Efficient Similarity Search by Reducing I/O with Compressed Sketches

Arnoldo José Müller-Molina, Takeshi Shinohara
Kyushu Institute of Technology
Department of Artificial Intelligence
Kawazu 680-4 Iizuka 820-8502, Japan
arnoldo@daisy.ai.kyutech.ac.jp

***Abstract*—*Sketches* are compact bit string representations of objects. Objects that have the same sketch are stored in the same database bucket. By calculating the hamming distance of the sketches, an estimation of the similarity of their respective objects can be obtained. Objects that are close to each other are expected to have sketches with small hamming distance values. This estimation helps to schedule the order in which buckets are visited during search time. Recent research has shown that sketches can effectively approximate L_1 and L_2 distances in high dimensional settings. A remaining task is to provide a general sketch for arbitrary metric spaces.**

This paper presents a novel sketch based on generalized hyperplane partitioning that can be employed on arbitrary metric spaces. The core of the sketch is a heuristic that tries to generate balanced partitions. The indexing method *AESA* stores all the distances among database objects, and this allows it to perform a small number of distance computations. Experimental evaluations show that given a good early termination strategy, our algorithm performs up to one order of magnitude fewer distance operations than AESA in string spaces. Comparisons against other methods show greater gains. Furthermore, we experimentally demonstrate that it is possible to reduce the physical size of the sketches by a factor of ten with different run length encodings.

***Keywords*-similarity search; k nearest neighbor; sketch; compression**

I. INTRODUCTION

Similarity search is the efficient retrieval of the closest objects to a query in a database. New similarity search applications are constantly being developed, ranging from language translation systems [1] to open source license violation detectors [2]. Similarity search is regarded as an important tool to address the problem of information overload.

In order to build efficient similarity search index implementations, it is necessary to take into account the computer hardware characteristics. When dealing with high dimensional spaces, for certain distributions and queries, hierarchical indexes perform worse than linear scan [3]. Therefore, some researchers have implemented sequential access methods that work on compact representations [4], [5], [6], [7]. Since the entries are smaller than the original objects, the matching takes less time as there is less data to read. Since CPU and secondary storage cache sub-systems can exploit sequential access patterns, these schemes are a natural fit for current hardware architectures. Very recently, *sketches*[8], [9], [10] have been proposed as a promising, space-efficient approach. A sketch is a binary string representation of an object. To estimate the similarity of two objects, the hamming distance of their sketches is calculated. Sketches with small hamming distance are expected to belong to objects that are close to each other. The hamming distance can be efficiently implemented with XOR and bit population counting operations that exploit bit parallelism.

In this paper we implement a sketch for general metric spaces by using *generalized hyperplane partitioning* [11]. This technique partitions the space into two sets based on a pair of distinguished objects called *pivots*. Sketch implementations exist for L_1 [8] and L_2 [12] spaces. To the best of our knowledge, this is the first sketch designed for arbitrary metric spaces. The core of our technique is a pivot selection strategy that tries to generate balanced partitions. Experimental evaluations show that our technique performs up to one order of magnitude fewer distance operations than AESA [13] in string spaces if a good early termination strategy is employed. If some error is allowed, further improvements can be obtained. Although our method outperforms or matches AESA with sketches of 64 bits, smaller sketches are still competitive.

To lower I/O operations, it is necessary to reduce the physical size of the sketch. We compare different run length encodings and their effects on sketch size. In our experiments, we show that sketches can be compressed to about a tenth of their original size. For example, in a database of 500 million unique sketches of 30 bits, a compressed sketch index requires about 130MB of disk space (2.1 bits per sketch). The original database uses 220 GB of disk space.

II. BACKGROUND

A large number of similarity search algorithms [14], [3], [15], [16] exist. As the dimensionality increases, for certain queries and data distributions the performance of hierarchical indexes decreases and sequential scans become more effective [3], [4]. The *VA-file* [4] is a sequential scan method that compresses the original feature vectors into quantized (compact) representations. For each dimension, a small number of bits is assigned. Each quantized vector represents a bucket that holds one or more objects. During search time, a *filter and refine* [17] procedure is applied on the quantized vectors. The *VA$^+$-file* [5] is an improvement that takes into account the discriminatory level of different features and data distribution. The *IQ-tree* [6] adds an extra level of minimum bounding rectangles (MBRs) that allows the algorithm to quantize different sections of the index in a different way. The *D-Index* [18] resembles sequential search only if the query radius does not exceed a small value ρ. The *lcluster* [7] can also considered as a sequential method because its structure consists of a list of compact ball partitions. Finally, *distance permutations* [19] is a method that stores compact representations of objects based on the proximity to a set of pivots.

When embedding algorithms [20], [21], [22], [23] transform an object into a feature vector, each resulting dimension of the feature vector usually becomes a real or integer number. In the case of the VA-file, the authors employed 4 to 8 bits for each quantized dimension in their experiments. To further reduce the size of the embedding, approximate methods called *sketches*, that transform objects from specific metric spaces (L_1, L_2) to binary strings have been proposed [8], [9], [10]. The similarity of the objects can be estimated by using the hamming distance of their respective sketches. The hamming distance can be computed very efficiently by exploiting bit parallelism of the hardware. At search time, a filter and refine procedure finds the closest objects to a query.

The main advantage of sketches is that they can be one order of magnitude smaller than the original feature vectors [12]. Given this limited amount of information, it is still possible to build *asymmetric estimators* (lower bounds) of the real distance function as recently shown by Dong *et al.* [12]. Sketches are related to *locality sensitive hashing* (LSH) [24]. Wang *et al.* argue that LSH only works efficiently in low dimensional spaces [10]. Closely related, *signature files* [25] can be considered as the precursors of sketches. Initially, signature files were created to index text documents, as an alternative to inverted indexes. Recently, signature files have been employed to perform similarity search on multimedia databases [26].

