A Brief Index for Proximity Searching

Eric Sadit Téllez[1], Edgar Chávez[1], and Antonio Camarena-Ibarrola[1]

Universidad Michoacana
Mexico

Abstract. A large number of computer tasks can be modeled as the search for an object near a given query. From multimedia databases to learning algorithms, data mining and pattern recognition, the metric space model of proximity and retrieval can be used as a searching paradigm.
For metric space indexing the permutation based approach consist in saving the order in which a set of reference objects (the permutants) is *seen* by every element of the database. Adding up the relative displacements with respect to the query is an excellent predictor of proximity, and is called the Spearman ρ distance.
In this paper we show how to represent the permutation as a binary vector, using just one bit for each permutant (instead of $\log k$ in the plain representation). Hamming distance can be used then to predict proximity. We tested this approach with many real life metric databases obtaining a recall close to the Spearman ρ (or even better in some examples), and speedup from 2 to 4 times faster.

1 Introduction

A metric space is composed by an universe of objects $\mathbb{U}$, and a distance function $d : \mathbb{U} \times \mathbb{U} \to \mathbb{R}$, following for any objects $x, y, z \in \mathbb{U}$:

1. Strict positiveness: $d(x, y) > 0$, excepting for the identity of indiscernibles, or the reflexivity property: $d(x, y) = 0 \iff x = y$.
2. Symmetry: $d(x, y) = d(y, x)$.
3. The triangle inequality: $d(x, z) + d(z, y) \geq d(x, y)$, which gives the searching facilities.

It's common to found distance function which are really expensive to compute (i.e. comparing fingerprints, searching by content in multimedia, etc), yielding to the necessity of a reduction in the number of distance computations during the the search process. Indexing metric spaces is devoted to create algorithms and structures for efficiently solve queries in metric spaces. There're two main search kinds, for a given database $S \subseteq \mathbb{U}$ which is finite with size $|S| = n$, the problem is to perform searches efficiently of the form $(q \in \mathbb{U}, r \in \mathbb{R})_d = \{x \in S \mid d(q, x) \leq r\}$, called *range queries*. The focus of the brief index is the k *Nearest Neighbors*, $kNN_d(q)$ searches for short. The kNN retrieves the k closest elements to q in

S, formally it retrieves the set $R \in S$ such that $|R| = k$ and $\forall u \in R, v \in \mathbb{U} - S$ it follows $d(q, u) \leq d(q, v)$.

In general, metric indexes uses the metric properties of d to allow fast searches over S, There exists many indexes, ranging from general solutions to specialized ones. We will cover a really small subset of them, used in the literature as canonical examples and comparison points.

Burkhard and Keller introduces the BKTree (BKT) [1], uses the triangle inequality to prune elements. The BKT is designed for discrete distances (continuos distances can be discretized in order to be useful). The BKT is an L-ary tree created recursively, as primitive build operation we choose some $p \in S$ and then: 1) If the BKT is empty, p is the new root's tree. 2) If the root node exists, then $l = d(p, root)$ and p will try to repeat step 1 with the l-th subtree of the current root. A general *bucket* b, can be applied allowing b elements in the $leafs$ instead only one element. Searching for $(q, r)_d$ can be searched using the hierarchical structure discarding children not matching $i - r \leq d(q, p_i) \leq i + r$. The kNN searches can be constructed systematically from range searches using branch and bound [2, 3], starting with $r = \infty$ and closing r under evidence of smaller radius. This procedure is optimal for BKT [2].

The Approximating and Eliminating Searching Algorithm (AESA) [3] is a simple and efficient algorithm for similarity searching, it reaches the extreme of pivot's based algorithms (in fastness and space usage). The AESA algorithm achieves $O(1)$ searching time, at cost of $O(n^2)$ space and construction time. It maintains a matrix of $n(n-1)/2$ cells, storing all the distances between element pairs in S. In the searching procedure for $(q, r)_d$, a random $p \in S$ is chosen, the distance $d(q, p)$ is calculated and all the elements $s \in S$ not matching $d(p, q) - r \leq d(p, s) \leq d(p, q) + r$ are discarded from the result set. The search process finish when all the surviving elements are inside the desired range. The $O(n^2)$ space cost is a really big restriction moderate n. As in the BKT, the kNN search can be built over range searches and branch and bound.

For a deeper and extended study over indexing and searching in metric spaces, please reefer to [2–4].

