us a precise idea of how much resources are needed to reach the expected performance.

## 5.1 About the parameter selection of EPT

A finely optimized index often comes at the cost of memory or construction time. EPT indexes have few parameters. We start our discussion with  $\ell$ , that is, the number of groups in the structure which controls the memory requirements.

As detailed in Sect. 4.3,  $\ell$  controls the discarding power; however, choosing a large  $\ell$  (more than 16), seriously affects the search speed since the complexity of the index becomes meaningful. The reason is that the speed function depends on both the time needed for the index to find the set of candidates and the direct comparison of the resulting set. Being aware of the real cost of the index is far more complicated than just focusing on the reduction in the number of evaluated distances; however, it cannot be ignored for applications. In the rest

<sup>3</sup> <https://github.com/sadit/natix/>.

of this experimental section, we describe the effect of  $\ell$  in the performance of EPT. As a rule of thumb, low-cost distances should use a few groups to keep the cost of the index low; expensive distances benefit more from large  $\ell$  values since the distance functions dominate the search speed of the index.

Once the desired memory is specified, the precision of the optimizing procedure can be seen as a parameter; we created different strategies for EPT and EPT\*. When the EPT was presented in [28], a parameter called  $\beta$  was used. Due to space restrictions, in that seminal paper, it was used  $\beta = 0.8$  without giving details about this selection. The reason of fixing  $\beta$  to 0.8 is depicted in Table 2. Notice how the  $\beta$  is dependent on the particular dataset, but setups are better than using  $\beta = 1.0$  for almost any case, so letting a smaller value works. In particular, 0.8 is better in the majority of our benchmarks. In any case, notice how the variance is relatively low, that is, the reason to fix  $\beta = 0.8$  in [28] and the rest of this document.

In this work, we introduce the EPT\* having *window* as an optimizing parameter. The proper value of *window* should be computed taking into account that  $window \times \ell$  extra distances will be evaluated at each search. Table 3 illustrates the effect of *window* in EPT\*. We can see how large *window* values show better search times until the internal cost becomes too large, and then the search speed decreases. Notice that the bad effects of large *window* are less noticeable in massive datasets than in smaller ones because  $\ell \times window/n$  remains relatively low. However, depending on the dataset, the construction time could be more extensive because our construction algorithm converges slower as *window* increases (Algorithm 3). As in the EPT, we encourage simplicity, so we fix  $window = 16$  for the rest of the manuscript. This simplification adds  $16\ell$  distance evaluations in the worst case.

**Table 2** Effect of the  $\beta$  parameter for EPT  $\ell = 4$  on five benchmarks

| $\beta$    | Speedup      | S. Cost       | $\beta$    | Speedup     | S. Cost       | $\beta$    | Speedup     | S. Cost       |
|------------|--------------|---------------|------------|-------------|---------------|------------|-------------|---------------|
| Colors     |              |               | Nasa       |             |               | CoPhIR     |             |               |
| 0.6        | 7.09         | 0.0563        | 0.6        | 2.17        | 0.1148        | <b>0.6</b> | <b>5.97</b> | <b>0.1771</b> |
| 0.7        | 7.27         | 0.0530        | 0.7        | 2.11        | 0.1195        | 0.7        | 5.92        | 0.1788        |
| <b>0.8</b> | <b>7.30</b>  | <b>0.0595</b> | <b>0.8</b> | <b>2.24</b> | <b>0.1265</b> | 0.8        | 5.09        | 0.1809        |
| 0.9        | 6.84         | 0.0566        | 0.9        | 2.01        | 0.1268        | 0.9        | 5.71        | 0.1856        |
| 1.0        | 7.15         | 0.0555        | 1.0        | 2.06        | 0.1188        | 1.0        | 5.78        | 0.1833        |
| $\beta$    | Speedup      | S. Cost       | $\beta$    | Speedup     | S. Cost       |            |             |               |
| Gutenberg  |              |               | RVEC-16-1M |             |               |            |             |               |
| 0.6        | 16.80        | 0.0154        | 0.6        | 2.09        | 0.0350        |            |             |               |
| 0.7        | 16.54        | 0.0154        | 0.7        | 2.22        | 0.0412        |            |             |               |
| <b>0.8</b> | <b>17.01</b> | <b>0.0152</b> | <b>0.8</b> | <b>2.35</b> | <b>0.0370</b> |            |             |               |
| 0.9        | 16.16        | 0.0158        | 0.9        | 2.26        | 0.0336        |            |             |               |
| 1.0        | 16.69        | 0.0152        | 1.0        | 2.20        | 0.0365        |            |             |               |

Search Cost and Speedup are defined in Eqs. 10 and 11, respectively. Best rows in terms of speedup are marked in bold

**Table 3** Performance of EPT as a function of *window*

| Window      | Speedup     | Search time (s) | Construction time (s) | $\ell \times window$ |
|-------------|-------------|-----------------|-----------------------|----------------------|
| $\ell = 4$  |             |                 |                       |                      |
| 1           | 5.82        | 0.0846          | 4.1299                | 4                    |
| 2           | 6.79        | 0.0634          | 8.5356                | 8                    |
| 4           | 8.17        | 0.0495          | 15.1299               | 16                   |
| 8           | 8.50        | 0.0453          | 25.2799               | 32                   |
| 16          | 8.75        | 0.0448          | 30.5726               | 64                   |
| <b>32</b>   | <b>9.01</b> | <b>0.0451</b>   | <b>156.2616</b>       | <b>128</b>           |
| 64          | 9.00        | 0.0446          | 167.8938              | 256                  |
| 128         | 8.81        | 0.0491          | 162.5611              | 512                  |
| 256         | 8.10        | 0.0588          | 203.8279              | 1024                 |
| $\ell = 8$  |             |                 |                       |                      |
| 1           | 6.40        | 0.0562          | 7.6865                | 8                    |
| 2           | 7.06        | 0.0514          | 22.2632               | 16                   |
| 4           | 7.99        | 0.0399          | 42.6136               | 32                   |
| 8           | 8.25        | 0.0396          | 54.0796               | 64                   |
| 16          | 8.14        | 0.0404          | 54.1276               | 128                  |
| <b>32</b>   | <b>8.53</b> | <b>0.0389</b>   | <b>87.4186</b>        | <b>256</b>           |
| 64          | 8.76        | 0.0424          | 129.9261              | 512                  |
| 128         | 8.33        | 0.0485          | 151.6394              | 1024                 |
| 256         | 7.39        | 0.0649          | 169.3116              | 2048                 |
| $\ell = 16$ |             |                 |                       |                      |
| 1           | 6.18        | 0.0417          | 11.9085               | 16                   |
| 2           | 6.10        | 0.0387          | 3.7159                | 32                   |
| 4           | 6.50        | 0.0369          | 30.8376               | 64                   |
| 8           | 7.43        | 0.0339          | 50.9435               | 128                  |
| <b>16</b>   | <b>7.93</b> | <b>0.0320</b>   | <b>69.8743</b>        | <b>256</b>           |
| 32          | 7.51        | 0.0335          | 81.7329               | 512                  |
| 64          | 7.68        | 0.0379          | 81.5890               | 1024                 |
| 128         | 7.30        | 0.0544          | 125.1052              | 2048                 |
| 256         | 5.91        | 0.0834          | 139.4746              | 4096                 |

Other datasets have a similar general behavior  
 Bold results are the best

### 5.2 The effect of the dimension on the search performance

Figure 2 compares the performance of the methods when the dimension increases, fixing the size of the dataset. For this experiment, we use synthetic vectors, having one million items per database. We dedicated a pair of figures for each dimension. The set of figures show the search cost (left) and speedup (right) comparisons; the search cost is measured as the number of evaluations of the distance function normalized with the size of the database. In both figures, the horizontal axis is the memory usage of the index. Each index and its configuration determine the memory footprint; these configurations are detailed at the beginning of Sect. 5.

