On Generalizing Permutation-Based Representations for Approximate Search

Lucia Vadicamo[0000−0001−7182−7038], Claudio Gennaro[0000−0002−3715−149X], and Giuseppe Amato[0000−0003−0171−4315]

Institute of Information Science and Technologies (ISTI), Italian National Research Council (CNR), Via G. Moruzzi 1, 56124 Pisa, Italy
{lucia.vadicamo,claudio.gennaro,giuseppe.amato}@isti.cnr.it

Abstract. In the domain of approximate metric search, the Permutation-based Indexing (PBI) approaches have been proved to be particularly suitable for dealing with large data collections. These methods employ a permutation-based representation of the data, which can be efficiently indexed using data structures such as inverted files. In the literature, the definition of the permutation of a metric object was derived by reordering the distances of the object to a set of pivots. In this paper, we aim at generalizing this definition in order to enlarge the class of permutations that can be used by PBI approaches. As a practical outcome, we defined a new type of permutation that is calculated using distances from pairs of pivots. The proposed technique permits us to produce longer permutations than traditional ones for the same number of object-pivot distance calculations. The advantage is that the use of inverted files built on permutation prefixes leads to greater efficiency in the search phase when longer permutations are used.

Keywords: Permutation-Based Indexing · Metric Space · Metric Search · Similarity Search · Approximate search · Planar projection.

1 Introduction

Searching a database for objects that are most similar to a query object is a fundamental task in many application domains, like multimedia information retrieval, pattern recognition, data mining, and computational biology. In this context, the *Metric Search* framework [24] provides us with a wide class of indexing and searching techniques for similarity data management. A common factor in all these approaches is that they are applicable on generic metric spaces, i.e. these techniques are not specialised for a particular type of data. A *metric space* is a pair (D, d) formed by a domain D and a distance function $d : D \times D \to \mathbb{R}$ that satisfies the metric postulates of non-negativity, identity of indiscernibles, symmetry, and triangle inequality [24]. In a general metric space we cannot use any algebraic function, e.g. sum of two objects or product by scalars, but the only operation that can be exploited is calculating the distance between any two objects. Therefore any technique that aims mapping a metric object $o \in D$

to another (more tractable) space, e.g. a vector space, must rely only on algorithms that use the distances of the object o to other metric objects, e.g. a set of reference objects selected within the space.

Many *approximate metric search* approaches employ transformations of the original metric space to overcome the *curse of dimensionality*, which affects exact metric search techniques whose performance may be not better than a sequential scan for spaces with high intrinsic dimensionality [18,23]. Successful examples of approximate methods are the *Permutation-based Indexing* (PBI) techniques that transform the metric data into permutations of a set of integers $\{1, \ldots, N\}$, which are then indexed and searched using data structures like prefix trees [13,19] and inverted files [3,20]. The original definition of permutation-based representation of a metric object was derived by computing the distances of the object to a set of *pivots* (reference objects) and then by reordering the pivot identifiers according these distances [4,9,10]. This characterization have been adopted in several research papers that further investigated the properties of this data representations and ways to efficiently index them, e.g.[2,13,14,15,17,18,19]. Moreover, some alternative permutation-based representations have been defined in the literature [1,21], but only for representing objects of specific metric spaces.

In this work, we aim at generalizing the definition of permutation associated with a metric object, by introducing the concept of permutation induced by a transformation $f : (D, d) \rightarrow \mathbb{R}^N$. The function f simply projects the metric objects of D into an N-dimensional vector space. This function typically relies only on some distance calculations to transform the objects, such as distances to a set of pivots as done in traditional permutations, but the way distances are combined and exploited to represent objects may be different from what is done in the traditional approach. We believe that this generalization can open up new lines of research, on the one hand, to understand theoretically what properties the function f should have in order to generate permutations that have good performance for the approximate search, and on the other hand, to define alternative permutation-based representations. In this paper, we have started investigating the latter aspect by defining permutations that rely on distances of objects to pairs of pivots. In this way, for a fixed set of n pivots, we can generate permutations with length $N > n$, while the length of traditional permutations is fixed equal to the number of pivots n. The advantage of having longer permutations (at the same cost in terms of original distance computations) is the more efficiency at searching time when using inverted index build upon permutation prefixes (e.g. MI-File [3]). In fact, the inverted index contains as many posting lists as the number N of permutants (i.e. the length of the full permutation) and so, for a fixed permutation prefix length λ, the higher N, the shorter the posting lists, and hence the smaller the fraction of the database accessed to answer a query.

The rest of the paper is structured as follows. Section 2 reviews and generalizes the concept of permutation-based representation of metric objects. Permutations built using distances to pivot-pairs are introduced in Section 3. Section 4 reports the experimental evaluation, and Section 5 draws the conclusions.