===

Permutantes

===

From a wider point of view, exists important spaces where the metric properties are not achieved, but the similarity search problem is still present. For example, when the triangle inequality is not kept (i.e. formally known as pseudo-metric spaces) the search capabilities can't be fully exploited by techniques based on metric spaces. The permutation index allows kNN searches in pseudo-metric spaces, because it's independent of the triangle inequality. Our technique effectively inherits this property allowing faster searches and smaller indexes in metric and pseudo-metric spaces, with an small (even nothing) recall's footprint against the permutation index.

The paper's organization is as follows: The Section 1 is devoted to an introductory overview to the problem and the previous work. The Section 2 explains

in detail the construction of the brief permutations, and the achieved characteristics. In order to show the efficiency and effectivity of the method, we provide experimental evidence over four well known spaces (documents, string's dictionary, image's color histograms, and audio fingerprints), the Section 3 is dedicated to this purpose. Finally, the Section 4 summarizes the remarkable points of the paper, our contributions, and a list of opened questions that will direct our future work.

2 The Brief Permutations

Algorithm 1 Bit-encoding of the permutation P under the module m

Encode(Permutation P, Positive Integer m)

Let P^{-1} be the inverse P.
$C \leftarrow 0^{|P|}$ {Bit string of size $|P|$, initialized to zeros}
for all i from 0 to $|P| - 1$ **do**
    $p \leftarrow P[i]$
    if $|i - p| > m$ **then**
        $C \leftarrow C|(1 << i)$
    end if
end for
return C

Algorithm 2 Bit-encoding of the permutation P under the module m using permutation of the center

EncodePermCenter(Permutation P, Positive Integer m)

Let P^{-1} be the inverse P
$C \leftarrow 0^{|P|}$ {Bit string of size $|P|$, initialized to zeros}
$M = \frac{|P|}{4}$
for all i from 0 to $|P| - 1$ **do**
    $p \leftarrow P[i]$
    $I \leftarrow i$
    if $\lfloor \frac{I}{M} \rfloor \neq 0$ **then**
        $I \leftarrow I + M$
    end if
    if $|I - p| > m$ **then**
        $C \leftarrow C|(1 << i)$
    end if
end for
return C

Algorithm 3 Construction of the Index

Construction(Permutants Ψ, Database S, Distance d, Positive Integer m)

1: $D = [\,]$ {A list to store the bit-encodings}
2: $C \leftarrow 0^{|P|}$ {Bit string of size $|P|$, initialized to zeros}
3: **for all** $s \in S$ **do**
4: $P \leftarrow$ Compute the permutation of s under Ψ and d.
5: Let P^{-1} be the inverse of P.
6: $D \leftarrow D + [Encode(P^{-1}, m)]$
7: **end for**
8: I $\leftarrow$ Construct a metric space index for Hamming over D
9: **return** I

Algorithm 4 Procedure to search kNN for q

SearchKNN(Index I, Permutants Ψ, Distance d, Object q, Positive Integer m, Positive Integer k, Positive Integer $Cand$)

1: $P \leftarrow$ Get permutation for q under Ψ and d.
2: Let P^{-1} be the inverse P.
3: $h \leftarrow Encode(P, m)$.
4: $R \leftarrow$ Retrieve the kNN for h with metric index I using $k = Cand$ {Remember that $R \subseteq S$}.
5: $Res \leftarrow [\,]$
6: **for all** $s \in R$ **do**
7: $Res \leftarrow Res + [(d(s, q), s)]$
8: **end for**
9: $Res \leftarrow$ sort Res by the first argument in the tuple, keep the smallest k results.
10: **return** Res

The encoding is made in two phases: the permutation transformation, and the permutation encoding. The first phase is omitted because the lack of space, and the definition given in previous sections should be sufficient. We use C programming language like syntax for bitwise operations (i.e. '&' for *and*, '|' for *or*, '<<' for *left shift*, '>>' to *right shift*, and '∧' for *xor*). We start in zero the indexing of bit sequences and permutations.

The Algorithm 1 shows the algorithm for condensing the permutation information into bit strings. In the Algorithm 1 we can notice that for bigger m (e.g $m \leq |P|$) the permutants in the center of the inverse are punished without reason. In order to reduce this effect, we should permute the center of the inverse, for example let $\Pi = 1234$ then we should transform using the permutation $\Gamma = 2143$ when encoding 2 and 3. This can be easily and efficiently implemented for any number of permutants with a few modifications to the original procedure, see the Algorithm 2. As we show experimentally in the next Section, this variant achieves better recalls than the naive encoding. In the following algorithms any reference to Encode is valid for the EncodePermCent procedure.