A. Metric Spaces

Let $\mathcal{M} = (\mathcal{D}, d)$ be a *metric space* for a domain of objects $\mathcal{D}$ and $d : \mathcal{D} \times \mathcal{D} \to \mathbb{R}$, *a total distance function* that satisfies the following properties:

$$\begin{aligned}
&\forall x, y \in \mathcal{D}, d(x,y) \geq 0 \\
&\forall x, y \in \mathcal{D}, d(x,y) = d(y,x) \\
&\forall x, y \in \mathcal{D}, d(x,y) = 0 \iff x = y \\
&\forall x, y, p \in \mathcal{D}, d(x,y) \leq d(x,p) + d(p,y)
\end{aligned}$$

Given a collection $X \subseteq \mathcal{D}$, a *k-nearest neighbor* (k-NN) query returns the k closest elements to the query object q, or the set $R \subseteq X$ such that $|R| = k$, for any $x \in R$ and any $y \in X - R : d(q,x) \leq d(q,y)$.

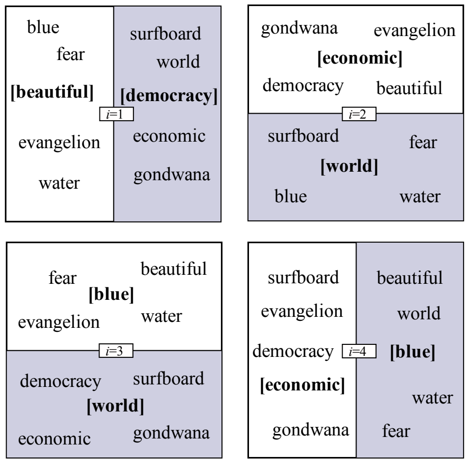

Figure 1. Illustration of hyperplane partitioning in string (Levenshtein) spaces. Each square encloses a different partition of the set. Words enclosed by brackets indicate the pivots.

III. Proposed Sketch: GHS

In this section, we introduce a sketch algorithm for arbitrary metric spaces. Our sketch is based on the *generalized hyperplane partitioning* [11] scheme. Given a collection $X \subseteq \mathcal{D}$, two *pivots* $p_0, p_1 \in \mathcal{D}$, this partitioning divides the objects $o \in X$ into two sets:

$$\begin{aligned}
S_0 &= \{o \in X | d(p_0, o) \leq d(p_1, o)\} \\
S_1 &= \{o \in X | d(p_0, o) > d(p_1, o)\}
\end{aligned}$$

For example, Figure 1 displays the four possible partitions for a small set of words in string spaces. Each square holds a different partition for the same set of words. In this example, we are taking the pivots from X.

For a sketch of m dimensions, we generate m independent hyperplane partitions. Each resulting dimension of the sketch is determined by the partition in which the object falls. Specifically,

GHS (Sketch) 1: The generalized hyperplane sketch (GHS) for an object $x \in \mathcal{D}$, is a bit vector $\sigma(x) \in \{0,1\}^m$, where each bit $\sigma_i(x)$ is generated by the function:

$$\sigma_i(x) = \begin{cases} 0 & \text{if } d(p_{i0}, x) \leq d(p_{i1}, x) \\ 1 & \text{if } d(p_{i0}, x) > d(p_{i1}, x) \end{cases} \quad \forall i = 1, 2, \ldots, m$$

where p_{i0}, $p_{i1} \in \mathcal{D}$ are pivots selected for the dimension i of the sketch. Each function σ_i partitions the dataset into the sets S_{i0} and S_{i1}. For two objects $x, y \in \mathcal{D}$, their sketch distance is defined as:

$$d_\sigma(x,y) = \sum_{i=1}^{m} \sigma_i(x) \oplus \sigma_i(y).$$

If we consider the shaded partitions in Figure 1 as pivots p_{i1}, then the sketch for the strings "water" and "fear" is "0101". The sketch for the strings "democracy" and "gondwana" is "1010". The sketch distance of both sketches is four. We employ the sketch distance to find the closest buckets to a query at search time. We then refine the final solution by using the distance function provided by the user. Our sketch does not make explicit use of the triangle inequality.

A. Pivot Selection

The generalized hyperplane partitioning does not guarantee a balanced split [11], [3], [14]. Uhlmann [11] proposed to use a median value c so that $d(p_0, o) - d(p_1, o) \leq c$ is true for about half of the dataset. On the other hand, Zezula *et al.* [14] suggested to divide the space in a balanced way by selecting suitable pivots as another way of dealing with this issue. In this paper we take the second approach by implementing *rf01*, a pivot selection heuristic that attempts to produce balanced partitions. Our algorithm is related to the *incremental* pivot strategy proposed by Bustos *et al.* [27]. The main difference is the fitness function employed. We consider only one pair of pivots at a time. This implies that each partition i of the sketch is determined independently of the others. We first randomly select a subset S of size l from the database X. This set is used to estimate the quality of the partition. Given two randomly selected pairs of pivots $P = \{p_0, p_1\}$ and $Q = \{q_0, q_1\}$, we consider P better than Q if $f(P, Q, S)$ is true (Figure 2). While keeping the best pivot set found so far we generate random pairs of pivots for n iterations until the best candidate has been found. In Figure 2 (lines 4, 5), we estimate the quality of the partition by calculating the differences of the sizes of the partitions. If the pivot sets balance the data in a similar way (line 6), then we proceed to choose the pivot set with the largest inter-pivot distance (line 7). Otherwise, the pivot set with the most balanced partition is chosen (line 9). The process is repeated m times. Based on the previous discussion, our pivot selection algorithm requires $2lmn$ distance computations.