2 Permutation-based representation(s) of a metric object

Chavez et al. [9,10] and Amato et al. [4] originally defined the permutation-based representation of a metric object by ordering the identifiers of a fixed set of pivots according to their distances to the object to be represented, that is

Definition 1 (Permutation of a metric object given a set of pivots). *The permutation-based representation Π_o (briefly* permutation*) of an object $o \in D$ with respect to the pivot set $\{p_1, \ldots, p_n\} \subset D$ is the sequence $\Pi_o = [\pi_1, \ldots, \pi_n]$ that lists the pivot identifiers $\{1, \ldots, n\}$ (called* permutants*) in an order such that $\forall\, i \in \{1, \ldots, n-1\}$*

$$d(o, p_{\pi_i}) < d(o, p_{\pi_{i+1}}) \qquad or \qquad \big[d(o, p_{\pi_i}) = d(o, p_{\pi_{i+1}})\big] \wedge [\pi_i < \pi_{i+1}]\,. \quad (1)$$

This representation is also referred to as the *full-length* permutation to distinguish it from the *permutation prefix* adopted in several PBI methods [3,13,19]. In facts, based on the intuition that the most relevant information in the permutation is present in its very first elements, i.e. the identifiers of the closest pivots to an object, several researchers proposed to represent the data by using a fixed-length prefix of the permutation, i.e. $\Pi_{o,\lambda} = [\pi_1, \ldots, \pi_\lambda]$ with $\lambda < n$. The use of permutation prefixes may be dictated by either the employed data structure (e.g. prefix tree), efficiency issues (more compact data encoding and better performance when using inverted files) or even by effectiveness reasons (in some cases the use of prefixes gives better results than full-length permutations [2,3]).

Alternative permutation-based representations have been defined in literature, but only for specific metric spaces. For example, the *Deep Permutations* [1,5] were defined by reordering the dimensions of a vector according to the corresponding element values. This approach can only be used in vector spaces and has so far only been tested on Convolutional Neural Network features. The *SPLX-Perms* [21] use the n-Simplex projection [12] followed by a random rotation to transform a metric object into a Euclidean vector and then computes the permutation by reordering the components of the vector as done in the Deep Permutations. This method can be used on the large class of spaces meeting the n-point property [12] but it is not applicable on general metric spaces.

We now observe that all these approaches belong to the same family of transformations, as explained hereafter, and thus the traditional definition of permutation associated to a metric object (Def. 1) could be generalized to be more inclusive. In this context, the first trivial but useful observation to make is that any sorting function defined on a finite-dimensional Coordinate space implicitly produce a permutation representation of the data. Suppose $\sigma : \mathbb{R}^N \to \mathbb{R}^N$ is a function that sorts the coordinate elements of a N-dimensional real vector with respect to a predefined criterion (e.g. ascending order). For any $\boldsymbol{v} \in \mathbb{R}^N$, the sort function σ is described by the permutation $\Pi^\sigma_{\boldsymbol{v}}$ of the indices $\{1, \ldots, N\}$ that specifies the arrangement of the elements of $\boldsymbol{v}$ into $\boldsymbol{v}' = \sigma(\boldsymbol{v})$. Specifically, if $\boldsymbol{v} = [v_1, \ldots, v_N]$ and $\sigma(\boldsymbol{v}) = [v_{i_1}, \ldots, v_{i_N}]$ then $\Pi^\sigma_{\boldsymbol{v}} = [i_1, \ldots, i_N]$. In other words, the j-th element of the permutation $\Pi^\sigma_{\boldsymbol{v}}$ is the index $i \in \{1, \ldots N\}$ such that the i-th element of $\boldsymbol{v}$ is equal to the j-th element of $\sigma(v)$. For example, if

$\boldsymbol{v} = [8, 10, 6]$ and $\sigma(\boldsymbol{v}) = [6, 8, 10]$ then $\Pi^{\sigma}_{\boldsymbol{v}} = [3, 1, 2]$. However, this characterization is not well defined if the vector $\boldsymbol{v}$ contains duplicate values, therefore we give the following definition.

Definition 2 (Permutation of a vector induced by a sort function σ). *A permutation representation of a vector $\boldsymbol{v} = [v_1, \ldots, v_N] \in \mathbb{R}^N$ associated to a given sort function $\sigma : \mathbb{R}^N \to \mathbb{R}^N$ is the permutation $\Pi^{\sigma}_{\boldsymbol{v}} = [\pi_1, \ldots, \pi_N]$ of the index identifiers $\{1, \ldots, N\}$ such that for any $j = 1, \ldots, N$ the element π_j is the smallest index for which v_{π_j} equals the j-th element of $\sigma(\boldsymbol{v})$.*

The Deep Permutations can be formalized by using the above definition, but this of course cannot be used to describe the SPLX-perms or the traditional permutations. However, these approaches share a common idea, that is using the distance to a set of pivots to first transform the metric object into a Cartesian coordinate space and then obtaining the permutation by applying a sort function. Therefore, for any function $f : (D, d) \to \mathbb{R}^N$ and a given *sort* function we may define a permutation representation of a metric objects as follows:

Definition 3 (Permutation of a metric object induced by a space transformation f and a sort function σ). *Let $f : (D, d) \to \mathbb{R}^N$ a space transformation, and $\sigma : \mathbb{R}^N \to \mathbb{R}^N$ a function that sorts the components of a N-dimensional vector according to some predefined criteria. We define the permutation representation of a object $o \in D$ induced by the functions f and σ as the permutation $\Pi^{\sigma,f}_{o} = [\pi_1, \ldots, \pi_N]$ that lists the index identifiers $\{1, \ldots, N\}$ in an order such that for any $j = 1, \ldots, N$ the permutant π_j is the smallest index for which the π_j-th element of the vector $f(o)$ is equals to the j-th element of $\sigma(f(o))$.*

For the sake of simplicity, in the following we assume that the sort function σ is the sorting of the elements in *ascending* order and we omit the dependency of this function in the definition of the permutation. Please note that the effect of using a different sorting function in most cases could be reproduced by changing the function f. For example, for a given f and object o the permutation obtained by sorting $f(o)$ in descending order is equal to the permutation obtained by applying the function $-f$ to the object o and then sorting the elements in ascending order. Therefore, we use the following characterization:

Definition 4 (Permutation of a metric object induced by a space transformation f). *The permutation representation of a object $o \in (D, d)$ with respect to the transformation $f : (D, d) \to \mathbb{R}^N$ is the sequence $\Pi^{f}_{o} = [\pi_1, \ldots, \pi_N]$ that lists the permutants $\{1, \ldots, N\}$ in an order such that $\forall\, i \in \{1, \ldots, N-1\}$,*

$$f(o)_{\pi_i} < f(o)_{\pi_{i+1}} \qquad or \qquad \left[f(o)_{\pi_i} = f(o)_{\pi_{i+1}}\right] \wedge [\pi_i < \pi_{i+1}] \tag{2}$$

where $f(o)_j$ indicates the j-th value of the vector $f(o)$.