In our implementation the index construction is fairly simple, since we only apply the bit-encoding transformation without really indexing, but any metric index could be used to index the resulting hamming space. The basic algorithm is show in Algorithm **??**. Search procedure is really simple (see Algorithm 4), and consists in searching for candidates in the hamming space, and a final reordering of the result using the original distance.

3 Experiments

The tested databases were taken from the *metric space* library[1] and the natix project's site[2]. Our implementation is available as open source from `www.natix.org`, the indexes were written in Python[3], and the distance functions were implemented in C++, and jointed using SWIG[4]. All indexes are sharing the same distance function's implementation. The experiments were performed in a laptop computer with Intel Core 2 at 2.4 GHz and 2GiB of RAM, running MacOS X 10.5.6. The presented indexes run in primary memory without using the processor's parallel processing capabilities.

The chosen databases are representatives of four typical : documents and dictionaries for textual information retrieval, color's histogram vectors for multimedia information retrieval, and song fingerprints for music information retrieval.

Trying to kept a clear and simple document, we only provide a deep comparison against the full permutation index. An exhaustive comparison between the permutation's index and other classical similarity search method is presented in [5]. In order to maintain a well known reference, we compare against the performance of the BKT the selected searches, but excluding it from the figures.

[1] Metric space library `www.sisap.org`.
[2] Natix web site is `www.natix.org`.
[3] The python programming language `www.python.org`.
[4] The swig's website is `www.swig.org`.

The recalls are measured against the exact result given from sequential searching and BKT. For the audio fingerprints, we use a ground truth for the performed queries.

Number of permutants	Permutants	Brief permutants
128	29	1.8
256	57	3.5
512	112	6.9
1024	222	14
2048	443	28

Table 1. Practical index's size for color's histogram collection. In the practice, each full permutant uses 16 bit, and 1 bit per brief permutant.

As an annotation for all the experiments, the Table 1 exemplifies the index's size reduction, which allows to kept in primary memory indexes for databases 16 times bigger than the permutation index (i.e. using the same number of permutants), and improve the search's speed (hamming can exploit the inherent bit parallelism and use lookup tables). Actually, the Hamming space is really simple and can be easily implemented in dedicated minimal processors, allowing a new gamma of applications in small specialized devices.

3.1 Documents

A collection of 25157 short news articles in the $TF \times IDF$ format from Wall Street Journal $1987 - 1989$ files from TREC-3 collection. We use the angle between vectors as distance measure [6].

We extract 100 random documents from the collection for queries, avoiding the indexation of queries. Each query searches for 30 nearest neighbors (the BKT needs to check up to 98% of the database to get the entire result set). The recall for the nearest neighbor is shown by the Figure 1(a), and Figure 1(b) shows the recall for 30NN. Is necessary to notice that the number of distance's evaluations is the number of permutants plus the number of candidates, then for a NN recall of 0.93 and $30NN$ recall of 0.82 we need to review only the 6% of the database. The recall for the brief permutations is tightly related with the module, with best results for small modules with special attention in small number of permutants for 0.3, whilst the number of permutants increases the module effect decreases. We can see that module 0.5 is a good choice for any number of permutants avoiding the necessity of optimizing the parameter for most of the document retrieval applications. We can see in both recall Figures that the brief index performs slightly better than the full permutants.

The Figure 1(c) shows the average real time per search needed for each number of permutants.

3.2 Vectors

A set of 112544 color histograms (112-dimensional vectors) from an image database[5]. We use the metric space library's version which doesn't kept the duplicated vectors.

For this experiment, we choose randomly 200 histogram vectors and we apply a *perturbation* of ± 0.5 on one random coordinate. The search consist on finding 30NN under L2 distance. The BKT needs to review 65% of the database. This is a hard space for the permutation transformation, achieving only a recall of 0.7 for 2000 checked candidates (equivalent to review a 2% of the database, but unfortunately the recall growth is slow from this point, even augmenting the number of permutants), the exact reasons of this behavior is unknown, and this experiment shows that the behavior is inherited by the brief index.

Even with this *bad* behavior, it's an excelent approximation for achieving fast searches for massive Multimedia Information Retrieval approaches [7].

In the Figure 2(a) the recall for different permutation's number is presented, even when the module controls the recall, a good option is to choose module 0.5, for any number of permutations. The behavior is similar to the 30NN, as can be seen in Figure 2(b).

The average time per search can be seen in Figure 2(c), we can see that always the brief index is faster.