B. Quality

We propose two metrics that estimate the overall quality of the GHS partitioning for a specific dataset. Let b the number of sketches of m bits generated for a given database X. The *spread* of the index is defined as: $b/min(2^m, |X|)$. This metric indicates the ratio of buckets filled from all the possible number of buckets available. The *distortion* is the normalized average difference between the cardinality of the partitions of each dimension of the sketch:

$$\frac{\sum_{i=1}^{m} ||S_{i0}| - |S_{i1}||}{|X| \times m}.$$

The distortion gives us an idea on how well the space is divided in each hyperplane partition. A good pivot selection heuristic is expected to have a distortion of 0.

```
    $S \subseteq \mathcal{D}$: sample of objects, $P, Q \subseteq \mathcal{D}$: pivot sets
    Returns true if $P$ is better than $Q$, otherwise false.
 1: function $f(P = \{p_0, p_1\}, Q = \{q_0, q_1\}, S)$
 2:     Calculate $S_{p0}$ and $S_{p1}$ based on $P$ and $S$.
 3:     Calculate $S_{q0}$ and $S_{q1}$ based on $Q$ and $S$.
                                  ▷ Get the difference of partition sizes:
 4:     $st_p \leftarrow ||S_{p0}| - |S_{p1}||$
 5:     $st_q \leftarrow ||S_{q0}| - |S_{q1}||$
 6:     if $st_p = st_q$ then        ▷ Equally balanced partitions
                                  ▷ Greater inter pivot distance is better
 7:         return $d(p_0, p_1) > d(q_0, q_1)$
 8:     else
 9:         return $st_p < st_q$     ▷ $S$ better balanced with $P$?
10:     end if
11: end function
```

Figure 2. Pivot selection algorithm: rf01

C. Sketch Compression

A sketch index is a sequence of binary strings of m bits. We note that binary strings can be interpreted as positive integers. Therefore, compression techniques that work on sorted, positive integers such as those used in bitmap indexes [28] or inverted indexes [29] can be applied to sketches. We compare the traditional Gamma and Delta [30] codes with d-gaps and the *Word-Aligned Hybrid* (WAH) code [28].

IV. EXPERIMENTS

In this section, we compare our pivot selection method against other similarity search methods in two SISAP datasets and a synthetic dataset. We first measure the efficiency of each method in different error settings to demonstrate the overall behavior of each algorithm. Then, we focus on our method by measuring performance as the database size grows. Finally, we evaluate the compression algorithms mentioned in Section III-C. All our experiments focus on k-NN queries.

A. Datasets

We employ two SISAP datasets and a synthetic dataset. Table I displays a summary of the datasets.

Strings: From the SISAP "dictionaries" dataset we use the "dutch" set (200000 words). The experiments of Sections IV-E and IV-F employ a concatenation of all the dictionaries (800000 words, "dict"). The distance function is the Levenshtein distance.

Trees: From the SISAP "slices" dataset we take 100000 trees of 300 nodes or less for validation. This subset is called "trees". The experiments of Sections IV-E and IV-F employ up to 300000 trees ("trees-full"). The distance function is called mtd [31]. It is roughly a sparse vector of at least 400000 dimensions with the L_1 distance. The dimension is always larger than the number of items in the dataset and

Table I
SUMMARY OF DATASETS. SPREAD AND DISTORTION FOR RF01 AND m.

Dataset	$\lvert X\rvert$	Size	spread	distortion	m
dutch	200000	2MB	0.999	0.09	64
dict	800000	8MB	0.96	0.05	64
trees	100000	5MB	0.54	0.07	64
trees-full	300000	17MB	0.42	0.02	64
vectors	1 billion	223GB	0.48	0.03	30

the intrinsic dimensionality is 0.45. The trees were extracted from program fragments [32].

Vectors: Vectors of 120 dimensions, uniformly generated. Each dimension holds two bytes and we employed the L_1 distance. The entire collection consists of one billion objects. This dataset is only used in the compression experiments.

B. Methods

We compare the following five methods:

- GHS sketch with the pivot selection strategy rf01. Unless otherwise noted, we employ sketches of 64 bits.
- Distance Permutations [19] (per) with 64 pivots. We used the incremental pivot selection strategy [27]. We employ one byte per dimension and therefore each compact representation of this method is eight times larger than the sketches.
- Symmetric L_2 sketch [12] with 64 bits on a projected space of 64 dimensions (64 pivots). The intended use of this sketch was for L_2 distance. Since we are not estimating the distance function, this is not a major concern. The best W parameter we found was 250 for trees and 60 for strings.
- Slim Tree [33], using the Arboretum library from Vieira *et al.*[1].
- AESA [13] employing the implementation from SISAP.

Pivot selection strategies were configured with 4000 random selections (n) on a sample of 1000 objects (l). The first three methods were implemented using the library OBSearch [2].

