On Locality Sensitive Hashing in Metric Spaces

Eric Sadit Téllez and Edgar Chávez
sadit@lsc.fie.umich.mx, elchavez@fismat.umich.mx

Universidad Michoacana / CICESE
México

Sep 18, 2010

Table of contents

Introduction / Proximity Search

- We investigate a method to extend Locality Sensitive Hashing (LSH), from vector spaces to the more general model of similarity spaces, in particular metric spaces.
- The use of LSH is limited since a hashing function should be defined for each data type.
- Here we provides a simple and effective solution to define a hashing function using the distance between objects as a black box.
- We will focus on the K nearest neighbors problem.

Challenges in *exact* proximity search

- Locality
- Secondary memory
- The curse of dimensionality

The promise of approximate proximity searching

Fast retrieval by allowing false dismissals

- In vector spaces: LSH has been tested in many database examples
- In similarity spaces: Various flavors of Permutation based indexing have proven to be effective in various real world settings

Locality Sensitive Hashing

Definition

A family $\mathcal{H}$ of functions $h : R^d \rightarrow U$ is called $(P1, P2, r, cr)$-sensitive, if for any p, q:

- if $||p - q|| < r$ then $Pr[h(p) = h(q)] > P1$
- if $||p - q|| > cr$ then $Pr[h(p) = h(q)] < P2$

The framework is general enough to cover similarity spaces, but concrete examples of LSH are tied to the data structure (e.g. Hamming, ℓ_p metrics, cosine distance, etc.) $\mathcal{H}$ is defined casuistically.

Permutation based indexes

Definition

Let $\mathbb{X}$ the universe of objects, $\mathbb{S} \subseteq \mathbb{X}$ the database, and $\mathbb{P} = \{p_1, p_2, \ldots, p_k\} \subseteq \mathbb{S}, u_i \in \mathbb{S}$ and $q \in \mathbb{X}$,.

Permutation indexes predict the relative proximity between objects by measuring the perspective to the set of references $\mathbb{P}$. The hypothesis is: If u *see* the set of permutants in a similar way to q they are likely to be close.

Permutation based indexes (cont)

Definition

Let $\mathbb{P} = \{p_1, p_2, \ldots, p_k\}$ and $x \in \mathbb{X}$. Then we define Π_x as a permutation of $(1 \ldots k)$ so that, for all $1 \leq i < k$ it holds either $d(p_{\Pi_x(i)}, x) < d(p_{\Pi_x(i+1)}, x)$, or $d(p_{\Pi_x(i)}, x) = d(p_{\Pi_x(i+1)}, x)$ and $\Pi_x(i) < \Pi_x(i+1)$.

Definition

Given permutations Π_u and Π_q of $(1 \ldots k)$, Spearman Footrule is defined as

$$S_\rho(\Pi_u, \Pi_q) = \sum_{1 \leq i \leq k} \left| \Pi_u^{-1}(i) - \Pi_q^{-1}(i) \right|.$$

Brief Permutation Index

The idea is to represent each permutation with only one bit.

1: Let Π_u^{-1} be the inverse Π_u.
2: $C \leftarrow 0$ {Bit string of size $|\Pi_u|$, initialized to zeros}
3: **for all** i **from** 0 **to** $|\Pi_u| - 1$ **do**
4: **if** $|i - \Pi_u^{-1}[i]| > m$ **then**
5: $C[i] \leftarrow 1$
6: **end if**
7: **end for**
8: **return** C

Brief Index (cont)

To compare brief permutations, the distance is defined as:

- $0 \oplus 0 = 0$ small relative displacement
- $0 \oplus 1 = 1$ large relative displacement
- $1 \oplus 1 = 0$ tricky one. We choose 0 to coincide with the Hamming distance

Properties:

- *Compact:* It needs a single bit per permutant
- *Fast:* Hamming distance can be computed in chunks of 32 or 64 bits at once
- We pay a small fee with the recall

The center of the brief permutation

- Center bits of the brief permutation are zero almost all the time
- We can pack two permutations using the center

2 Locality Sensitive Hashing for Metric Spaces

- Faster, compact and cute
- Building the index
- Searching

Making the index

- Create a brief index of the database
- Use Hamming LSH on top of the brief index

REMARK: LSH can be used for full permutations and brief permutations. Here we present only LSH on the brief permutations because it is faster and more studied.