3.3 Dictionary

A common task in textual information retrieval is searching in dictionaries, correcting misspelled words, OCR, etc. We use the metric space library's English dictionary with 69069 words. We choose the english dictionary to avoid encoding particularities of other languages, but we expect the same behavior from other non-agglutinant languages. We use the edit distance.

We randomly chooses 200 words as queries from the database. We perform searches for 30NN. For the first NN (i.e the word as query) we have recall of 1.0 for more than 128 permutants, then a better effectivity measure is the recall for 2NN (Figure 3(a)) and the full 30NN (Figure 3(b)). It's necessary to notice that there exists several possibilities at radius 1, but we show the recall for the second NN given in the exact result. This gives a lower recall but gives a general idea of what's happening in the result set in addition to the 30NN recall.

In order to get the exact result for 30NN, the BKT needs to review 56% of the database. The brief index needs to review 3% to get a recall of 0.97 for 2NN and 0.84 for 30NN using our standard module 0.5. We get 1.0 recalls reviewing 6% of the database. The average search time is shown in Figure 4(b).

3.4 Audio Fingerprints

A database of 10254 MBSES [8] fingerprints using three byte's frame for each 46 ms. The fingerprints were extracted from full songs of different genres and

[5] The original database source is `http://www.dbs.informatik.uni-muenchen.de/~seidl/DATA/histo112.112682.gz`

authors was used[6]. We use a non-metric distance called *probabilistic pairing psudo metric* [9] which is defined as the minimum hamming distance from one short sequence of length m against all m-grams inside a larger sequence. The distance's cost is $O(m \times (n - m + 1))$

We use permutations of song excerpts of $20s$, and queries of $20s$.

The Figure 4(a) shows the recall given for brief index and full permutations. We can see a recall of 0.83 for 512 permutants. The full permutation index has a recall of 0.92, this means a ratio of 0.9 between brief permutations and full permutations.

In Figure **??** we show the time taken per search. We must notice than BKT can be used loosing some results because the distance function doesn't follows the triangle inequality, but we can get recalls bigger than 0.9 but we need to review more than 40% of the database, resulting in 30 seconds per search. The brief index needs to review 512 distance's evaluations, and 1000 distances verification (i.e. review 15% of the database). The verification is done using the transitivity kept by the distance, using only 12 or 24 frames, reducing the final cost of the query. This schema needs a non-metric index, and full permutations and brief permutations can be used to reduce search costs.

4 Conclusions

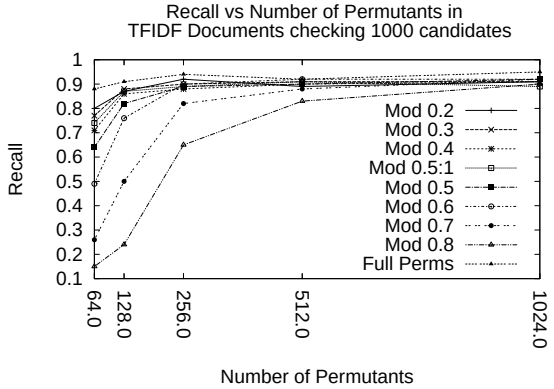
- Necesitamos acelerar la busqueda secuencial para pesar las permutaciones
- Usar el indice metrico, reducir a vectores con solo los permutantes "escenciales" para generalizar el esquema de indice invertido metrico,probar con una db gigante. Usar distancia de coseno para este punto, ademas del spearman rho.
- Optimizar hamming para secuencias largas
- Hemos visto que para modulo 0.5 se comporta bien en cualquier permutacion, pero no es optimo, sacar una relacion exacta que diga que modulo debe usarse.
- Enfatizar que el espacio es reducido en 16 veces (short int a bit) para implementaciones practicas.
- Poner en tablas los tamanios de los indices

References

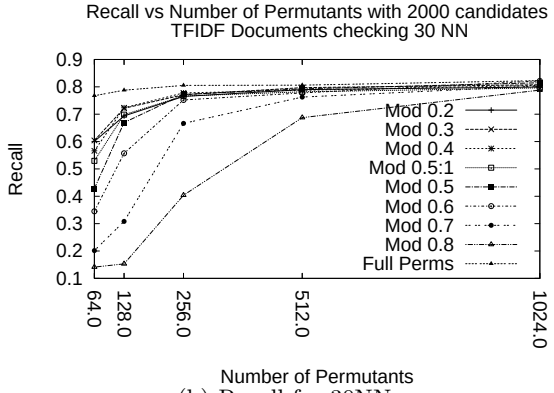
1. Burkhard, W.A., Keller, R.M.: Some approaches to best-match file searching. Communications of the ACM **16**(4) (1973) 230–237
2. Samet, H.: Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann Publishers (2006)
3. Chávez, E., Navarro, G., Baeza-Yates, R., Marroquín, J.L.: Searching in metric spaces. ACM Comput. Surv. **33**(3) (2001) 273–321

[6] This database is available from the `www.natix.org` website.