C. Evaluation

Since sketches are inherently approximate methods, we employ the *error on the position* (EP) [14] to measure the quality of the results. Consider a result set S^A returned by an approximate algorithm. Let OX be the ordered list containing all elements of X, ordered by increasing distance from the query. If $L(o_i^A)$ denotes the position of object $o_i^A \in S^A$ in the ordered list L, the EP measure is denoted as:

$$EP = \frac{\sum_{i=1}^{|S^A|}(OX(o_i^A) - S^A(o_i^A))}{|S^A| \times |X|}.$$

It is not clear yet which is the best way for sketch algorithms to return results within a specific EP value[3]. Therefore, we sort all the buckets of the database based on the hamming distance or the Spearman footrule distance to the query. Then, we read one by one the sorted buckets and match their objects against the query. We stop the process if the current EP value is less or equal than a specified threshold. Then, we proceed to count the number of disk access operations (buckets read), the number of objects read from secondary storage, and the number of distance computations performed. We use the *improvement in efficiency* measurement [14]: $IE = \frac{cost(Q)}{cost^A(Q)}$ where $cost(Q)$ and $cost^A(Q)$ denote respectively the cost for the precise and the approximate execution of a set of queries. We employ three definitions of cost: the improvement on the number of disk access operations IE_{acc}, the number of objects read IE_{obj} and the number of distance computations performed IE_{dis}. We use as baselines the Slim Tree and AESA methods. All our time measurement experiments are done on a PC with an Intel Core 2 Quad CPU (2.66 Ghz) and 4GB of memory running Ubuntu Linux 8.04. The programs were written in Java and were executed on Java JDK 1.6 (64 bit). We take the average of 1000 different queries.

D. Efficiency

For the “dutch” dataset we generated 1000 queries with 10 random modifications using the tool provided by the SISAP library. The “trees” dataset was evaluated with 1000 randomly selected queries that are removed from the database. In Figures 3 and 4, the IE_{acc} and IE_{obj} over the Slim Tree is shown for rf01, distance permutations (per) and the symmetric L_2 sketch for the “dutch” and “trees” datasets. In both datasets, rf01 is the clear winner. It shows improvements of up to one order of magnitude over per and the L_2 sketch for small EP. L_2 sketch and per are very close to each other. In any case, all the methods provide considerable gains when compared to the Slim Tree. For example, in Figure 3 the results for IE_{obj} when $EP = 0.0005$ show that rf01 is reading 10000 times fewer objects than the Slim Tree. For the same EP value, per and the L_2 sketch have an IE_{obj} of respectively, 800 and 900. Even for small EP, our method obtains improvements of two to three orders of magnitude over the Slim Tree.

We can observe that the IE_{obj} is higher than the IE_{acc} for the “dutch” dataset. The opposite occurs for the “trees” dataset. The notion of spread can be useful to understand this (Table I). First, note that the distortion levels are relatively low for both datasets. Nevertheless, the spread in the “trees” dataset is much lower than for the “dutch” dataset. This means that more objects per bucket are available in the “trees” dataset. Fewer access operations will be executed because each bucket holds more objects. Moreover, distance computations increase as the discriminatory power of the sketch is lower.

In Figure 5, we show the IE_{dist} results of comparing rf01 against AESA for different values of m. The datasets

[1]http://gbdi.icmc.usp.br/arboretum/index.php
[2]http://obsearch.net
[3]A possible solution is presented in appendix A

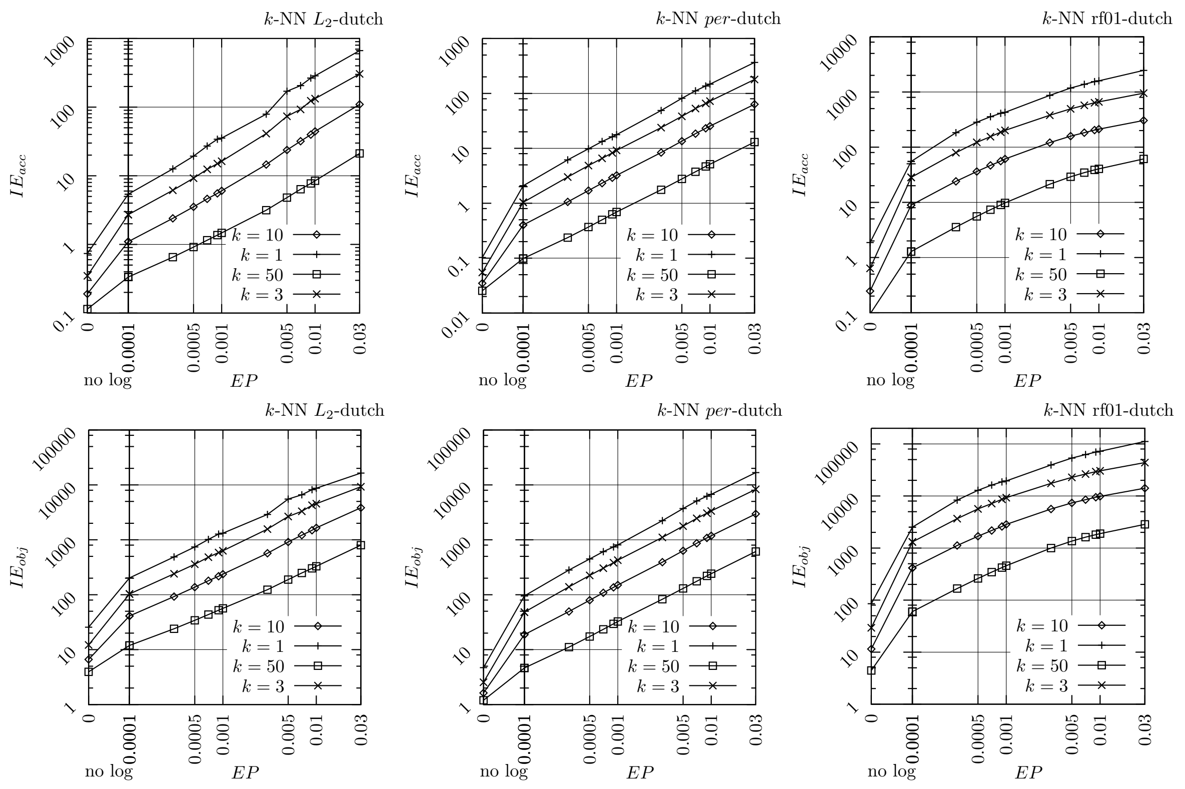

Figure 3. IE_{acc} and IE_{obj} over the Slim Tree for the "dutch" dataset (200000 objects).