Note that, according to this definition, the traditional permutation is induced by the transformation $f(o) = [d(o, p_1), \ldots, d(o, p_N)]$, where $\{p_1, \ldots, p_N\}$ is a fixed set of pivots. The Deep Permutation is induced by the identity function.

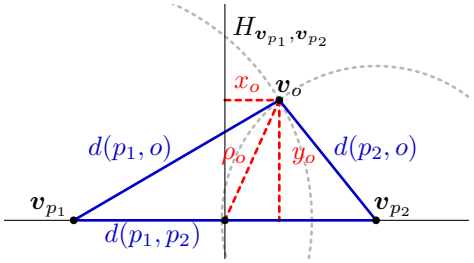


Fig. 1: Planar Projection of two pivots p_1, p_2 and a data point o.

The SPLX-Perm, instead, is induced by the composition of the n-Simplex projection and a random rotation. Moreover, this generalization suggests that new permutation representations of generic metric objects can be defined but assuming that we use a transformation $f : (D, d) \rightarrow \mathbb{R}^N$ that relies only on distance computations and metric postulates to transform the objects. The function f in some cases can also be generated using machine learning techniques, as in [8]. Nevertheless, for particular metric spaces, e.g. vector spaces, other operations or properties of the space can be employed when defining the transformation f. However, not all the transformations may produce permutations which are suitable for metric search, as we would like that similar objects are projected into similar permutations. An in-depth theoretical and experimental study on the properties that the function f should have to produce "good" permutations for approximate metric search is beyond the scope of this paper and we reserve it for future work. Here, as proof of concepts, we define a novel permutation-based representation that uses a transformation f that relies not only on the distance of the objects to a fixed set of pivots but also exploits information on the distances between pivot pairs.

3 Pivot Pairs Permutations

Thanks to the triangle inequality, we know that any three points of a metric space can be isometrically embedded in a two dimensional Euclidean space. Specifically, let $p_1, p_2 \in D$ two pivots and $o \in D$ an arbitrarily metric object. Without loss of generality, we could consider an isometric embedding that maps the points p_1, p_2, o to the vectors $v_{p_1}, v_{p_2}, v_o \in (\mathbb{R}^2, \ell_2)$, such that (i) v_{p_1} and v_{p_2} lies in the X-axis; (ii) v_o is above the X-axis and its coordinate are given by the intersection of the ball centered on p_1 with radius $d(p_1, o)$ and the ball centered on p_2 with radius $d(p_2, o)$. Figure 1 depicts this situation in a 2D coordinate space where the two pivots are projected in the X-axis symmetrically with respect to the origin and a single data object o is mapped to the point $\boldsymbol{v}_o = (x_o, y_o)$, where

$$x_o = \frac{d(o,p_1)^2 - d(o,p_2)^2}{2 \cdot d(p_1,p_2)}, \qquad y_o = \sqrt{d(o,p_1)^2 - \left(x_o + \frac{d(p_1,p_2)}{2}\right)^2} \tag{3}$$

Note that the only information used in the projection is the distances of the object o to the two pivots and the inter-pivot distance. Moreover, all the three distances between the points are preserved, i.e. $\ell_2(\boldsymbol{v}_{p_1}, \boldsymbol{v}_{p_2}) = d(p_1, p_2)$, and $\ell_2(\boldsymbol{v}_{p_i}, \boldsymbol{v}_o) = d(p_i, o)$ for $i = 1, 2$. This projection, called *planar projection* [11], could be repeated for all the data points $o \in D$ while fixing the two pivots p_1, p_2. So we have a projection $\phi_{p_1,p_2} : (D, d) \to (\mathbb{R}^2, \ell_2)$ that preserves the distances of data objects to the two pivots. Therefore, since the distance to the pivots is preserved for each data point, it can be easily proved that all the objects in the hyperplane separating the two pivots in the original space are projected in the hyperplane $H_{\boldsymbol{v}_{p_1},\boldsymbol{v}_{p_2}} = \{\boldsymbol{v} \in \mathbb{R}^2 \,|\, \ell_2(\boldsymbol{v}, \boldsymbol{v}_{p_1}) = \ell_2(\boldsymbol{v}, \boldsymbol{v}_{p_2})\}$ separating the pivots in the 2D projection.