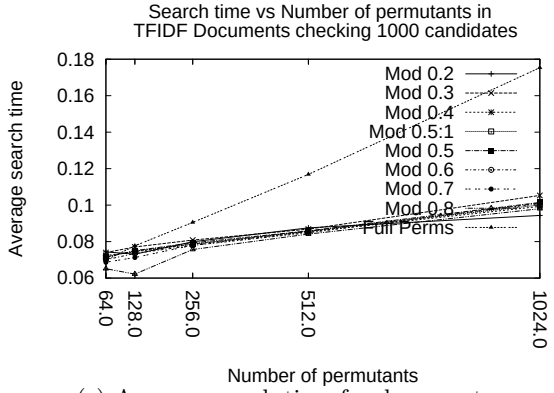
4. Böhm, C., Berchtold, S., Keim, D.A.: Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. ACM Computing Surveys **33**(3) (2001) 322–373
5. Chavez, E., Figueroa, K., Navarro, G.: Effective proximity retrieval by ordering permutations. IEEE Transactions on Pattern Analysis and Machine Intelligence **30**(9) (September 2008) 1647–1658
6. Baeza-Yates, R.A., Ribeiro-Neto, B.A.: Modern Information Retrieval. ACM Press / Addison-Wesley (1999)
7. Amato, G., Savino, P.: Approximate similarity search in metric spaces using inverted files. In: InfoScale '08: Proceedings of the 3rd international conference on Scalable information systems, ICST, Brussels, Belgium, Belgium, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering) (2008) 1–10
8. Ibarrola, A.C., Chávez, E.: A robust entropy-based audio-fingerprint, IEEE (July 2006)
9. Chavez, E., Camarena-Ibarrola, A., Téllez, E.S., Bainbridge, D.: A permutations based index for fast and robust music identification. Technical Report. Universidad Michoacana (2009)



(a) Recall for NN

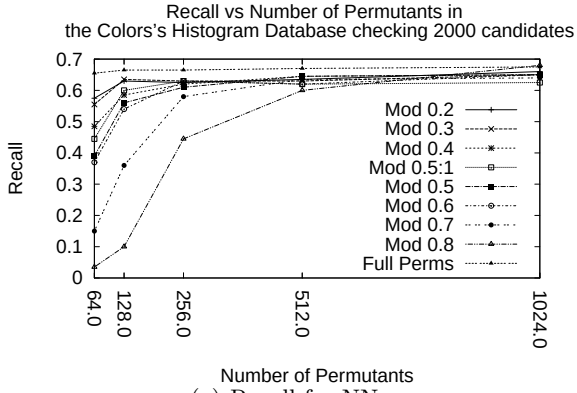


(b) Recall for 30NN

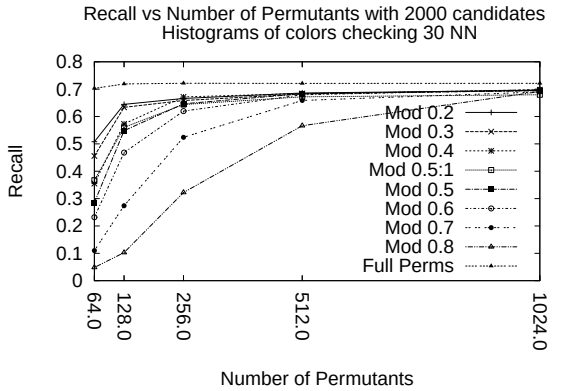


(c) Average search time for documents

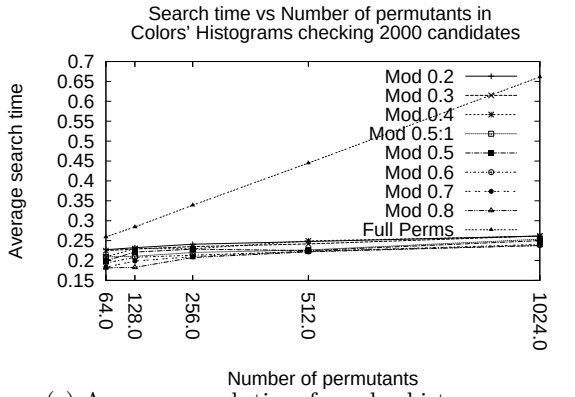
Fig. 1. Experiment results for brief index against full permutations using the documents $TF \times IDF$ collection and cosine's angle as distance.