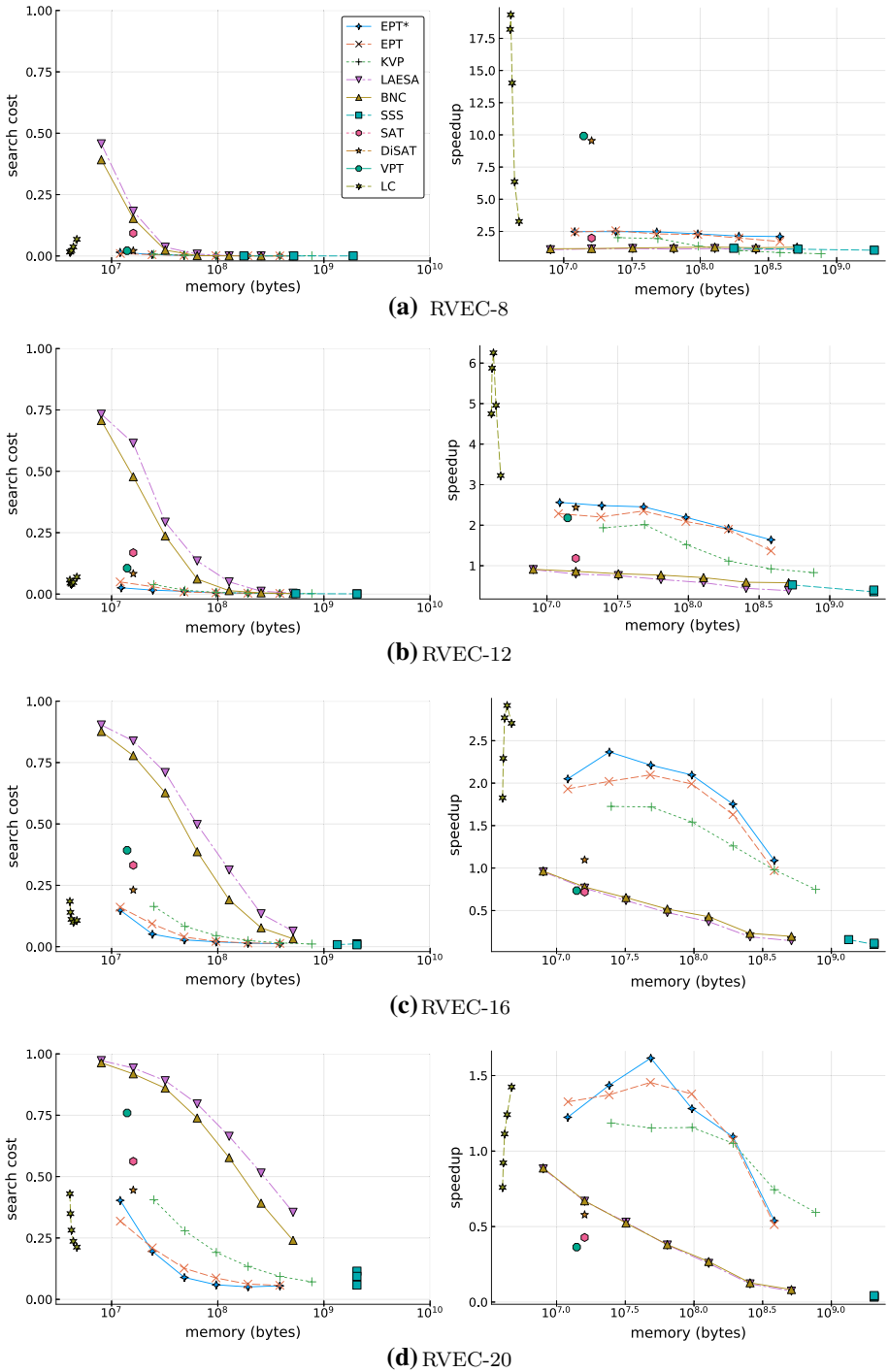


Fig. 2 Search performance comparison for datasets with different dimensions; each dataset contains one million objects. Lower values are better for search cost (left side), while higher are better for speedup (right side)

The leftmost column of figures illustrates how the search cost is increased with the dimension from almost free searches in low dimensions to close to an exhaustive evaluation of the dataset in higher dimensions. These figures also show how increasing memory (adding pivots) is an effective way to reduce the number of evaluations of the distance function. Nonetheless, the real effect of reducing the number of evaluations is the impact on the speedup (right column of figures); the real searching time is increased as the index becomes complex, that is, it incorporates larger machinery to discard objects it produces slower indexes as a net effect. This effect is easy to see on pivot indexes with a large number of pivots as SSS.

Figure 2a compares indexes for the eight-dimensional benchmark. In the left, we can observe that almost any index has a low cost, except for LAESA and BNC in their low memory configurations. Regarding speedup, the best performing is the LC, followed by other small indexes like VPT and DiSAT. After these indexes, the pivot-based methods appear in the rank; the better pivot methods are EPT\* and EPT. Figure 2b shows the comparison among searching methods for the 12-dimensional dataset. On the search cost, it is possible to see a small increase in the cost; here, we can see that EPT\* and EPT are more or less robust to the increase in the dimension since they remain low in the search cost and high in the speedup. The best indexes are those generated by LC; please notice that VPT and DiSAT begin to degrade its performance.

Figure 2c shows the performance comparison for the 16-dimension benchmark. Here, the search cost is significantly higher than that of RVEC-8 and RVEC-12. LC remains to have an excellent performance, but it is clear that EPT\* and EPT are catching up, yet using more memory. The VPT and DiSAT are left behind for pivot-based indexes. Regarding speedup, LC remains as the best index but EPT\*, EPT and KVP are now the best pivot-based indexes and are competitive with LC's speed up. Other indexes are below the exhaustive search time. However, please recall that the distance evaluation of 16-dimensional distances is pretty fast and that real-world datasets have higher cost distance evaluations.