The Euclidean norm of a projected object ($\rho_o = \|\boldsymbol{v}_o\|$) could be interpreted as the distance of the point o to a synthetic pivot that is equidistant to the two original pivots, i.e. a sort of midpoint which may not exist in the original metric space. Its calculation is immediate if we already know $d(p_1, p_2)$, $d(o, p_1)$, and $d(o, p_2)$ as it is equals to

$$\rho_o = \sqrt{(x_o)^2 + (y_o)^2} = \frac{1}{2}\sqrt{2\,d(o, p_1)^2 + 2\,d(o, p_2)^2 - d(p_1, p_2)^2} \tag{4}$$

We can repeat this procedure for several pairs of pivots to characterize a metric object based on the distribution of its distance from the synthetic midpoints between the original pivots. Formally, given a set $\{p_1, \ldots, p_n\} \subset D$ of n pivots, we select $m < \binom{n}{2}$ pivot pairs that we enumerate using an index i, so that (p_{i_1}, p_{i_2}) indicates the i-th pivot pair. For each object $o \in D$ and for each selected pair of pivots (p_{i_1}, p_{i_2}) we use Eq. 4 to compute the norm $\rho_o^{(i)}$ of the projected point $\phi_i(o) = (x_o^{(i)}, y_o^{(i)})$. Then we generate a permutation $\Pi_o^{f'}$ of length m by reordering the components of $f'(o) = \left(\rho_o^{(1)}, \ldots, \rho_o^{(m)}\right)$. Moreover, since we can interpreted the values ρ_i as the distance to some synthetic pivots, we may combine these information with the distances to the actual pivots by computing the permutations induced by the function

$$f'' : o \in D \to \left(d(o, p_1), \ldots, d(o, p_n), \rho_o^{(1)}, \ldots, \rho_o^{(m)}\right) \in \mathbb{R}^{n+m} \tag{5}$$

In the following we refer to the permutations $\Pi_o^{f'}$ and $\Pi_o^{f''}$ as *Pairs Permutation* (`P-Perms`), and *Pivot-Pairs Permutation* (`PP-Perms`), respectively.

4 Experiments

In this section, we compare the performance of `P-Perms`, `PP-Perms`, and the traditional permutations (`Perms`) for approximate k-nearest neighbors (k-NN) search. The experiments were conducted both on real-word and publicly available datasets (CoPhIR and ANN-SIFT) and on synthetic datasets, which are described below. In the following, we first introduce the measures used for the evaluation and then we present the experimental results.

Evaluation Protocol For each dataset we build a ground-truth for the exact k-NN search related to $1,000$ randomly-selected queries. The ground-truths were used to evaluate the quality of the approximate results obtained either by performing a k-NN search in the permutation space or by using the actual distance to re-rank a candidate result set of size $k' \geq k$ that was selected using a k'-NN search in the permutation space. Please note that the latter is a filter-and-refine approach, which requires to store the original dataset and access to it at query time to refine the permutation-based candidate results. In the experiments we used $k = 10$. For the filter-and-refine approach we used $k' = 100$. The quality of the approximate results was evaluated using the *recall@k*, defined as $|\mathcal{R} \cap \mathcal{R}^A|/k$, where $\mathcal{R}$ is the result set of the exact k-NN search in the original metric space and $\mathcal{R}^A$ is the approximate result set.

As done by many PBI approaches [13,3,19], we index and search the data using fixed-length permutation prefixes instead of the full-length permutations. The permutation prefixes were compared using the *Spearman's rho with location parameter* ($S_{\rho,\lambda}$), defined as in [1, Sec. 3.5]), where the location parameter is the length λ of the permutation prefixes. If N is the length of the full permutations (i.e we have N different permutants that may appear in a permutation prefix) and we index the permutation prefixes using inverted files [3], we have that

- the inverted index is composed of N posting lists (one for each permutant).
- each object is stored in exactly λ posting lists (corresponding to the permutants appearing in its permutation prefix). Thus, the i-th posting list contains t_i entries related only to the data objects whose permutations prefixes contain the permutant i.
- each entry of the i-th posting list is of the form $(ID_o, pos_o(i))$, where ID_o is the identifier of a data object, $pos_o(i)$ is the position of the permutant i in the permutation prefix associated to the object o.
- at query time, we access only to the λ posting lists corresponding to the permutants in the query permutation prefix. For each object o in those selected posting lists, we use the stored $pos_o(i)$ to compute the $S_{\rho,\lambda}$ distance to the query permutation prefix.

In this setting, *the size in bits of the inverted index* is a function of the number of permutants N, the prefix length λ, and the number of data objects $|X|$:

$$\text{Size(Inverted Index)} = \underbrace{N \lceil \log_2 N \rceil}_{\text{posting list identifiers}} + \underbrace{\lambda |X| \left(\lceil \log_2 |X| \rceil + \lceil \log_2 \lambda \rceil \right)}_{\text{posting list entries}} \quad (6)$$

The cost at query time includes 1) the cost of transforming the query into the permutation representation; 2) the search cost; 3) the cost of re-ranking the candidate set using the actual distance (only for the filter-and-refine approach). *The cost for computing the permutations* (Table 1) varies with the employed permutation-based representation and the specific metric of the space. For a given set of n pivots the traditional permutation has $N = n$ permutants and requires the calculation of n object-pivot distances. For the same set of pivots and for m selected pivot pairs the `P-Perms` and the `PP-Perms` have $N = m$

Table 1: Distance computations needed for generating various permutation-based representations given the same set of n pivots. m is the number of pairs used in the Pairs and Pivot-Pair Permutations. N is the number of permutants.