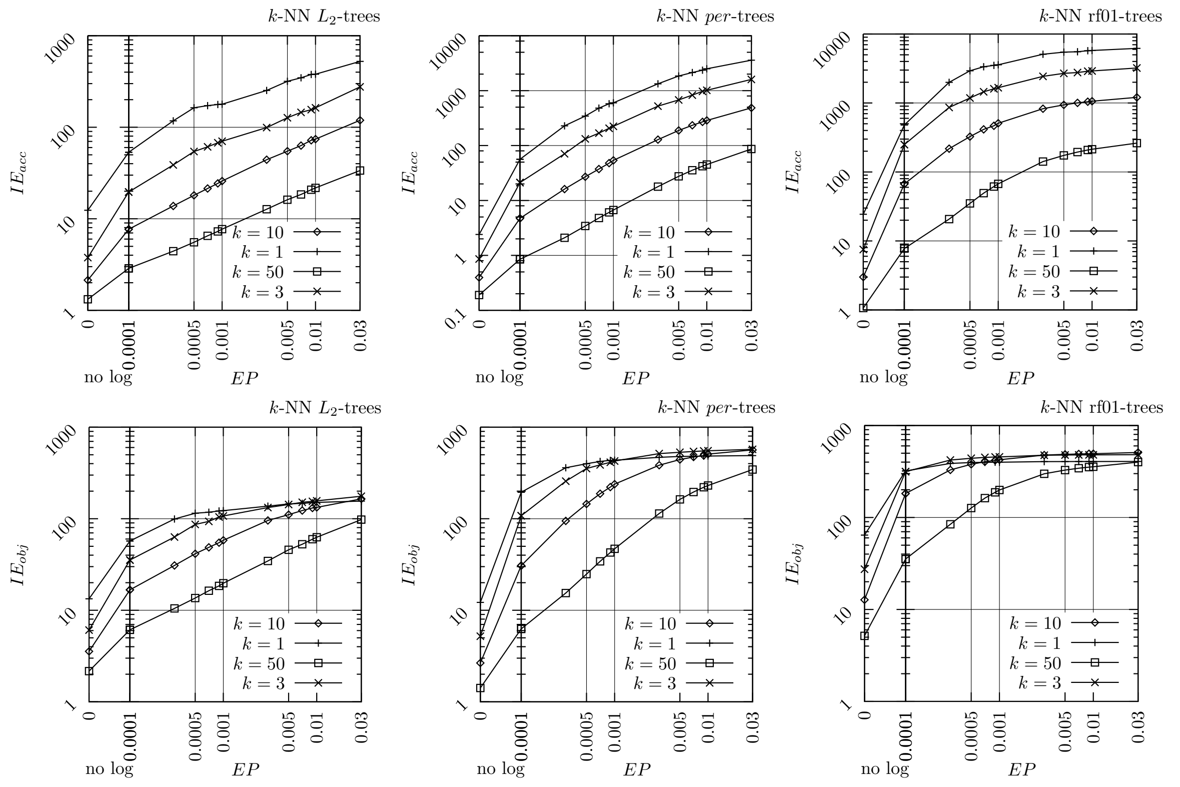

Figure 4. IE_{acc} and IE_{obj} over the Slim Tree for the "trees" dataset (100000 objects).

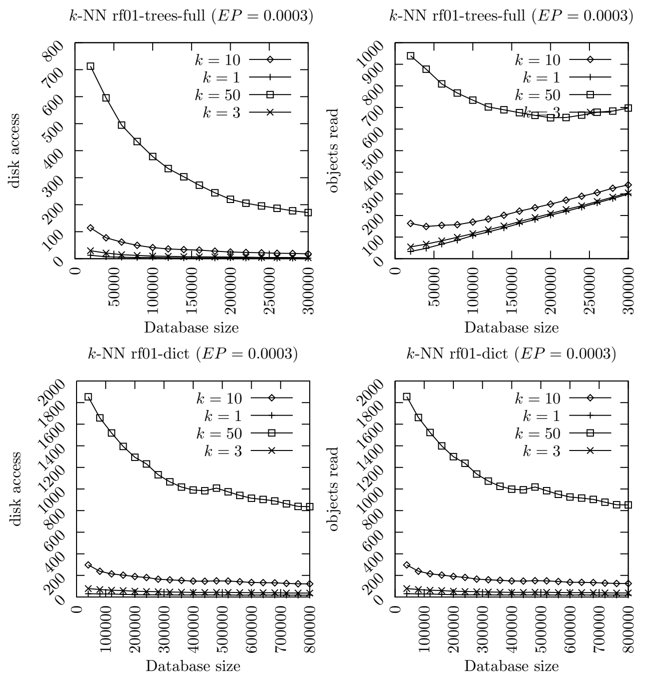

Figure 6. Performance when database size increases ("trees-full" and "dict" datasets).

were reduced to 10000 objects. The most relevant result is achieved for the "dutch" dataset when $m = 64$ and $k = 1$. Our method achieved one order of magnitude improvement over AESA for $EP = 0$. Lower values of m still perform well. In the case of "trees", our method is able to calculate about the same number of distance computations than AESA. Considering the small size of the index, this is still a promising result.