The comparison for our 20-dimensional benchmark is shown in Fig. 2d; it is notorious that most indexes have a poor performance in terms of search cost (left). The EPT\*, EPT, KVP, and LC perform the better in both search cost and speedup. Nonetheless, from the dynamic of these experiments, it is possible to see that all these indexes and others based on the same framework degrade its net performance as the dimension increases. The effect can be reduced using sophisticated techniques like the EPT family of indexes, but all of them are negatively affected by the dimension increases.

One important aspect of EPT is that it is easy to configure and has a small construction time as compared with LC (see Table 6). The LC needs an expensive and handcrafted optimization while having a quadratic construction time. Moreover, after reaching the optimum, the LC cannot be improved further by using more memory, while our proposal can use as much memory as it fits in the system's RAM.

### 5.2.1 Statistical significance of the comparison

Table 4 shows the  $p$  values of the statistical test applied to the achieved speedups on RVEC-1M benchmarks, for different dimensions. Each configuration was evaluated 30 times. The table shows the  $p$  values of six indexes; among the six methods, we selected our contribution EPT\* and EPT to prove that Algorithms 2 and 3 are different; we also included KVP since it also selects near and far pivots to create a sparse pivot table, as EPT can be seen; LC is also considered since its excellent performance and simplicity. Similarly, VPT and DiSAT are also included due to its simplicity and performance.

**Table 4** Statistical hypothesis tests for speedup performance on varying dimension

|             | EPT*       | EPT        | KVP        | LC         | DiSAT      | VPT         |             | EPT*       | EPT         | KVP        | LC          | DiSAT       | VPT        |
|-------------|------------|------------|------------|------------|------------|-------------|-------------|------------|-------------|------------|-------------|-------------|------------|
| (a) RVEC-8  |            |            |            |            |            |             | (b) RVEC-12 |            |             |            |             |             |            |
| EPT*        | <b>1.0</b> | 0.0        | 0.0        | 0.0        | 0.0        | 0.0         | EPT*        | <b>1.0</b> | 0.0         | 0.0        | 0.0         | <b>0.34</b> | 0.0        |
| EPT         | –          | <b>1.0</b> | 0.0        | 0.0        | 0.0        | 0.0         | EPT         | –          | <b>1.0</b>  | 0.0        | 0.0         | 0.01        | 0.01       |
| KVP         | –          | –          | <b>1.0</b> | 0.0        | 0.0        | 0.0         | KVP         | –          | –           | <b>1.0</b> | 0.0         | 0.0         | 0.0        |
| LC          | –          | –          | –          | <b>1.0</b> | 0.0        | 0.0         | LC          | –          | –           | –          | <b>1.0</b>  | 0.0         | 0.0        |
| DiSAT       | –          | –          | –          | –          | <b>1.0</b> | <b>0.11</b> | DiSAT       | –          | –           | –          | –           | <b>1.0</b>  | 0.0        |
| VPT         | –          | –          | –          | –          | –          | <b>1.0</b>  | VPT         | –          | –           | –          | –           | –           | <b>1.0</b> |
|             | EPT*       | EPT        | KVP        | LC         | DiSAT      | VPT         |             | EPT*       | EPT         | KVP        | LC          | DiSAT       | VPT        |
| (c) RVEC-16 |            |            |            |            |            |             | (d) RVEC-20 |            |             |            |             |             |            |
| EPT*        | <b>1.0</b> | 0.0        | 0.0        | 0.0        | 0.0        | 0.0         | EPT*        | <b>1.0</b> | <b>0.42</b> | 0.0        | 0.0         | 0.0         | 0.0        |
| EPT         | –          | <b>1.0</b> | 0.0        | 0.0        | 0.0        | 0.0         | EPT         | –          | <b>1.0</b>  | 0.0        | 0.0         | 0.0         | 0.0        |
| KVP         | –          | –          | <b>1.0</b> | 0.0        | 0.0        | 0.0         | KVP         | –          | –           | <b>1.0</b> | <b>0.86</b> | 0.0         | 0.0        |
| LC          | –          | –          | –          | <b>1.0</b> | 0.0        | 0.0         | LC          | –          | –           | –          | <b>1.0</b>  | 0.0         | 0.0        |
| DiSAT       | –          | –          | –          | –          | <b>1.0</b> | 0.0         | DiSAT       | –          | –           | –          | –           | <b>1.0</b>  | 0.0        |
| VPT         | –          | –          | –          | –          | –          | <b>1.0</b>  | VPT         | –          | –           | –          | –           | –           | <b>1.0</b> |

We use RVEC-\*-1M benchmarks and Wilcoxon signed-rank to compute *p* values; values higher than 0.05 are in bold face

For the hypothesis test, we used the Wilcoxon signed-rank test since it is nonparametric, and we cannot ensure a particular distribution on our result data; please note that Wilcoxon is a paired difference test, and outputs the probability that the two samples were taken from the same distribution, i.e., which may imply that indexes are the same in practice. Please notice that indexes use different configurations, and the test requires samples of equal size; therefore, we always use configurations with the lesser memory to adjust the size of the sample. For instance, this decision means that VPT and DiSAT compare only with that EPT\* index with the smaller memory usage in the set of configurations.

Table 4a, b shows *p* values for RVEC-8-1M and RVEC-12-1M, respectively. Here, we found that almost all of these indexes perform different except DiSAT and VPT, with a small *p* of 0.11 for the eight-dimensional dataset. In the case of RVEC-12-1M, the smallest EPT\* and DiSAT also perform similarly with a *p* = 0.34.

Table 4c shows that all indexes perform different. For Table 4d, EPT\* and EPT perform similarly since the null hypothesis cannot be rejected; the same is true for KVP and LC, but in particular, this result seems to be induced by the variability of LC.

### 5.3 When the size of the database changes

Now, we study the performance when the size of the database changes. The sizes are  $10^5$ ,  $3 \times 10^5$ ,  $10^6$ ,  $3 \times 10^6$ , and  $10^7$  vectors. We used the synthetic database of dimension 16 for these experiments.

Besides studying the performance as *n* grows, this experiment also tries to capture the performance on large datasets where the memory is a scarce resource, so we limit the amount of memory of pivot-based indexes LAESA, BNC, EPT, and KVP to use at most 4 pivots or groups. The present experiment can be seen as a large and high-dimensional benchmark.