Approach	$\boldsymbol{N}$	**Number of Distance Calculations**	
		computed for each object/query	*computed once at indexing time*
`Perms`	n	n actual distances	
`P-Perms`	m	n actual distances +	$\min(m, n(n-1)/2)$ actual distances
`PP-Perms`	$n+m$	m 2D Euclidean distances	

and $N = n + m$ permutants, respectively, and require n object-pivot distance calculation plus m 2D Euclidean distances to calculate the ρ_o (Eq. 4), which in most cases is a negligible cost with respect to object-pivot distance calculations. The `P-Perms` and the `PP-Perms` also require $\min(m, n(n-1)/2)$ pivot-pivot distances, that can be computed and stored once at indexing time and then reused for calculating all the objects/query permutations. The *search cost* (SC), calculated as the *number of bits accessed per query*, is given by

$$SC = \left(\sum_{i=1}^{N} \delta_i t_i \right) C(\text{pEntry}) \tag{7}$$

where t_i is the number of objects stored in the i-th posting list, δ_i is the fraction of samples in the database having the permutant i in their permutation prefixes (i.e. $\delta_i = t_i/|X|$), and $C(\text{pEntry}) = \lceil \log_2 |X| \rceil + \lceil \log_2 l \rceil$ is the size in bits of a single entry of a posting list. In facts, given a query q, we access the i-th posting list only if the index i is in the permutation prefix associated to the query, which is true with probability δ_i as query and database objects share the same distribution. Therefore, the number of elements accessed per query is $\sum_{i=1}^{N} \delta_i t_i$ since the i-th posting list contains t_i entries, and we access it with δ_i probability. Note that for a fixed N, the larger the prefix λ, the greater t_i, thus the higher the search cost. Moreover, for the filter-and-refine approach, the bytes accessed per query are those needed to select the k' candidate results (given by Eq 7) plus those needed to re-rank the candidate results using the actual distance, i.e. $k' * C(Obj)$, where $C(Obj)$ is the size in bits of one original data object.

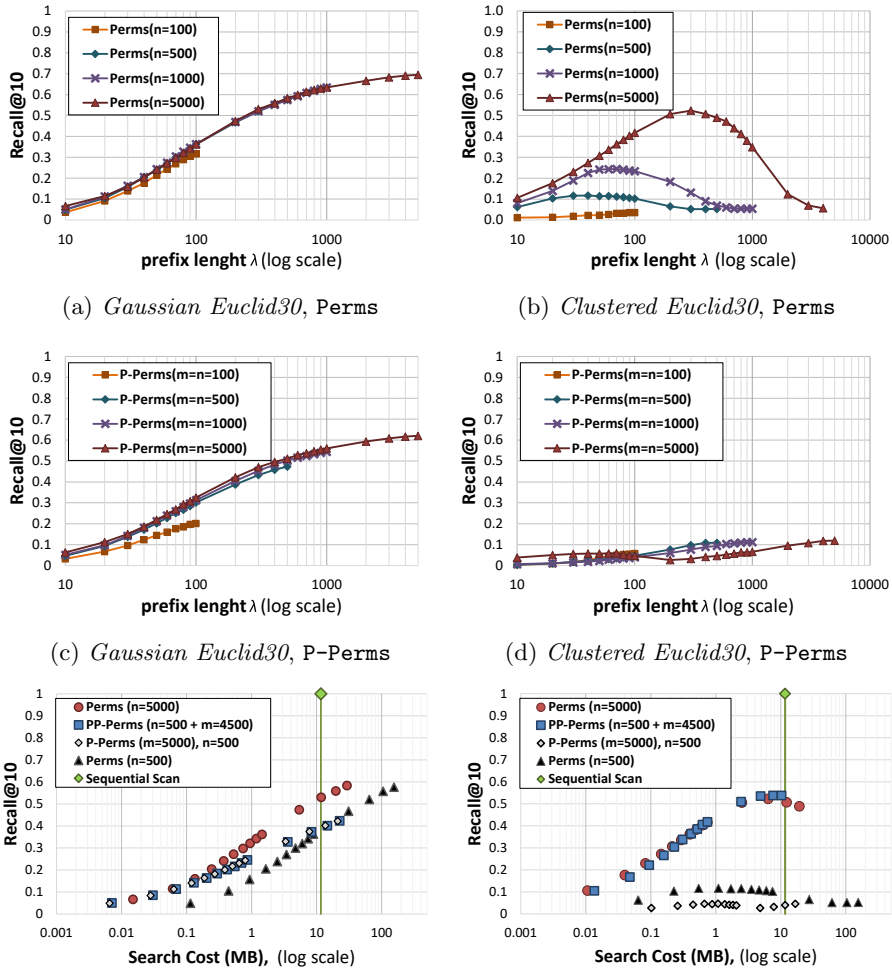
4.1 Experiments on Synthetic Data

The first question that may arise when considering the `P-Perms` representation as an alternative to the traditional permutation (`Perms`) is whether using the distances to the synthetic midpoint pivots instead of the actual pivots still helps in distinguishing similar data points from dissimilar ones in an approximate search scenario. Moreover, since the `P-Perms` and the `PP-Perms` allows producing permutations that are longer than the number of the employed pivots, it would be also interesting to analyze the performance of these permutations when the number m of pairs is increased while fixing the number n of pivots (i.e. fixing the

number of actual distance computation needed to generate the permutations). To this scope, we performed experiments on two representative typologies of synthetic data, clustered and not clustered Euclidean vectors, because it was already proved in the literature [3] that the traditional permutations have different behaviors on these data when varying the number n of pivots and the prefix length λ. Specifically, we considered two datasets, each containing 100K vectors in a 30-dimensional Euclidean space. The first dataset, named *Gaussian Euclid30*, contains vectors whose coordinates are generated using a Gaussian distribution centered in the origin and with a standard deviation $\sigma = 0.1$. The second dataset, named *Clustered Euclid30*, contains vectors arranged in 20 clusters. The cluster centers were randomly selected in the hypercube $[0, 1]^{30}$. For each cluster we generated $5K$ vectors using a Gaussian distribution with a small standard deviation ($\sigma = 0.01$).