Building the index (cont)

We need to tune:

- Number of permutants (references) (numrefs) to be used by brief permutations.
- Number of hashing functions (numhashes) in the LSH layer. More hashes imply more candidates, a bigger recall and higher cpu cost.
- Size of the bit sample (samplesize) for LSH hashes. Larger values produces sparser tables, resulting in faster searches but with few candidates.

Searching

The procedure for solving KNN queries for q is quite simple:

- Compute the brief permutation of q, π_q.
- Compute the hashes of π_q, Q.
- Retrieve associated candidates for Q.
- Optionally: Resort candidates for importance in the brief index space. This is useful for expensive distances. In our experiments we consider all distances as expensive.
- Compute the distance from q to each candidate, retrieve the K closest objects.

3 Experimental results
- Experiment setup
- Documents
- MPEG7 Vectors

Experiment setup

Our indexes were developed in C# and ran under the mono 2.6.1 framework and linux 2.6.27 under Ubuntu/Linux 8.10 in a Xeon 2.33GHz with 8GiB of ram workstation.
The indexes run in main memory and without parallelization.
Two databases will be shown: documents for information retrieval, and MPEG7 vectors of images for multimedia information retrieval.

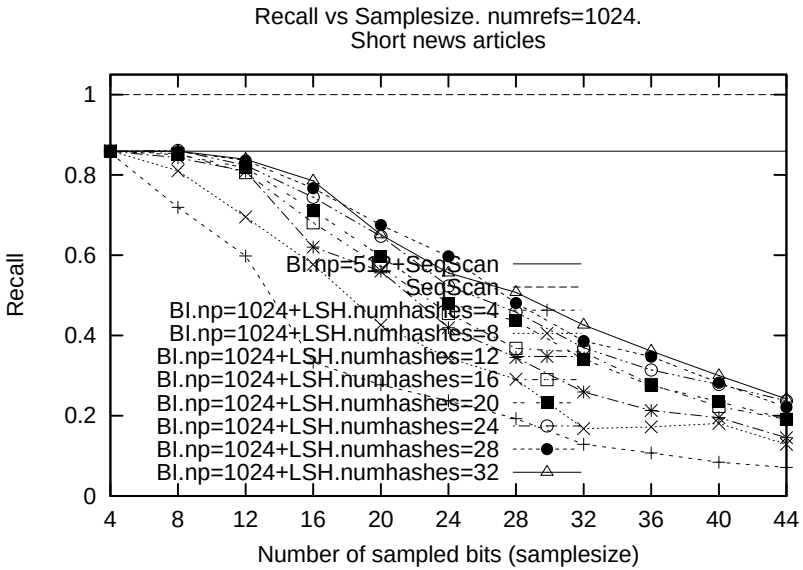
Documents

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

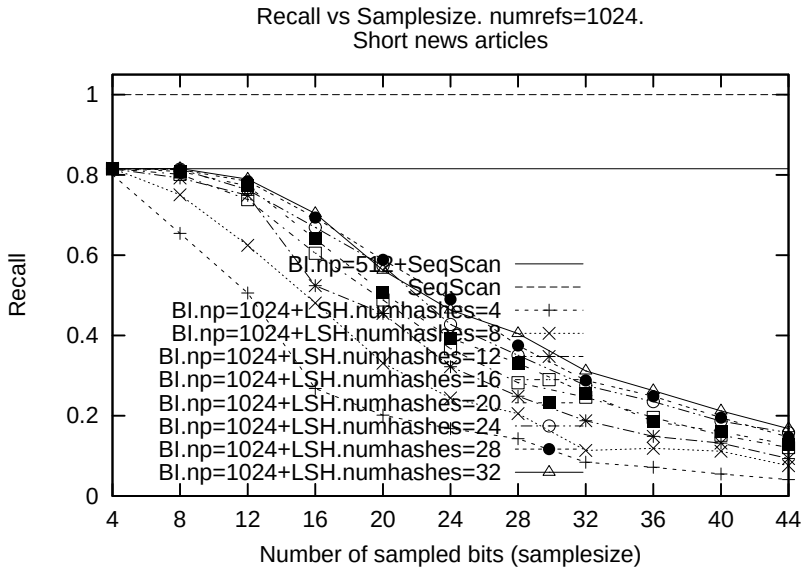
Description

We extracted 100 random documents as queries, these documents were not indexed.
In the next slide we show the recall for 10 and 30 nearest neighbors (a metric index, like BKT using a ring width of 0.001 needs to check up to 98% of the database for 30NN). We fix the number of distance computations to 1000 + the number of references.
For 30*NN* with recall bigger than 0.82 we need to review only the 8% of the database instead of the 98% in the alternative metric index (not shown for space constrains).

Recall



(a) Recall for 10NN



(b) Recall for 30NN

Figure 1: Experiment results for brief index against the two layer index using the documents $TF \times IDF$ collection and vector's angle as distance.

Time

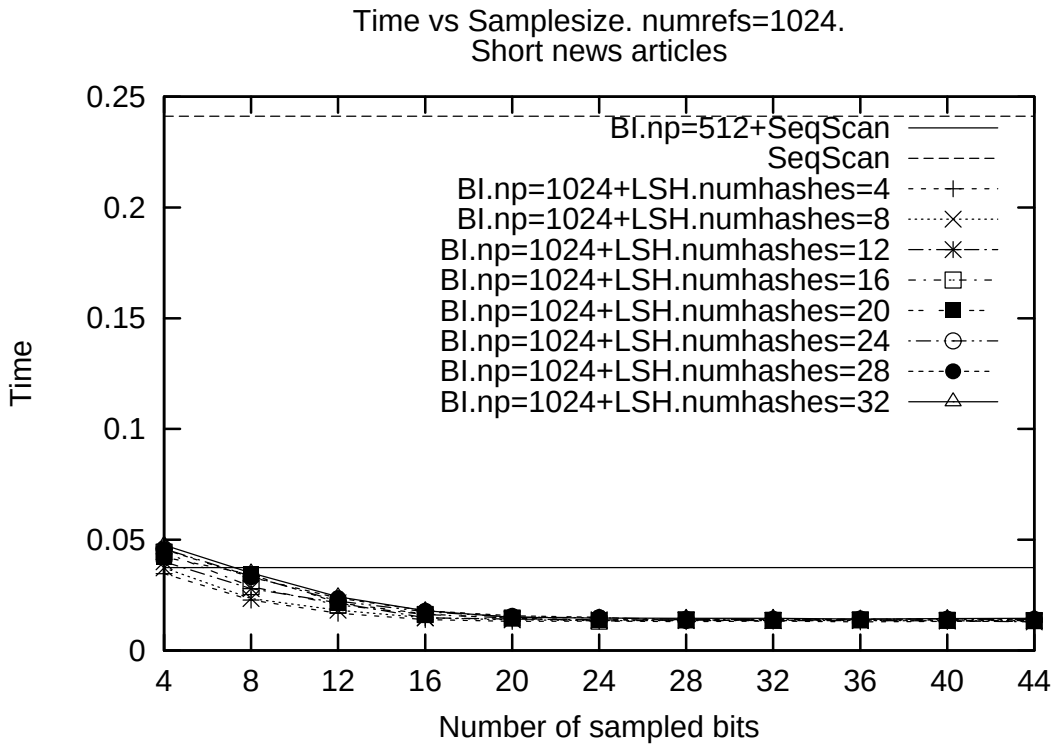
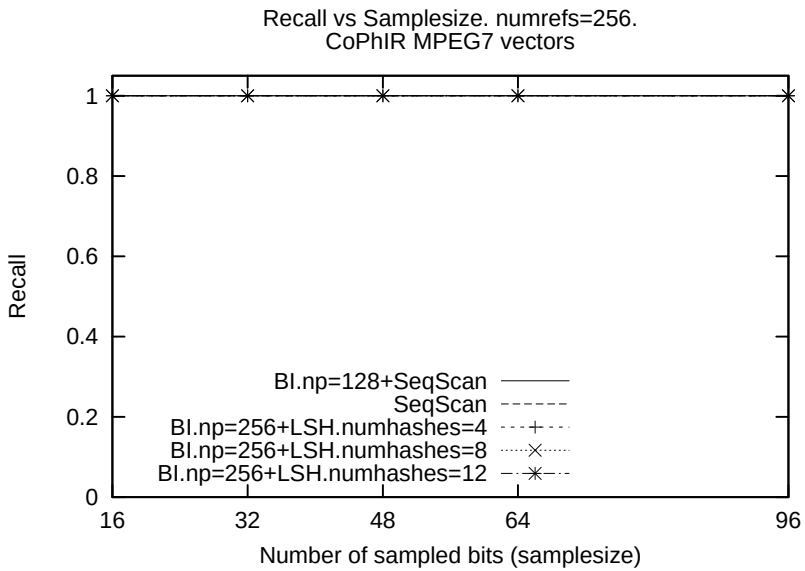