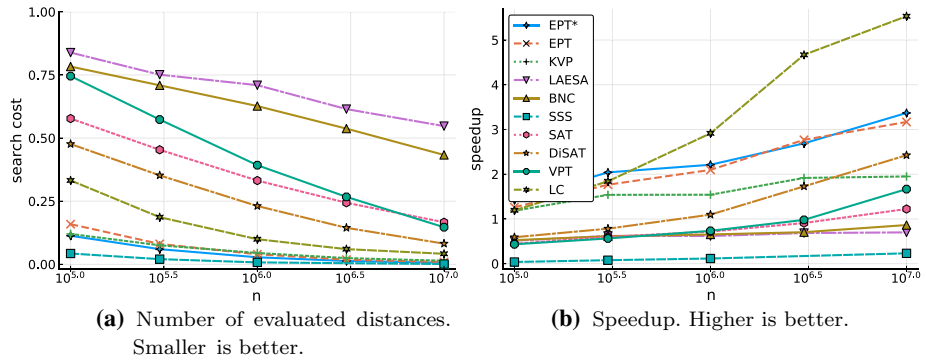


Fig. 3 Performance on RVEC of 16 dimensions and increasing  $n$

Table 5 Statistical test for RVEC-16- $10^5$

|       | EPT*        | EPT         | KVP         | LC          | DiSAT       | VPT         |
|-------|-------------|-------------|-------------|-------------|-------------|-------------|
| EPT*  | <b>1.00</b> | 0.00        | 0.00        | 0.00        | 0.00        | 0.00        |
| EPT   | –           | <b>1.00</b> | 0.01        | <b>0.09</b> | 0.00        | 0.00        |
| KVP   | –           | –           | <b>1.00</b> | <b>0.76</b> | 0.00        | 0.00        |
| LC    | –           | –           | –           | <b>1.00</b> | 0.00        | 0.00        |
| DiSAT | –           | –           | –           | –           | <b>1.00</b> | 0.00        |
| VPT   | –           | –           | –           | –           | –           | <b>1.00</b> |

Bold text marks  $p$  values higher than 0.05. Larger RVEC-16 benchmarks show that all tested indexes have different performances as  $n$  increases

In Fig. 3, each point of the curves corresponds, from left to right, to an increasing number of items in the database. Figure 3b shows the speedup. Here, the EPT and EPT\* are surpassed only by the LC; however, they are the indexes computing fewer distances (Fig. 3a). Again, it is necessary to mention that  $L_2$  over 16 coordinates can be seen as high dimensionality, but it is still computed very fast. Notice how VPT, SAT, LAESA, and BNC are reduced to a sequential scan, the latter mostly because the memory usage is limited. In this scenario, the EPT is beaten just by the optimum LC in speed, but remember that our method is simpler to configure and less expensive to build. The preprocessing time will be detailed shortly.

Table 5 shows the  $p$  values of the Wilcoxon signed-rank test applied to RVEC-16- $10^5$  benchmark. We can see that almost all indexes are statistically different except KVP and LC that are pretty similar at this database’s size; also, there is a small similarity for EPT and LC. For larger datasets,  $3 \times 10^5$ ,  $10^6$ ,  $3 \times 10^6$ , and  $10^7$ , we found that all indexes are statistically different and we omit their listing, i.e.,  $p$  values of the upper-triangle matrix are 0 for two decimal values.

*About preprocessing time.* Table 6 shows the construction time for a number of methods of Fig. 3. Please remember that DiSAT, SAT, VPT, and LC cannot be improved using more memory. Despite this, LC’s performance surpasses almost any other index for increasing  $n$ . This performance is because LC trades preprocessing time by search speed, adjusting the size of the clusters, i.e., setting  $n/m$ . In particular, Table 6 shows the cost of LC maximizing the Speedup. To fix ideas, when  $n = 10^7$ , we found that  $n/m = 64$  is the best setup; however, if we optimize to decrease the number of evaluated distances, the best parameter is  $n/m = 16$ . So, the optimum LC needs 57 days. Since LC is naturally sequential, a parallel optimizer



**Table 6** Construction time as a function of  $n$ 

| Method | Construction time (grouped by size) |                 |         |
|--------|-------------------------------------|-----------------|---------|
|        | $10^7$                              | $3 \times 10^6$ | $10^6$  |
| LC     | 7 days 4 h                          | 3 days 16 h     | 6 h     |
| EPT*   | 1 h 50 min                          | 30 min          | 5 min   |
| EPT    | 46 min                              | 6 min           | 2 min   |
| KVP    | 40 min                              | 11 min          | 5 min   |
| SAT    | 13 min                              | 4 min           | < 1 min |
| DiSAT  | 26 min                              | 9 min           | 2 min   |
| VPT    | 6 min                               | 1 min           | < 1 min |

We used random vectors in 16 dimensions for this experiment. Please notice that EPT and EPT\* have a slower preprocessing step in high dimensions than in low dimensions because it is translated as a slow convergence in its construction algorithms

could do the job in half the time, creating all LC's at the same time; however, this implies to have a hint of on  $n/m$ . In any case, it is too expensive for practice, especially on large datasets and costly distance functions.

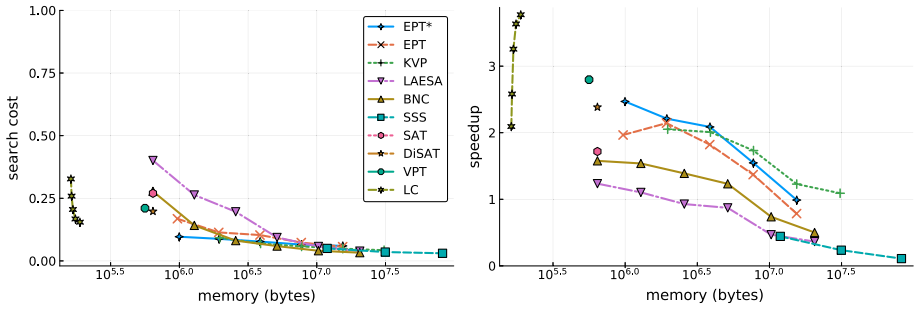
We selected the best setup among a large number of configurations, described at the beginning of this section. For KVP, the number of near and far pivots should be specified, and also the number of available pivots. The table shows the construction time to select two nearest and two furthest pivots from a total of 1024. The parameters can also be called in parallel, and even when it could be easily implemented, it could be extremely costly without a hint.

In the case of EPT and EPT\*, we suggest to determine the amount of memory to be  $\ell \leq 16$ ; in particular, we use  $\ell = 4$  for this section. We also fixed  $\beta = 0.8$  and  $window = 16$ , respectively, for EPT and EPT\*. The idea is that even when  $\beta$  and  $window$  determines the EPT's search performance, they are in fact, meta-parameters leading the optimizing algorithm to find the proper number of pivots per group. So, whenever we fix  $\ell$  to some value, we are just simplifying the optimizing algorithm, such that it only needs to find the proper number of pivots. Since EPT's groups are independent of each other, Table 6 shows the time of construction of a single group. We spawn one thread per group.

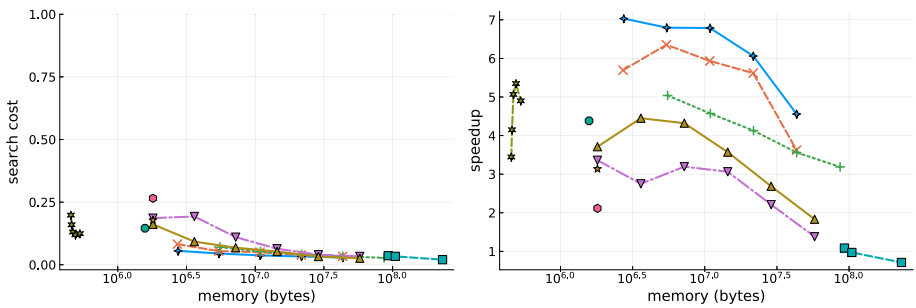
## 5.4 Performance of our indexes on real databases

In this section, we explore the performance of exact metric indexes in real datasets. These datasets were selected to be different; the idea is to give a comprehensive perspective of the methods, and how they could perform on different domains. In these benchmarks, we do not limit the memory usage, excepting for SSS which is limited to use at most 512 pivots (i.e., at most 4K per object) due to practical restrictions.