Figures 2a and 2b show, for the two datasets, the *recall@10* achieved by the traditional permutation when varying the number n of pivots and the prefix length λ. Please note the different behaviors of the permutations on these two kind of data. On Gaussian Euclid30, the recall increases when increasing both n and λ, but for a fixed λ the recall is almost unchanged when increasing only n (Fig. 2a). For the clustered data, instead, the performance of the *full-length* permutations (i.e. the cases $\lambda = n$) is not improved when increasing the number of pivots more than $n = 500$. However, for a fixed n there exists an optimal prefix length $\lambda < n$ for which the recall achieves a maximum. Amato el al. [3] noted that this maximum is systematically achieved around the prefix length $\lambda = n/cl$, where cl is the number of clusters. Note that n/cl represents the average number of pivots taken from each cluster since we use n random pivots. This suggests that an object of a cluster is well represented by the pivots that belong to its same cluster, but when we increase the length of the permutation prefixes we also include pivots taken from other clusters which seems to introduce noisy information. In facts, when we fix n the recall begins to decrease sharply for $\lambda > n/cl$ (Fig. 2b).

Regarding our initial question, that is, whether `P-Perms` represent a valid alternative to classical permutations in distinguishing objects, for a preliminary analysis we selected a number m of random pivot pairs equal to n, so that the full permutations have the same length (i.e. $N = n = m$). For this settings, we discovered that on *Gaussian Euclid30* the `P-Perms` had similar behaviour and slightly lower effectiveness than the classical `Perms` when varying n and λ (Fig. 2c), thus confirming us that the synthetic pivots computed from the pivot pairs could be used as alternative pivots for generating permutations. However, on the clustered data, the `P-Perms` seems to be completely useless (Fig. 2d) with the exception of the case $\lambda = n$ for which the full-length `P-Perms` slightly outperforms the traditional full-length `Perms` (nevertheless both the approaches achieved very low recall when using their full-length representations). One possible reason for the low performance of the `P-Perms` on clustered data is that we are using as reference objects the synthetic midpoints of just $m = n$ random pivot pairs out



(a) *Gaussian Euclid30*, `Perms` (b) *Clustered Euclid30*, `Perms`

(c) *Gaussian Euclid30*, `P-Perms` (d) *Clustered Euclid30*, `P-Perms`

(e) *Gaussian Euclid30*, `PP-Perms` vs `Perms` (f) *Clustered Euclid30*, `PP-Perms` vs `Perms`

Fig. 2: *Synthetic Datasets*: Recall@10 for the traditional permutations (graphs (a) and (b)) and the `P-Perms` (graphs (c) and (d)) varying the number of pivots and the prefix lengths. Graphs (e) and (f) show the recall for increasing prefix lengths as function of the Search Cost for `Perms` and `PP-Perms`. For each method, the points in the graphs correspond to the prefix lengths $\lambda = 10, 20, 30, 40, 50, 100, 200, 300, 400, 500$

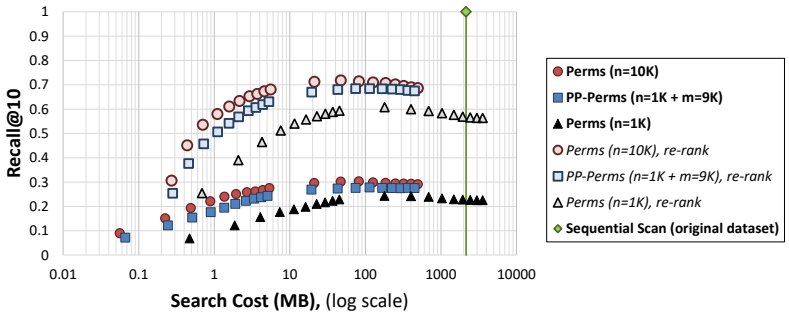
of $n(n-1)/2$ possible pairs. In facts, since the data is uniformly distributed over the cl clusters, the synthetic midpoint of a pair (p_{i_1}, p_{i_2}) is representative of a cluster $\mathcal{C}$ if both p_{i_1} and p_{i_2} belong to the same cluster $\mathcal{C}$, which happen with probability $(n-cl)/cl(n-1)$. Conversely, with probability $n(cl-1)/cl(n-1)$

the two pivots belong to different clusters. For example, if $cl = 20$ and $n = 5K$ the probability of picking a pair of pivots of different clusters is about 95%, so when we use only $m = n = 5K$ random pairs we have on average $4,750$ pairs of pivots form different clusters and just about 12-13 pairs representative of each cluster. To mitigate this inconvenience we may try to use use $m \gg n$ or, as proposed in the following, use the `PP-Perms` representation that employs both traditional and synthetic pivots. The latter approach guarantees to have a percentage of pivots that are still representative of the original data distribution which seems to be fundamental for clustered data. Note that for n pivots and m pairs the `PP-Perms` produces permutations of length $N = m+n$. For some prefix lengths the effectiveness of the `PP-Perms` may decrease when considering $m \gg n$ as the percentage of synthetic pivots will be a way larger than the percentage of actual pivots, which may be a issue for clustered data. Anyway the loss in effectiveness is compensated by the more efficiency at searching time since the Search Cost (number of bits accessed per query) typically increases proportionally to λ^2/N. Therefore, since `PP-Perms` and `Perms` have different lengths N, in the rest of this paper, we report the *recall* values as function of the Search Cost.