Figure 2: Experiment results for brief index against our two layer index using the documents $TF \times IDF$ collection and vector's angle as distance.

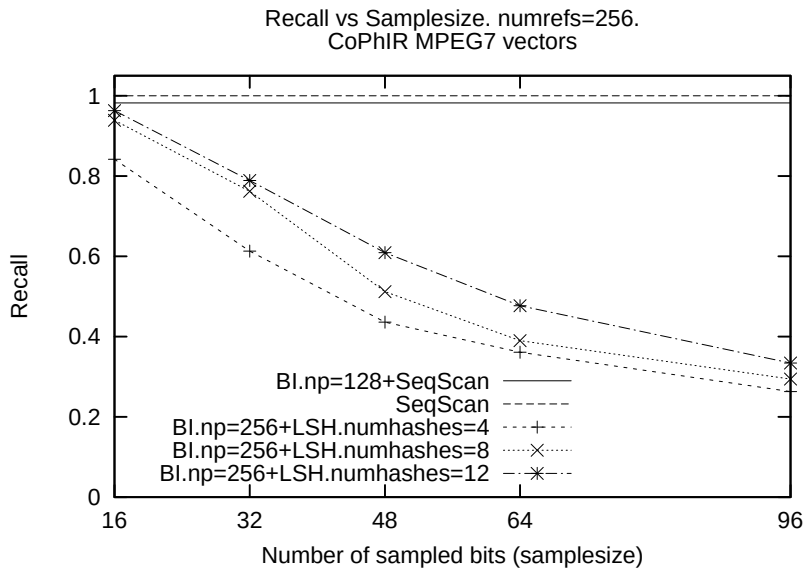
MPEG7 Vectors

A database of 10 million 208-dimensional vectors from the CoPhIR project [Bolettieri et al., 2009] were selected. We use L_1 as distance. Every single vector were created in a linear combination of five different MPEG7 vectors as described in [Bolettieri et al., 2009].
We choose the first 200 vectors from the database as queries. Search for 1,5,10, and 30 nearest neighbors were performed. The number of neighbors is a common number in interactive applications like multimedia information retrieval systems.
Traditional metric indexes like *BKT* doesn't perform well under the database's size. This database is difficult because the size, since it avoids the use of traditional structures with high space overhead.