E. Database Growth

Using the same queries of the previous section, we first create a small database of 20000 objects (datasets "dict" and "trees-full" and we extract pivots from this set. We then continue inserting objects in increments of 20000 objects. The results for rf01 can be observed in Figure 6 for $EP = 0.0003$. For the "dict" dataset, the amount of objects read and the access count remain constant as the database grows. In the case of the "trees-full" dataset, only the access count remains constant. For the reasons explained in the previous experiment, buckets grow as we add more data, and therefore the number of objects read increases. The decreasing behavior for $k = 10$ and $k = 50$ occurs because the probability of getting a sketch closer to a query increases as we add more data. Since closer sketches are more likely to contain data that is closer to the query, the number of buckets/objects that must be accessed slowly decreases because the search procedure finds the result sets early on.

Table II
SKETCH COMPRESSION FOR RF01, $m = 30$. THE VALUE BETWEEN PARENTHESES IN THE FIRST COLUMN IS THE SIZE OF THE SKETCH SET WITHOUT COMPRESSION.

Data Set	Method	Size	Milliseconds
dict(2.4MB)	BitSet	134MB	151.73
	WAH	4MB	67.79
	Delta	1.1MB	23.97
	Gamma	1.2MB	28.67
trees-full(187Kb)	BitSet	134MB	151.73
	WAH	264Kb	5
	Delta	79Kb	2
	Gamma	92Kb	.72
vectors(1.7GB)	BitSet	134MB	14174
	WAH	131MB	20917
	Delta	251MB	10534
	Gamma	204MB	10890

F. Compression

We use our pivot selection criterion to generate a set of sketches. In the case of the "vectors" dataset, we extract pivots from the first million objects. We evaluate different compression schemes and measure the sizes of the compressed sketches. We also measure the time it takes to sequentially scan the compressed sketch set to get the closest 1000 sketches to a query. Due to platform limitations, the sketch size was set to 30 bits. We compare the methods listed in Section III-C along with a bit vector (the bit i is set if the corresponding sketch exists). We show the results in Table II. We can see that delta and gamma provide better compression than WAH in "dict" and "trees-full" datasets. On the other hand, WAH gives much better results for the "vectors" dataset. In the case of this dataset, sketches are already two orders of magnitude smaller than the original database (223GB). Nevertheless, compression methods are able to reduce the size of the sketch to roughly a tenth of its original size.

V. CONCLUSIONS

The main contribution of this paper is the comparison of sketches against established similarity search techniques. Our experiments show that sketches are not only a competitive approach, their space usage can also be substantially reduced with standard compression techniques for inverted files or bitmap indexes. In our experiments, sketches were up to three orders of magnitude smaller than the original objects. Our sketch based on hyperplane partitioning is the first to be applicable to general metric spaces. Our simple pivot selection strategy was capable to outperform AESA by a factor of 10 in exact k-NN and for small k. We reduce I/O by compressing the sketch and by minimizing the number of objects that must be read from the hard disk. Even with small error thresholds, our sketch read 100-10000 times fewer objects than the Slim Tree. A simple heuristic that allows GHS to search within some EP value is presented in appendix A. In the future, we would like to develop better