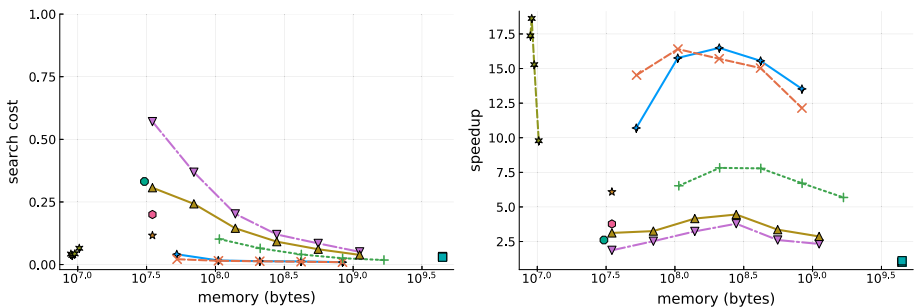
The Nasa database is the smallest, and almost any method has a relatively good performance on it. In Fig. 4a, we show how much faster are the methods compared to sequential search. Not all the indexes profit from using more memory because the internal mechanism to navigate the index has complexity comparable to an exhaustive scan. Here, LC and VPT are the best performing indexes. Notice how these two methods are not the best ones for the number of evaluated distances (left side). This place corresponds to SSS, but it is the worst performing from the speedup (right figure), and the memory usage perspective. Our EPT and EPT\* show a good trade-off; this is important to be noticed since datasets having



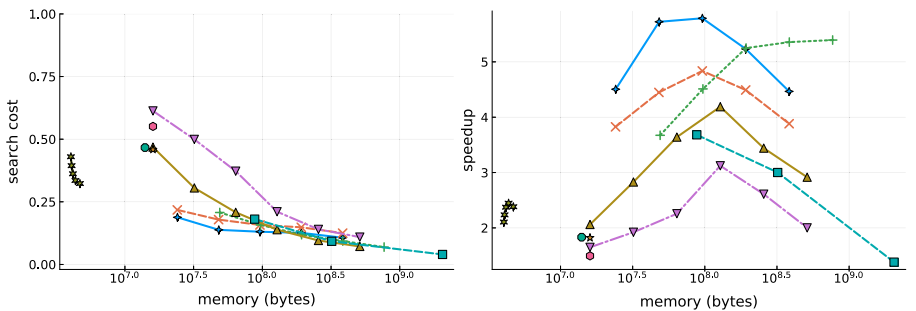
(a) Performance comparison on Nasa,  $n = 40,150$ ,  $L_2$ , 20 coordinates.



(b) Performance comparison on Colors,  $n = 112,682$ ,  $L_2$ , 112 coordinates.



(c) Performance comparison on Gutenberg,  $n = 2,178,587$ , edit distance.



(d) Performance comparison on CoPhIR-1M,  $n = 1,000,000$ ,  $L$ , 208 coordinates.

**Fig. 4** Performance comparison of state-of-the-art exact metric indexes for nearest neighbor search on datasets generated by real-world processes. Figures on the left side show the search cost, and figures on the right side show the speedup

**Table 7** Wilcoxon signed-rank  $p$  values for our real-world benchmarks for speedup performance

|            | EPT*        | EPT         | KVP         | LC          | DiSAT       | VPT         |               | EPT*        | EPT         | KVP         | LC          | DiSAT       | VPT         |
|------------|-------------|-------------|-------------|-------------|-------------|-------------|---------------|-------------|-------------|-------------|-------------|-------------|-------------|
| (a) Nasa   |             |             |             |             |             |             | (b) Colors    |             |             |             |             |             |             |
| EPT*       | <b>1.00</b> | 0.00        | 0.00        | 0.00        | <b>0.53</b> | 0.01        | EPT*          | <b>1.00</b> | 0.00        | 0.00        | 0.00        | 0.00        | 0.00        |
| EPT        | –           | <b>1.00</b> | <b>0.97</b> | 0.00        | 0.00        | 0.00        | EPT           | –           | <b>1.00</b> | 0.00        | 0.00        | 0.00        | 0.00        |
| KVP        | –           | –           | <b>1.00</b> | 0.00        | 0.00        | 0.00        | KVP           | –           | –           | <b>1.00</b> | 0.00        | 0.00        | 0.00        |
| LC         | –           | –           | –           | <b>1.00</b> | 0.01        | 0.00        | LC            | –           | –           | –           | <b>1.00</b> | 0.00        | 0.00        |
| DiSAT      | –           | –           | –           | –           | <b>1.00</b> | 0.00        | DiSAT         | –           | –           | –           | –           | <b>1.00</b> | 0.00        |
| VPT        | –           | –           | –           | –           | –           | <b>1.00</b> | VPT           | –           | –           | –           | –           | –           | <b>1.00</b> |
|            | EPT*        | EPT         | KVP         | LC          | DiSAT       | VPT         |               | EPT*        | EPT         | KVP         | LC          | DiSAT       | VPT         |
| (c) CoPhIR |             |             |             |             |             |             | (d) Gutenberg |             |             |             |             |             |             |
| EPT*       | <b>1.00</b> | 0.00        | 0.00        | 0.00        | 0.00        | 0.00        | EPT*          | <b>1.00</b> | <b>0.76</b> | 0.00        | <b>0.11</b> | 0.00        | 0.00        |
| EPT        | –           | <b>1.00</b> | 0.00        | 0.00        | 0.00        | 0.00        | EPT           | –           | <b>1.00</b> | 0.00        | <b>0.63</b> | 0.00        | 0.00        |
| KVP        | –           | –           | <b>1.00</b> | 0.00        | 0.00        | 0.00        | KVP           | –           | –           | <b>1.00</b> | 0.00        | 0.00        | 0.00        |
| LC         | –           | –           | –           | <b>1.00</b> | 0.00        | 0.00        | LC            | –           | –           | –           | <b>1.00</b> | 0.00        | 0.00        |
| DiSAT      | –           | –           | –           | –           | <b>1.00</b> | <b>0.78</b> | DiSAT         | –           | –           | –           | –           | <b>1.00</b> | 0.00        |
| VPT        | –           | –           | –           | –           | –           | <b>1.00</b> | VPT           | –           | –           | –           | –           | –           | <b>1.00</b> |

Bold text marks those values where  $p \geq 0.05$

similar properties to Nasa could find a better index in EPT, especially if the distance function is expensive.