Recall



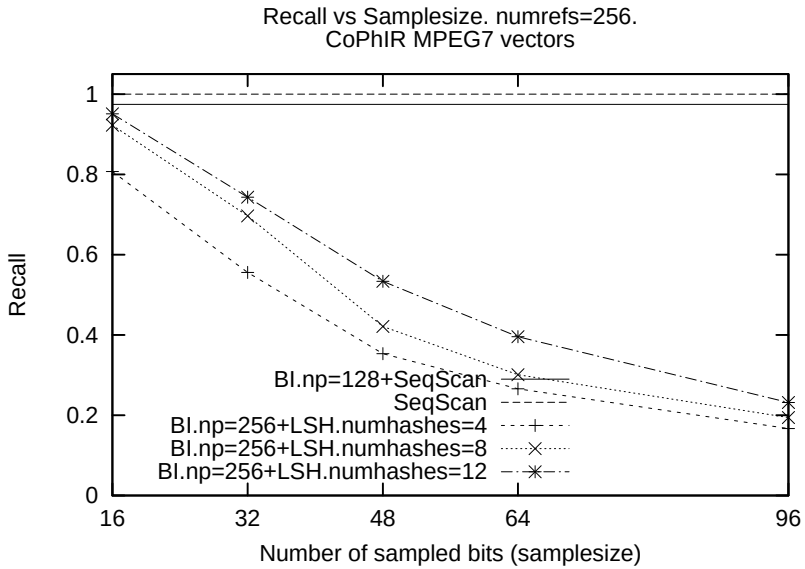
(a) Recall for matching the NN. 256 permutants



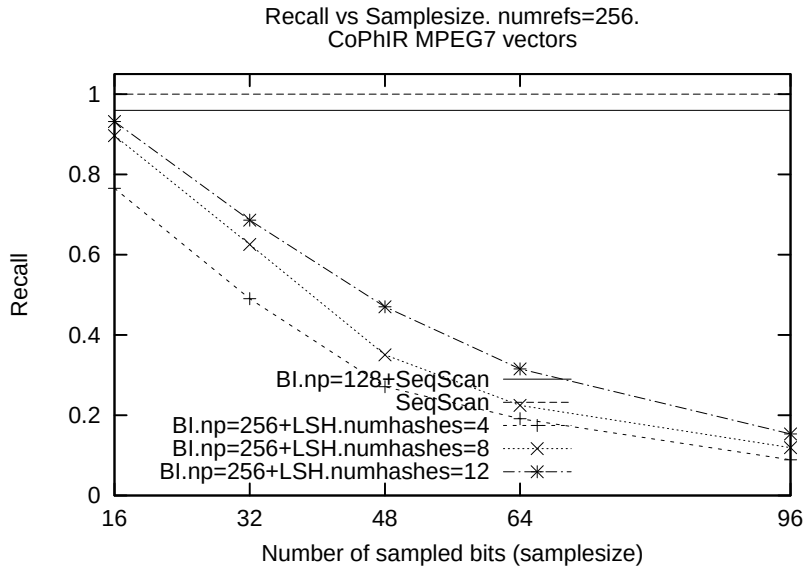
(b) Recall for matching the 5NN. 256 permutants

Figure 3: Results for brief index and our two layer index for CoPhIR 10M MPEG7 database, 1 and 5 Nearest neighbors.

Recall



(a) Recall for matching the 10NN. 256 permutants



(b) Recall for matching the 30NN. 256 permutants

Figure 4: Results for brief index and our two layer index for CoPhIR 10M MPEG7 database, 10 and 30 nearest neighbors.