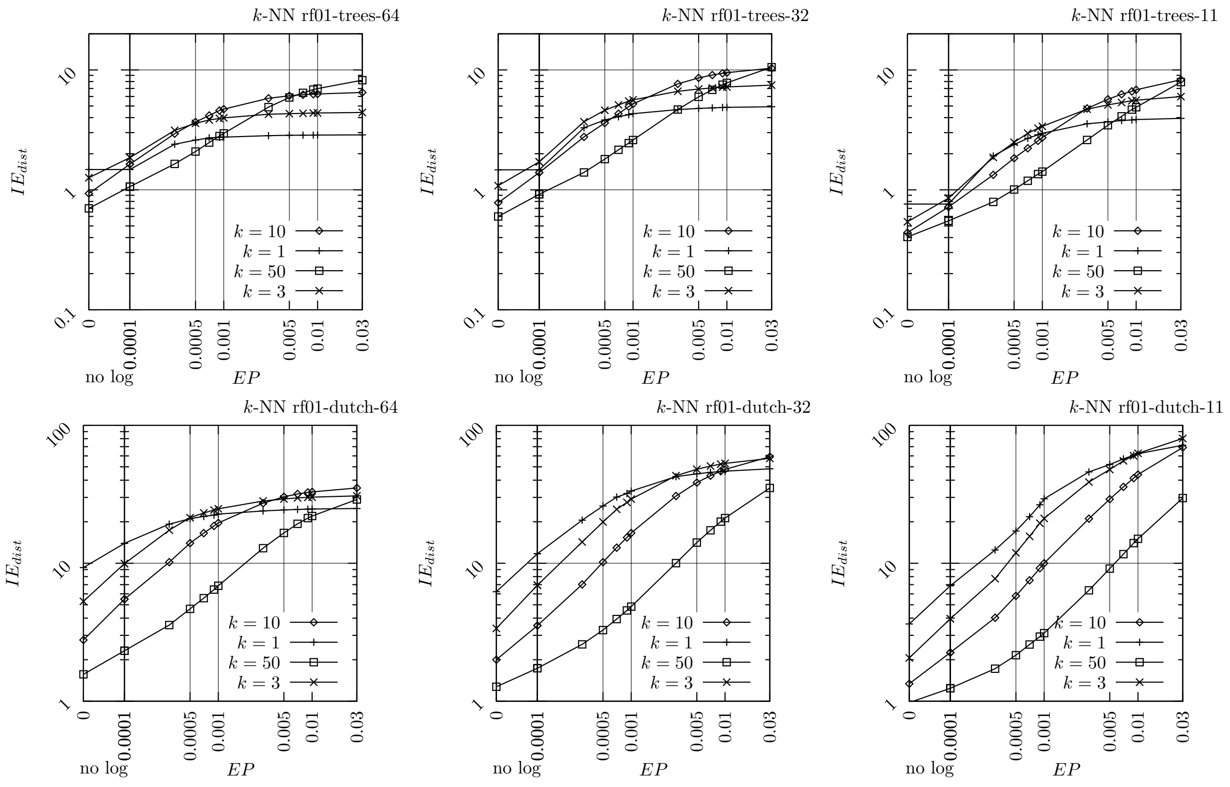

Figure 5. IE_{dist} over AESA for the "trees" and "dutch" dataset (10000 objects) for $m = 64, 32, 11$.

heuristics that achieve this goal. In addition, we are also interested in studying alternative pivot selection strategies.

VI. Acknowledgments

The authors would like to thank the reviewers for all their valuable feedback. This work was partially supported by the Japanese Ministry of Education, Culture, Sports, Science and Technology (grant # 040506).

References

[1] M. M. Gonzalez and T. Endo, "Tense and mood decision with similarity search in japanese to spanish machine translation," in *IMECS*, 2009, pp. 86–91.

[2] A. J. Müller-Molina and T. Shinohara, "Fast approximate matching of programs for protecting libre/open source software by using spatial indexes," in *SCAM '07*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 111–122.

[3] H. Samet, *Foundations of Multidimensional and Metric Data Structures*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005.

[4] R. Weber, H.-J. Schek, and S. Blott, "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces," in *VLDB '98*. San Francisco, CA, USA: Morgan Kaufmann Publishers, 1998, pp. 194–205.

[5] H. Ferhatosmanoglu, E. Tuncel, D. Agrawal, and A. El Abbadi, "Vector approximation based indexing for non-uniform high dimensional data sets," in *CIKM '00*. New York, NY, USA: ACM, 2000, pp. 202–209.

[6] S. Berchtold, C. Böhm, H. V. Jagadish, H. peter Kriegel, and J. S, "Independent quantization: An index compression technique for high-dimensional data spaces," in *ICDE*. Washington, DC, USA: IEEE Computer Society, 2000, pp. 577–588.

[7] E. Chávez and G. Navarro, "A compact space decomposition for effective metric indexing," *Pattern Recogn. Lett.*, vol. 26, no. 9, pp. 1363–1376, 2005.

[8] Q. Lv, M. Charikar, and K. Li, "Image similarity search with compact data structures," in *CIKM '04*. New York, NY, USA: ACM, 2004, pp. 208–217.

[9] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Efficient filtering with sketches in the ferret toolkit," in *MIR '06*. New York, NY, USA: ACM, 2006, pp. 279–288.

[10] Z. Wang, W. Dong, W. Josephson, Q. Lv, M. Charikar, and K. Li, "Sizing sketches: a rank-based analysis for similarity search," *SIGMETRICS*, vol. 35, no. 1, pp. 157–168, 2007.

[11] J. Uhlmann, "Satisfying general proximity/similarity queries with metric trees," *Inf. proc. letters*, vol. 40, no. 4, pp. 175–179, 1991.

[12] W. Dong, M. Charikar, and K. Li, “Asymmetric distance estimation with sketches for similarity search in high-dimensional spaces,” in *SIGIR ’08*. New York, NY, USA: ACM, 2008, pp. 123–130.

[13] E. V. Ruiz, “An algorithm for finding nearest neighbours in (approximately) constant average time,” *Pattern Recogn. Lett.*, vol. 4, no. 3, pp. 145–157, 1986.

[14] P. Zezula, G. Amato, V. Dohnal, and M. Batko, *Similarity Search: The Metric Space Approach*. Secaucus, NJ, USA: Springer-Verlag, 2005.

[15] G. R. Hjaltason and H. Samet, “Index-driven similarity search in metric spaces (survey article),” *ACM Trans. Database Syst.*, vol. 28, no. 4, pp. 517–580, 2003.

[16] E. Chavez, G. Navarro, R. Baeza-Yates, and J. L. Marroquin, “Searching in metric spaces,” *ACM Comput. Surv.*, vol. 33, no. 3, pp. 273–321, 2001.

[17] J. A. Orenstein, “Redundancy in spatial databases,” *SIGMOD Rec.*, vol. 18, no. 2, pp. 295–305, 1989.

[18] V. Dohnal, C. Gennaro, P. Savino, and P. Zezula, “D-index: Distance searching index for metric data sets,” *Multimedia Tools Appl.*, vol. 21, no. 1, pp. 9–33, 2003.

[19] E. Chávez, K. Figueroa, and G. Navarro, “Effective proximity retrieval by ordering permutations,” *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 30, no. 9, pp. 1647–1658, 2008.

[20] C. Faloutsos and K.-I. Lin, “Fastmap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets,” in *SIGMOD*. New York, USA: ACM, 1995, pp. 163–174.

[21] T. Shinohara, J. Chen, and H. Ishizaka, “H-map: A dimension reduction mapping for approximate retrieval of multi-dimensional data,” in *DS ’99*. London, UK: Springer-Verlag, 1999, pp. 299–305.