In Fig. 4b, we see that in the Colors database the EPT and EPT\* are the best choices, in both the number of computed distances and the speedup. In this database, the cost of the distance function is higher than in our previous benchmarks. Thus, the number of computed distances has a more significant component in the total search time. In some way, this is what is expected in practical applications using complex objects and complex distance functions.

The Gutenberg database experiments took more time than the others. The results are shown in Fig. 4c. Here, we found that the optimal setup of LC has a higher speedup than the rest of the indexes. However, its construction cost is many times larger than the one needed by any of the other methods. After LC, EPT and EPT\* are better options. (Also they are the best performing when the cost is the number of computed distances.) Recall that Gutenberg uses edit distance to find similar words in a dictionary. If distances become slightly more expensive, e.g., larger words or different comparison functions for the word’s symbols, our EPT could outperform better than others since it performs fewer evaluations of the distance function.

Figure 4d shows that the best performance is with the EPT on CoPhIR-1M. Also, the EPT outperforms the alternatives for speedup, as shown at the right side of the figure. The LC index is useful, but it cannot perform better than most pivot-based indexes, mainly because performing  $L_2$  over two 208-dimensional vectors is a costly operation. Other compact partition indexes like SAT, DiSAT, and VPT are also surpassed. One observation is that memory-limited indexes can be used in mobile devices. The smartphones and tablets processors are powerful enough to run these indexes and, since their memory is limited, the EPT is the best option for them. On the other hand, high-end workstations and servers can use more massive databases, without relying on secondary memory.

Table 7 shows the statistical test for our EPT\* and EPT and a selection of our tested metric indexes for our real-world benchmarks, and in particular, for the results of the speedup

performance. Notice that the diagonal shows the reflexivity of applying the statistical test to the same index's results. Table 7a lists the  $p$  values for the Nasa dataset, where our EPT\* and EPT perform differently between them and with almost all indexes. We found two exceptions, the first one is DiSAT, where EPT\* performs pretty similar to the smallest configuration of EPT\*; the second one is on EPT and KVP that perform almost identical regarding speedup. This behavior is illustrated in Fig. 4a.

Table 7b shows that all indexes perform differently, and thus, Fig. 4b gives a transparent snapshot of the performance of our indexes.

Table 7c shows that our EPT\* and EPT are different indexes under the CoPhIR benchmark. It is interesting to note the  $p$  value between DiSAT and VPT is high, 0.78, and this value implies a high variance in their results that must be taken into account whenever we use these methods on a similar benchmark, i.e., this benchmark is based on  $L_1$  and vectors are small integers. This behavior is hidden if we use only mean values, Fig. 4c, to compare, which also illustrates the necessity of performing statistical hypothesis tests to lead the selection of an index for a specific task.

Table 7d shows  $p$  values for the Gutenberg benchmark. In this benchmark, we found that EPT\* and EPT perform almost identical, and even EPT\* and LC performs similarly. It is also interesting to find that EPT and LC perform pretty similar, both in their configurations with smaller memory usage. As in the case of CoPhIR, this behavior is unnoticeable without a proper statistical test.

## 6 Discussion

It is important to contrast the EPT with the list of clusters (LC), which is a fast and memory-lightweight index. Given a little extra memory, EPT can surpass the search performance of LC, as measured in the number of computed distances. From the speedup point of view, EPT surpasses LC whenever the distance function is costly enough. However, fast distance functions convert LC in a very fast index. On the other hand, the size of the database is a significant problem for LC, especially for high intrinsic dimensionalities which imply impractical construction times, as noted in Table 6.

In contrast to LC, our EPT accepts incremental refining of the index, both of our construction algorithms are based on that; basically, this means that we only need to perform an initialization of the index and then it can solve queries. So, applications that cannot afford large preprocessing times can use a suboptimal EPT and let the optimizing construction run in the background, while queries are solved in a different thread, in this way indexes become faster as time goes.

Based on Figs. 2 and 4, we can observe that search cost and speedup do not optimize their performance on the same index's complexity (captured in figures as index's memory); and it is common to see that small search costs regarding distance evaluations do not correspond with high speedup (measured as clock time improvement to sequential search), and in fact, better speedups occur on EPT indexes with relatively small  $\ell$  values. This behavior implies that we must keep in mind this trade-off for practical applications.

Nevertheless, EP is not a cure-all remedy for metric indexing. On the one hand, the curse of dimensionality is not avoided, it is only deferred, as the dimension increases the EP needs larger  $\alpha$  values, i.e., EP degrades to a sequential search in the limit. On the second hand, even in the reasonable range of dimension, EP is highly dependent on the quality of the estimations of  $\sigma_X^2$ ,  $\sigma_Y^2$  and  $r$ . So, if query distributes pretty different, then the performance

cannot be ensured. However, the assumption that the query distribution can be estimated using the database seems reasonable for a broad range of practical applications.

## 6.1 Comments on approximated algorithms

Any exact metric index will degrade to sequential search as the intrinsic dimensionality of the dataset increases. EP is not an exception; it is just more robust than many other exact indexes. Whenever it is possible to accept some deviations of the result, an approximate algorithm should improve the performance of an exact one. The same argument applies when the speed is preferred over the precision of the response.

There exist a few yet well-known, systematic techniques to transform an exact metric index into an approximate or probabilistic one. In [9], the key idea is to keep untouched the index structure, while the search algorithm takes optimistic decisions in determining if an object is part of the result set. More precisely, the technique, sometimes called aggressive pruning, is based on *stretching* the triangle inequality. Think on  $D(q, u)$  which is a lower bound of  $d(q, u)$  as induced by pivots (Sect. 1). Since  $D(q, u) \leq d(q, u)$ , the set of items retrieved by  $D$  will always contain that retrieved by  $d$ . The question is then, how the addition of a parameter  $0 < \gamma < 1$  in the formulation  $D(q, u) \leq \gamma d(q, u)$  will affect the result set? The answer is a probabilistic one, and in practical terms, a small perturbation ( $\gamma$  close to 1) will improve the search speed significantly, while the quality of the result set remains high. The interested reader is referenced to dedicated studies of the technique [9,39]. Notice that this is a way to convert our EPT in an approximate index, a generic way in fact.

A general method to compute the necessary  $\gamma$  to obtain the desired result quality or search speed is still missing in the area.

## 6.2 Comments on multi-threading implementations

Nowadays, most of the available commodity computers contain a multi-core architecture. Thus, we believe it is important commenting on how to take advantage of the current hardware in our EPT index. The comments focus on both the construction and the searching steps.

Recall that EPT is composed of  $\ell$  pivot groups. Each pivot group is independently constructed (Algorithms 2 and 3); thus at this level, the construction step is a pleasingly parallel problem. It just needs  $\ell$  parallel runs of the construction algorithm. In the case of Algorithm 3, an additional point of pleasingly parallelization is located if the maximization process is transposed, i.e., interchanging for loops of lines 5 and 11. This process is equivalent to searching for the best pivot among *window* alternatives. Notice that the update of  $\sigma_X^2$  should be performed outside of the for loops.