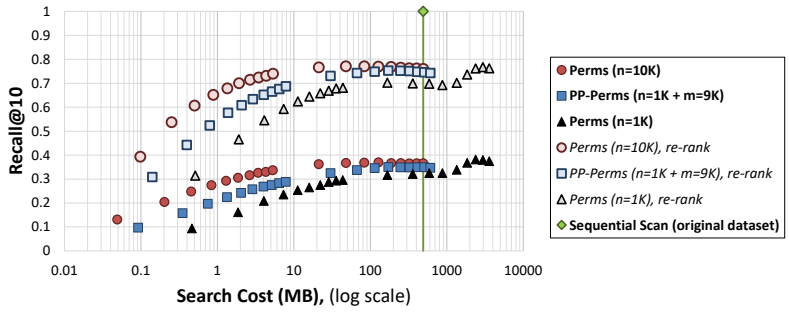
In Figures 2e and 2f we compared the performance of the traditional `Perms` using $n = 500$ pivots ($N = 500$), the `PP-Perms` using $n = 500$ pivots and $m = 4,500$ pairs ($N = 5,000$), the `P-Perms` using $m = 5,000$ pairs selected from $n = 500$ pivots ($N = 5,000$), and the traditional `Perms` using $n = 5,000$ pivots ($N = 5,000$). The latter approach is plotted for reference as it has the same length of the tested `PP-Perms` and `P-Perms`, but note that it requires $5,000$ actual object pivot distance computations, while the other approaches uses a order of magnitude less object-pivot distances computations. For all the approaches we plot the recall versus the search cost when increasing the prefix length λ form 10 to 500. As expected, the `P-Perms`, which rely only on synthetic pivots, has poor performance on the clustered data. However, on the clustered dataset, the `PP-Perms` approach, which uses both actual and synthetic pivots, not only outperforms the `Perms` techniques that use the same set of actual pivots ($n = 500$) but also reaches the performance of the traditional permutations built upon the larger set of pivots ($n = 5,000$). Thus we observed a great advantage in combining synthetic and real pivots to represent clustered data. In facts, the `PP-Perms` shows the best trade-off between recall, search cost and the cost for computing the permutations (i.e. the actual object-pivot distance computations). On Gaussian data, both the `P-Perms` and the `PP-Perms` still outperforms the traditional permutation built on the same pivot set (we are not interested in the recalls when the search costs is greater than the sequential scan). Moreover, for small search cost values, it achieves recalls in line with the more expensive traditional permutation built upon the larger pivot set. Given these outcomes, in the following we focus our attention only on the `PP-Perms` and `Perms` approaches.

4.2 Experiments on Real-World Data

For the experiments on real-world data we used two sets of 1M objects from the *CoPhIR* [7] and *ANN-SIFT* [16] datasets, for which we used different kinds of



(a) *CoPhIR*



(b) *SIFT*

Fig. 3: Recall@10 as function of the Search Cost (with and without re-rank based on the actual distance), for increasing permutation prefix lengths. For each method, the points plotted in the graphs correspond to the prefix lengths $\lambda = 10, 20, 30, 40, 50, 100, 200, 300, 400, 500, 600, 700, 800, 1000$

image features compared with distinct metrics. On the *CoPhIR* data we used as metric the linear combination of the five distance functions (Manhattan, Euclidean, and other special metrics) for the five MPEG-7 descriptors that have been extracted from each image. We adopted the weights proposed in [6, Table 1]. The *ANN-SIFT* contains SIFT local features (128-dimensional vectors) compared with the Euclidean distance. Note that the SIFT data contains some clusters as the distance distribution is a mixture of Gaussians (see [22, Fig. 1]). On both the datasets, we tested the traditional `Perms` using $n = 1,000$ pivots ($N = 1,000$), the `PP-Perms` using $n = 1,000$ pivots and $m = 9,000$ pairs ($N = 10,000$), and the traditional `Perms` using $n = 10,000$ pivots ($N = 10,000$). For each approach we varied the prefix length λ from 10 to 1,000. The results are depicted in Figures 3a and 3b for *CoPhIR* and *SIFT* data, respectively. For reference we also reported the cost of the sequential scan for searching the original data descriptors using the actual distance. Moreover, we also include the results when the actual distance is used to refine (*re-rank*) the candidate results

selected in the permutation space. We observed that on both the datasets the `PP-Perms` performs better than the traditional permutation build upon the same set of pivots. Moreover, for $\lambda > 100$ it achieves recall values in line with that of the more expensive permutation built upon the 10 times larger set of pivots. Therefore, the `PP-Perms` can be profitably used as alternative to the traditional permutation to generate long permutations while limiting the number of actual distance computations. For example, to search 1M SIFT data with a query cost of about 8 MB, the `PP-Perms` achieves a *recall@10* of 0.29 (0.69 when using the re-ranking) while the `Perms` that uses the same set of pivots has a recall of 0.24 (0.56 with the re-ranking). On the CoPhIR data the improvement is even more evident: for a search cost of about 4 MB the `PP-Perms` reaches a recall of 0.24 (0.61 when using the re-ranking) while the traditional permutations has a recall of 0.16 (0.47 when using the re-ranking).