(a) Recall for NN

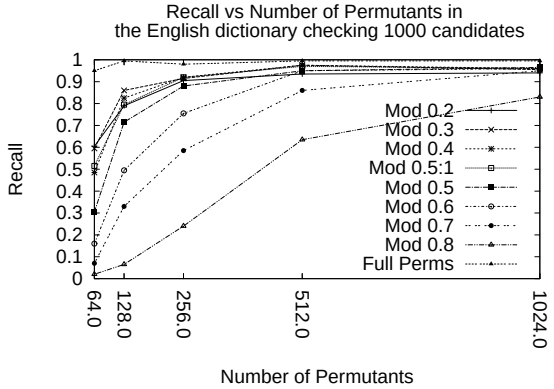


(b) Recall for 30NN

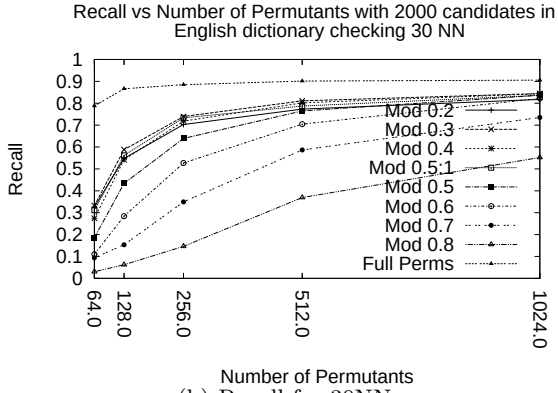


(c) Average search time for color histograms

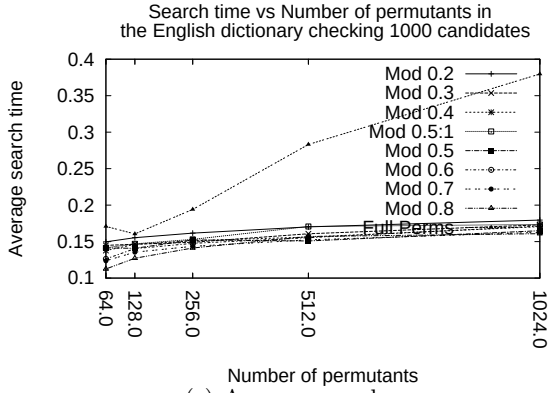
Fig. 2. Results for brief index against full permutations using the color's histogram collection and L2 distance.



(a) Recall for 2NN

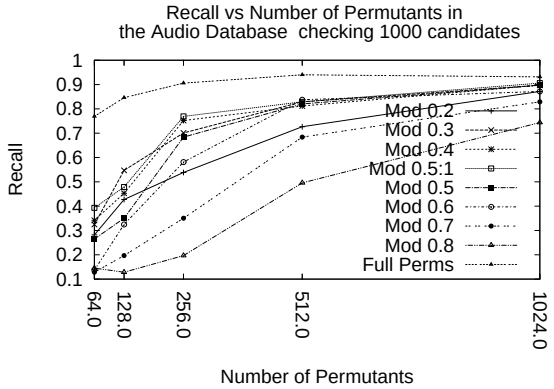


(b) Recall for 30NN

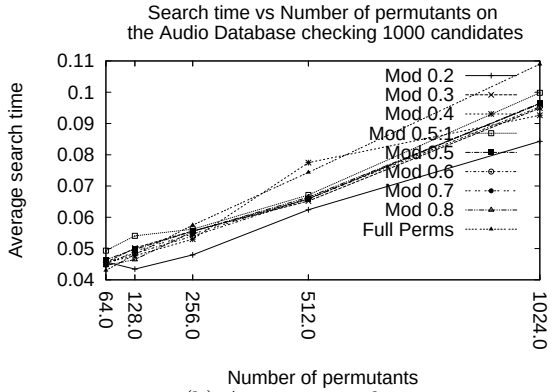


(c) Average search

Fig. 3. Results for brief index against full permutations for and english dictionary, using edit distance.



(a) Recall for matching the NN against the ground truth



(b) Average search

Fig. 4. Results for brief index against full permutations for the audio collection.