The searching step can also be divided into  $\ell$  searches. Then, gathering the result discarding duplicates or items outside of the desired set (further  $k$  in  $k$ -nn queries). If the distance function is costly, then the search must avoid the evaluation of distances as much as it is possible. In this case, each thread collects candidates; then, in a final step, those items not being discarded by any of the  $\ell$  pivot groups should be reviewed against the distance function. The transposition of *for* loops (lines 3 and 5) also allows good parallelizations; however,  $H$  should be kept isolated for each different thread.

In our experimental section, we used parallel construction to speed up the creation of indexes, but the search algorithm was not parallelized. More complex parallel algorithms are beyond the scope of this manuscript. The same goes for distributed systems.

### 6.3 A note on secondary memory indexes

When the size of the dataset plus the size of the indexes surpasses the available main memory, it is mandatory to move the index to a larger memory level like the disk. Notice that this implies slower accesses than in RAM. As listed in Sect. 1, there exist some metric indexes working on disk. We have maybe listed the most popular ones: M-Tree, PM-Tree, and M-Index. In all these cases, the dynamism and secondary memories strategies (routing nodes, insertion, and deletion) are strongly related to B+-Trees. Also, in all of them, we can observe a data structure based on compact partitions (along with the explicit use of pivots).

Please note that this manuscript is dedicated to set the foundations of EP. Then, we believe essential in focusing on main memory structures due to the difficulty of design indexes in secondary memory. However, we recognize the existence of two major problems needing special attention from the memory perspective and how to tackle them using EPT.

- Large objects. Depending on the size of the object and the dataset, the available main memory can be surpassed. This issue is complex even for disk-based indexes since each object could need random access.
- Very large datasets. No matter the memory efficiency of the index, an index over a very large dataset will need a secondary memory index or the use of a large distributed searching system.

In the former case, it is possible to transform the dataset into a smaller representation. There exist several transformations capturing the proximity of the original space while reducing the representation. This step is standard in many areas since it can clean-up noisy data. Perhaps the most prevalent techniques are PCA, random projections, quantization, and LSI, among many others. The interested reader is referred to the detailed literature on the field [4]. Nevertheless, it is true that these techniques are dependent on the representation and not suitable for the generality of the metric space approach. Also, due to the nature of these methods, its usage implies to lose some information.

In the case of very large datasets, it is impossible to overcome the memory issues arising from their indexing. However, since our EP technique has low memory needs and high-performance searches, it could be part of the filters of secondary memory indexes or even part of the workers of a distributed searching system.

Notice that for moderate datasets (or very large ones segmented in a distributed system), EPT can be used in a hybrid approach as follows.

- The EPT matrix of pairs ( $\text{piv}(u)$ ,  $d(u, \text{piv}(u))$ ) and the actual representation of all pivots are maintained in main memory.
- The rest of the dataset (the objects not being pivots in any group) are stored in secondary memory.
- The search algorithm corresponds to the one described in Algorithm 1; however, in the disk structure, the review's order is fixed to the natural order of the stored dataset (line 5).

A proper study of a secondary memory version of EP beyond the scope of this work and it is left for further research.

## 7 Conclusions

This manuscript presented a novel metric search technique called extreme pivots (EP). EP describes a way to create efficient metric indexes, mainly based on the study of how to relate

objects in a dataset to routing objects (pivots). Recently, most research in the field has been devoted to find the best set of pivots discarding items in a dataset for a given query; several heuristics have been proposed to reach that objective. EP shifts the paradigm by selecting the best items for a given pivot. This simple change of perspective allows for producing small and fast metric indexes. In particular, we fix the implementation to a table-based index called EP Table (EPT), a metric index performing better than related alternatives. The indexing technique incorporates a model which was experimentally proved to be useful if the distance distribution from the objects to the pivots is known, or can be estimated as it is the case of our benchmarks.

We introduced two algorithms to construct EPT indexes; both of them only need to know how much memory should be used in the index structure, and the distribution of the objects and queries. The rest of the parameters are adjusted based on this information.

An EP index is composed of a fixed number of instances (pivot groups). Also, each pivot group uses a small constant number of machine words per database element (two or three, depending on the width of floating point values). We have shown that our implementation based on tables surpasses the performance of both pivot tables and compact partitioning indexes, in a vast number of setups. Wherever EPT does not have the best performance, it achieves attractive trade-offs among memory, search speed, and construction time. Nonetheless, obtaining the best index among different scores is a complex multi-objective problem which is beyond the scope of this manuscript and requires more research. It is worth to notice that this equilibrium among resource requirements makes the EPT a method that can be used in real scenarios.

Some aspects of the EPT that need further development are an approximate version, the use of multiple cores and GPUs to enhance performance, and a dynamic variant where the size of the index grows over time and adjusts without the need of reconstruction.

## References

1. Arya S, Mount D, Netanyahu N, Silverman R, Wu Y (1998) An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *J ACM* 45(6):891–923
2. Böhm C, Berchtold S, Keim DA (2001) Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Comput Surv* 33(3):322–373. <https://doi.org/10.1145/502807.502809>
3. Bolettieri P, Esuli A, Falchi F, Lucchese C, Perego R, Piccioli T, Rabitti F (2009) CoPhIR: a test collection for content-based image retrieval. *CoRR* abs/0905.4627v2. <http://cophir.isti.cnr.it>
4. Burges CJC (2010) Dimension reduction: a guided tour (foundations and trends(r) in machine learning), 1st edn. Now Publishers Inc, Microsoft Research, Boston. <https://doi.org/10.1561/2200000002>
5. Bustos B, Navarro G, Chávez E (2003) Pivot selection techniques for proximity searching in metric spaces. *Pattern Recognit Lett* 24(14):2357–2366
6. Celik C (2002) Priority vantage points structures for similarity queries in metric spaces. In: *EurAsia-ICT '02: proceedings of the 1st EurAsian conference on information and communication technology*. Springer, London, pp 256–263
7. Celik C (2008) Effective use of space for pivot-based metric indexing structures. In: *SISAP '08: proceedings of the 1st international workshop on similarity search and applications (sisap 2008)*. IEEE Computer Society, Washington, pp 113–120. <https://doi.org/10.1109/SISAP.2008.22>
8. Chávez E, Marroquin JL, Baeza-Yates R (1999) Spaghettis: an array based algorithm for similarity queries in metric spaces. In: *String processing and information retrieval symposium, 1999 and international workshop on groupware*, pp 38–46. IEEE
9. Chávez E, Navarro G (2003) Probabilistic proximity search: fighting the curse of dimensionality in metric spaces. *Inf Process Lett* 85:39–46
10. Chávez E, Navarro G (2005) A compact space decomposition for effective metric indexing. *Pattern Recognit Lett* 26:1363–1376. <https://doi.org/10.1016/j.patrec.2004.11.014>