5 Conclusions

In this paper, we generalized the definition of permutations associated to metric objects by introducing the concept of permutations induced by a metric transformation f. As a practical example, we defined permutations induced by a combination of pivots and the tensor product of several planar projections related to some pivot pairs. In our experiments, this novel representation, called `PP-Perms`, achieved the best trade-off between effectiveness (recall) and efficiency (search cost and data distance computations) with respect to the traditional permutations. In facts, for the same set of object-pivot distance calculations, `PP-Perms` allows producing longer permutations, which can be more efficiently searched using inverted files. As future work, on one hand, we would like to investigate theoretical properties that the function f should meet in order to induce effective permutation-based representations; on the other hand, we would like to exploit artificial intelligence techniques to automatically learn suitable functions f.

Acknowledgements

The work was partially supported by H2020 project AI4EU under GA 825619, by H2020 project AI4Media under GA 951911, and by NAUSICAA (CUP D44E20003410009).

References

1. Amato, G., Falchi, F., Gennaro, C., Vadicamo, L.: Deep Permutations: Deep convolutional neural networks and permutation-based indexing. In: Similarity Search and Applications. pp. 93–106. Springer International Publishing (2016)
2. Amato, G., Falchi, F., Rabitti, F., Vadicamo, L.: Some theoretical and experimental observations on permutation spaces and similarity search. In: Similarity Search and Applications. pp. 37–49. Springer International Publishing (2014)
3. Amato, G., Gennaro, C., Savino, P.: MI-File: Using inverted files for scalable approximate similarity search. Multimed. Tools. Appl. (3), 1333–1362 (2014)

4. Amato, G., Savino, P.: Approximate similarity search in metric spaces using inverted files. In: Proceedings International ICST Conference on Scalable Information Systems. pp. 28:1–28:10. ICST / ACM (2008)
5. Amato, G., Carrara, F., Falchi, F., Gennaro, C., Vadicamo, L.: Large-scale instance-level image retrieval. Inf. Process. Manag. **57**(6), 102100 (2020)
6. Batko, M., Falchi, F., Lucchese, C., Novak, D., Perego, R., Rabitti, F., Sedmidubsky, J., Zezula, P.: Building a web-scale image similarity search system. Multimed. Tools. Appl. **47**(3), 599–629 (May 2010)
7. Bolettieri, P., Esuli, A., Falchi, F., Lucchese, C., Perego, R., Piccioli, T., Rabitti, F.: Cophir: a test collection for content-based image retrieval (2009)
8. Carrara, F., Gennaro, C., Falchi, F., Amato, G.: Learning distance estimators from pivoted embeddings of metric objects. In: International Conference on Similarity Search and Applications. pp. 361–368. Springer (2020)
9. Chávez, E., Figueroa, K., Navarro, G.: Effective proximity retrieval by ordering permutations. IEEE Trans. Pattern Anal. Mach. Intell. **30**(9), 1647–1658 (2008)
10. Chávez, E., Figueroa, K., Navarro, G.: Proximity searching in high dimensional spaces with a proximity preserving order. In: MICAI 2005: Advances in Artificial Intelligence. pp. 405–414. Springer (2005)
11. Connor, R., Vadicamo, L., Cardillo, F.A., Rabitti, F.: Supermetric search. Inf. Syst. **80**, 108–123 (2019)
12. Connor, R., Vadicamo, L., Rabitti, F.: High-dimensional simplexes for supermetric search. In: Similarity Search and Applications. pp. 96–109. Springer International Publishing (2017)
13. Esuli, A.: Use of permutation prefixes for efficient and scalable approximate similarity search. Inf. Process. Manage. **48**(5), 889–902 (2012)
14. Figueroa, K., Chavez, E., Navarro, G., Paredes, R.: Speeding up spatial approximation search in metric spaces. ACM J. Exp. Algorithmics **14** (Jan 2010)
15. Hetland, M.L., Skopal, T., Lokoč, J., Beecks, C.: Ptolemaic access methods: Challenging the reign of the metric space model. Inf. Syst. **38**(7), 989–1006 (2013)
16. Jegou, H., Douze, M., Schmid, C.: Product quantization for nearest neighbor search. IEEE Trans. Pattern Anal. Mach. Intell. **33**(1), 117–128 (2010)
17. Kruliš, M., Osipyan, H., Marchand-Maillet, S.: Employing gpu architectures for permutation-based indexing. Multimed. Tools. Appl. **76**(9), 11859–11887 (2017)
18. Naidan, B., Boytsov, L., Nyberg, E.: Permutation search methods are efficient, yet faster search is possible. Proceedings International Conference on Very Large Data Bases **8**(12), 1618–1629 (2015)
19. Novak, D., Zezula, P.: PPP-Codes for Large-Scale Similarity Searching, vol. 24, pp. 61–87. Springer Berlin Heidelberg (2016)
20. Tellez, E.S., Chávez, E., Navarro, G.: Succinct nearest neighbor search. Inf. Syst. **38**(7), 1019–1030 (2013)
21. Vadicamo, L., Connor, R., Falchi, F., Gennaro, C., Rabitti, F.: SPLX-Perm: A novel permutation-based representation for approximate metric search. In: Similarity Search and Applications. pp. 40–48. Springer International Publishing (2019)
22. Vadicamo, L., Mic, V., Falchi, F., Zezula, P.: Metric embedding into the hamming space with the n-simplex projection. In: Similarity Search and Applications. pp. 265–272. Springer International Publishing (2019)
23. Weber, R., Schek, H.J., Blott, S.: A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In: Proceedings International Conference on Very Large Data Bases. vol. 98, pp. 194–205 (1998)
24. Zezula, P., Amato, G., Dohnal, V., Batko, M.: Similarity search: the metric space approach, vol. 32. Springer Science & Business Media (2006)