[22] T. Shinohara and H. Ishizaka, “On dimension reduction mappings for approximate retrieval of multi-dimensional data,” in *Progress in Discovery Science*. London, UK: Springer-Verlag, 2002, pp. 224–231.

[23] G. R. Hjaltason and H. Samet, “Properties of embedding methods for similarity searching in metric spaces,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 25, no. 5, pp. 530–549, 2003.

[24] A. Gionis, P. Indyk, and R. Motwani, “Similarity search in high dimensions via hashing,” in *VLDB ’99*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 518–529.

[25] C. Faloutsos and S. Christodoulakis, “Signature files: an access method for documents and its analytical performance evaluation,” *ACM Trans. Inf. Syst.*, vol. 2, no. 4, pp. 267–288, 1984.

[26] A. Davidson, J. Anvik, and M. A. Nascimento, “Parallel traversal of signature trees for fast cbir,” in *MULTIMEDIA ’01*. New York, NY, USA: ACM, 2001, pp. 6–9.

[27] B. Bustos, G. Navarro, and E. Chávez, “Pivot selection techniques for proximity searching in metric spaces,” *Pattern Recogn. Lett.*, vol. 24, no. 14, pp. 2357–2366, 2003.

[28] K. Wu, E. J. Otoo, and A. Shoshani, “An efficient compression scheme for bitmap indices,” University of California, Berkeley, Tech. Rep., 2004.

[29] N. Ziviani, E. S. de Moura, G. Navarro, and R. Baeza-Yates, “Compression: A key for next-generation text retrieval systems,” *Computer*, vol. 33, no. 11, pp. 37–44, 2000.

[30] P. Elias, “Universal codeword sets and representations of the integers,” *IEEE Transactions on Information Theory*, vol. 21, no. 2, pp. 194–203, 1975.

[31] A. J. Müller-Molina, K. Hirata, and T. Shinohara, “A tree distance function based on multi-sets,” in *ALSIP’08, PAKDD Workshops*, 2008, pp. 90–100.

[32] A. J. Müller-Molina and T. Shinohara, “On approximate matching of programs for protecting libre software,” in *CASCON ’06*. New York: ACM Press, 2006, pp. 275–289.

[33] C. Traina, Jr., A. J. M. Traina, B. Seeger, and C. Faloutsos, “Slim-trees: High performance metric trees minimizing overlap between nodes,” in *EDBT ’00*. London, UK: Springer-Verlag, 2000, pp. 51–65.

APPENDIX

A. *k-NN Algorithm*

In this section we describe a simple early termination strategy algorithm that allows GHS to perform k-NN searches within a specified EP value. Dong *et al.* employed [12] the following approach for k-NN queries:

- Select the $h = t \times k$ buckets with the smallest sketch distance to the query sketch.
- Compare the objects stored in each of the buckets found to get the k nearest neighbors of the query.

In the experiments presented in [12], t was arbitrarily set to 20. Our k-NN algorithm follows the same approach. The only difference is the way we calculate h.

We assume that GHS will be called only on a predefined set of k values. We attempt to estimate the number of buckets h_k that must be accessed to return an approximate result for a k-NN query. Given a collection $X \subseteq \mathcal{D}$, and a sample of objects $S \subseteq \mathcal{D} | S \cap X = \emptyset$, the algorithm of Figure 7 calculates statistics on the number of buckets that must be accessed in order to fulfill some EP goal tEP for a k-NN query. Specifically, we find the mean number of buckets accessed $\bar{x}_k$ and the corresponding standard deviation σ_k. In line 3, we loop through a set of sample objects. These sample objects must not exist in X, otherwise the EP value calculated in line 9 would always find an object with distance 0. In line 7, we iterate through all the sketches of X ordered by hamming distance to the sketch of s. In each iteration we search the objects embedded in the bucket and update the approximate result set S^A. When the current EP value cEP is lower or equal than the target EP value tEP (line 11), we can stop the loop and we record the number of buckets that had to be read in st. When the process is finished, we have accumulated statistics on the number of buckets that were read for the sample S and for a given k value. We store this information in the meta-data of the index. At query time, we can estimate the number of buckets that must be read to satisfy a k-NN query with the following expression:

$$h_k = \bar{x}_k + (\alpha \times \sigma_k)$$

The parameter α is provided by the user. A value of 3 should give a good estimation. Given h_k, the search procedure by Dong *et al.*(described above) can be applied.

```
$B$: set of sketches of $X$
$S \subseteq \mathcal{D}$: sample of objects
$X \subseteq \mathcal{D}$: database ($S \cap X = \emptyset$)
$k$: $k$-NN query that will be calculated
$tEP$: target $EP$ value.
Output: mean number of buckets accessed $\bar{x}_k$, standard deviation $\sigma_k$
1:  function $est(B, S, X, k, tEP)$
2:      $st$: statistics object
3:      for $s \in S$ do
4:          $OX$: objects of $X$ ordered by distance to $s$
5:          $OB$: sketches of $B$ ordered by hamming distance to the sketch of $s$
6:          $i = 0$
7:          for $b \in OB$ do
8:              Update $k$-NN result $S^A$ by searching $b$
9:              Calculate current $EP$ value $cEP$ with $OX$ and $S^A$
10:             $i++$
11:             if $cEP \leq tEP$ then
12:                 $st$.add($i$)                ▷ update stats
13:                 break;                ▷ Process the next $s$
14:             end if
15:         end for
16:     end for
    We can now obtain $\bar{x}_k$ and $\sigma_k$ from $st$.
17: end function
```

Figure 7. h_k estimation algorithm