11. Chavez E, Navarro G, Baeza-Yates R, Marroquin JL (2001) Searching in metric spaces. *ACM Comput Surv* 33(3):273–321. <https://doi.org/10.1145/502807.502808>
12. Chen L, Gao Y, Zheng B, Jensen CS, Yang H, Yang K (2017) Pivot-based metric indexing. *Proc VLDB Endow* 10(10):1058–1069. <https://doi.org/10.14778/3115404.3115411>
13. Chávez E, Ludueña V, Reyes N, Roggero P (2016) Faster proximity searching with the distal sat. *Inf Syst* 59:15–47. <https://doi.org/10.1016/j.is.2015.10.014>
14. Ciaccia P, Patella M, Zezula P (1997) M-tree: an efficient access method for similarity search in metric spaces. In: *Proceedings of the 23rd international conference on very large data bases, VLDB '97*. Morgan Kaufmann Publishers Inc., San Francisco, pp 426–435. <http://dl.acm.org/citation.cfm?id=645923.671005>
15. Cormen TH, Leiserson C, Rivest RL, Stein CELC (2001) *Introduction to algorithms*, 2nd edn. McGraw-Hill Inc, New York
16. Hjalton GR, Samet H (2003) Index-driven similarity search in metric spaces. *ACM Trans Database Syst* 28(4):517–580. <https://doi.org/10.1145/958942.958948>
17. Hjalton GR, Samet H (2003) Index-driven similarity search in metric spaces (survey article). *ACM Trans Database Syst (TODS)* 28(4):517–580
18. Jagadish HV, Ooi BC, Tan KL, Yu C, Zhang R (2005) idistance: an adaptive b+–tree based indexing method for nearest neighbor search. *ACM Trans Database Syst* 30(2):364–397. <https://doi.org/10.1145/1071610.1071612>
19. Micó ML, Oncina J, Vidal E (1994) A new version of the nearest-neighbour approximating and eliminating search algorithm (aes) with linear preprocessing time and memory requirements. *Pattern Recognit Lett* 15:9–17. [https://doi.org/10.1016/0167-8655\(94\)90095-7](https://doi.org/10.1016/0167-8655(94)90095-7)
20. Mirylenka K, Giannakopoulos G, Do LM, Palpanas T (2017) On classifier behavior in the presence of mislabeling noise. *Data Min Knowl Discov* 31(3):661–701. <https://doi.org/10.1007/s10618-016-0484-8>
21. Navarro G (2002) Searching in metric spaces by spatial approximation. *Very Large Databases J (VLDBJ)* 11(1):28–46
22. Novak D, Batko M (2009) Metric index: an efficient and scalable solution for similarity search. In: *Second international workshop on similarity search and applications, 2009. SISAP '09*, pp. 65–73. <https://doi.org/10.1109/SISAP.2009.26>
23. Pedreira O, Brisaboa N (2007) Spatial selection of sparse pivots for similarity search in metric spaces. In: van Leeuwen J, Italiano G, van der Hoek W, Meinel C, Sack H, Plášil F (eds) *SOFSEM 2007: theory and practice of computer science. Lecture notes in computer science*, vol 4362. Springer, Berlin, pp 434–445. [https://doi.org/10.1007/978-3-540-69507-3\\_37](https://doi.org/10.1007/978-3-540-69507-3_37)
24. Pestov V (2007) Intrinsic dimension of a dataset: what properties does one expect? In: *Proceedings of 20th International Joint Conference on Neural Networks*, pp 1775–1780
25. Pestov V (2008) An axiomatic approach to intrinsic dimension of a dataset. *Neural Netw* 21(2–3):204–213
26. Pestov V (2010) Indexability, concentration, and VC theory. In: *Proceedings of 3rd international conference on similarity search and applications (SISAP)*, pp 3–12
27. Pestov V (2010) Intrinsic dimensionality. *ACM SIGSPATIAL* 2:8–11. <https://doi.org/10.1145/1862413.1862416>
28. Ruiz G, Santoyo F, Chávez E, Figueroa K, Tellez ES (2013) Extreme pivots for faster metric indexes. In: Brisaboa N, Pedreira O, Zezula P (eds) *Similarity search and applications*. Springer, Berlin, pp 115–126
29. Samet H (2006) *Foundations of multidimensional and metric data structures*. Morgan Kaufmann, Los Altos
30. Shaft U, Ramakrishnan R (2006) Theory of nearest neighbors indexability. *ACM Trans Database Syst* 31:814–838. <https://doi.org/10.1145/1166074.1166077>
31. Skopal T (2004) Pivoting m-tree: a metric access method for efficient similarity search. In: *DATESO'04*, pp 27–37
32. Skopal T (2010) Where are you heading, metric access methods?: a provocative survey. In: *Proceedings of the 3rd international conference on similarity search and applications, SISAP'10*. ACM, New York, pp 13–21. <https://doi.org/10.1145/1862344.1862347>
33. Skopal T, Bustos B (2011) On nonmetric similarity search problems in complex domains. *ACM Comput Surv* 43(4), art. 34
34. Tellez E, Ruiz G, Chavez E (2016) Singleton indexes for nearest neighbor search. *Inf Syst* 60:50–68. <https://doi.org/10.1016/j.is.2016.03.003>
35. Theiler J (1990) Estimating fractal dimension. *J Opt Soc Am A* 7(6):1055–1073. <https://doi.org/10.1364/JOSAA.7.001055>
36. Vidal Ruiz E (1986) An algorithm for finding nearest neighbours in (approximately) constant average time. *Pattern Recognit Lett* 4:145–157



37. Volnyansky I, Pestov V (2009) Curse of dimensionality in pivot based indexes. In: Proceedings of 2nd international workshop on similarity search and applications (SISAP), pp 39–46. <https://doi.org/10.1109/SISAP.2009.9>
38. Yianilos PN (1993) Data structures and algorithms for nearest neighbor search in general metric spaces. In: Proceedings of the 4th annual ACM-SIAM symposium on discrete algorithms, SODA '93. Society for Industrial and Applied Mathematics, Philadelphia, pp 311–321. <http://dl.acm.org/citation.cfm?id=313559.313789>
39. Zezula P, Amato G, Dohnal V, Batko M (2006) Similarity search—the metric space approach. Advances in database systems, vol 32. Springer, Belrin

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



**Guillermo Ruiz** is a Research Fellow of the National Research Council of México at CentroGeo, México. He got a PhD degree in computer science from the Universidad Michoacana in 2015. His research interests are similarity search, machine learning, and computational geometry.



**Edgar Chavez** is a senior researcher at Centro de Investigación Científica y de Educación superior de Ensenada, (CICESE) in Mexico. He obtained a PhD in computer science from Centro de Investigación en Matemáticas (CIMAT) in 1999. His research interests include geometric invariants, metric indexes, classifiers, and multimedia.



**Ubaldo Ruiz** is a Research Fellow of the National Research Council of México at CICESE, México. He obtained a PhD in Computer Science from Centro de Investigación en Matemáticas (CIMAT), in 2013. His research interests include robotics, motion planning, and similarity search.



**Eric S. Tellez** is a Research Fellow of the National Research Council of México at INFOTEC, México. He received a PhD from the Electrical Engineering School at Universidad Michoacana, Mexico. His main research interests are information retrieval, text classification, and similarity search.