Time

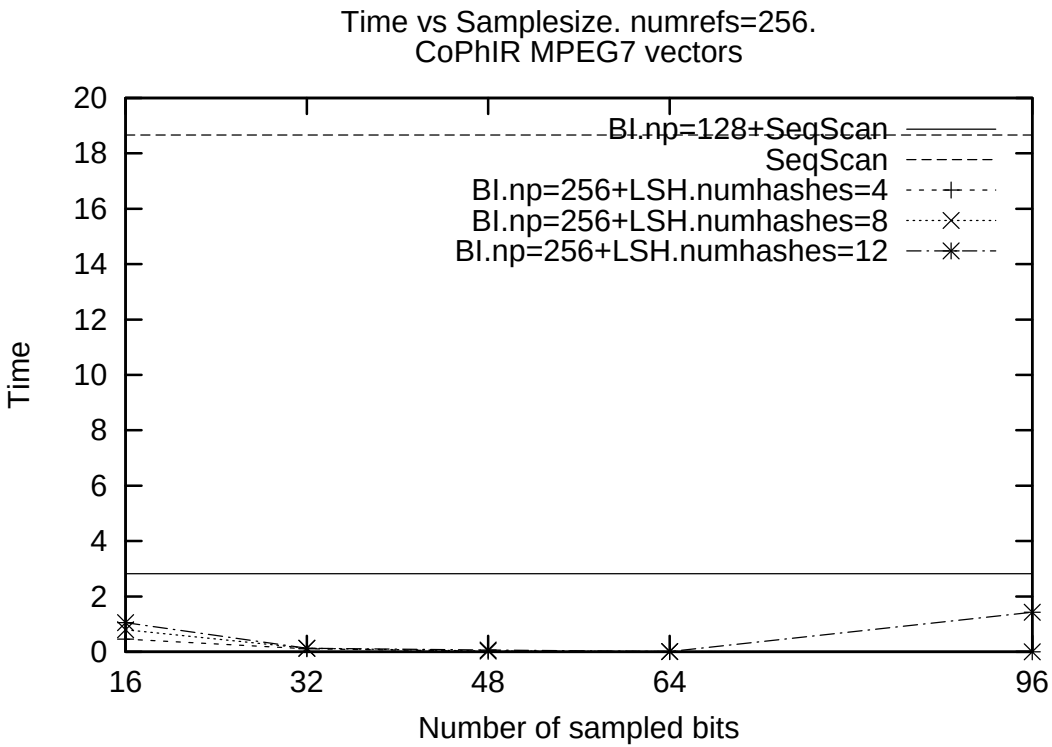


Figure 5: Time performance for brief index and the two layer index for MPEG7 vectors of 10M CoPhIR database.

Time (Continuation...)

For index with *samplesize* $= 32$, queries are solved in close to 0.1 seconds, and using *samplesize* $= 96$ over 0.002 seconds.
The recall behavior is explained using the number of candidates, since LSH layer never gives the entire set of requested candidates. For example *samplesize* $= 32$ achieves approximately 30000 candidates for the presented *numhashes*. Fixing *samplesize* $= 96$ yields to 500, 1100 and 1600 candidates for 4, 8, 12 *numhashes* respectively.
So, if we optimize the index for performance it needs to review 0.016% of the database. A higher recall can be obtained reviewing 0.3% of the objects.

4 Conclusions and Future Work

- Summary and Conclusions
- Future Work

Summary and Conclusions

- We presented an effective combination of indexes, LSH and permutations (brief permutations index), producing fast and scalable indexes for general metric spaces (and similarity space, since permutations do not use the triangle inequality).
- Even when our index is good for small databases, large databases show the real gain for our indexing technique.

Future Work

- An alternative quality assessment for inexact proximity queries is how far are the putative answer wrt the true answer.
- We always consider distances as expensive. We are conducting experiments considering cheap distances, showing the relation between distance counting and actual query time.
- A comparison between LSH for full and brief permutations is needed
- We plan to exploit the parallelism in the technique by using GPGPU and multi-core schemes.

Thanks for your attention!

... and sorry for the inconvenients...