JMLR: Workshop and Conference Proceedings 13: 145-160
2nd Asian Conference on Machine Learning (ACML2010), Tokyo, Japan, Nov. 8–10, 2010.

Single versus Multiple Sorting in All Pairs Similarity Search

Yasuo Tabei YASUO.TABEI@GMAIL.COM
Minato Discrete Structure Manipulation System Project, ERATO, Japan Science and Technology Agency, Sapporo, 060-0814, Japan

Takeaki Uno UNO@NII.JP
National Institute of Informatics, Tokyo, 101-8430, Japan

Masashi Sugiyama SUGI@CS.TITECH.AC.JP
Department of Computer Science, Tokyo Institute of Technology, Tokyo, 152-8552, Japan

Koji Tsuda KOJI.TSUDA@AIST.GO.JP
Computational Biology Research Center, National Institute of Advanced Industrial Science and Technology (AIST), Tokyo, 135-0064, Japan, and Minato Discrete Structure Manipulation System Project, ERATO, Japan Science and Technology Agency, Sapporo, 060-0814, Japan

Editor: Masashi Sugiyama and Qiang Yang

Abstract

To save memory and improve speed, vectorial data such as images and signals are often represented as strings of discrete symbols (i.e., sketches). Charikar (2002) proposed a fast approximate method for finding neighbor pairs of strings by sorting and scanning with a small window. This method, which we shall call "single sorting", is applied to locality sensitive codes and prevalently used in speed-demanding web-related applications. To improve on single sorting, we propose a novel method that employs blockwise masked sorting. Our method can dramatically reduce the number of candidate pairs which have to be verified by distance calculation in exchange with an increased amount of sorting operations. So it is especially attractive for high dimensional dense data, where distance calculation is expensive. Empirical results show the efficiency of our method in comparison to single sorting and recent fast nearest neighbor methods.

Keywords: SketchSort, Multiple sorting, Localty sensitive hashing, All pairs similarity search, Nearest neighbor graph

1. Introduction

Recently it is increasingly popular in machine learning and data mining that vectorial data such as images and signals are mapped to strings of discrete symbols (i.e., sketches). A main motivation of using sketches is to save memory and increase speed of subsequent learning algorithms. Locality sensitive hashing (LSH) employs random mapping to obtain bit or integer strings such that the distance in the original space is preserved as the Hamming distance among sketches (Gionis et al., 1999). There are several methods for different distances, Hamming (Gionis et al., 1999), cosine (Goemans and Williamson, 1995) and Euclidean (Datar et al., 2004). In view of machine learning, it is not always necessary to preserve global geometry. Several methods aim to design the mapping such that essential information is preserved for further inference. Examples are semantic hashing (Salakhut-

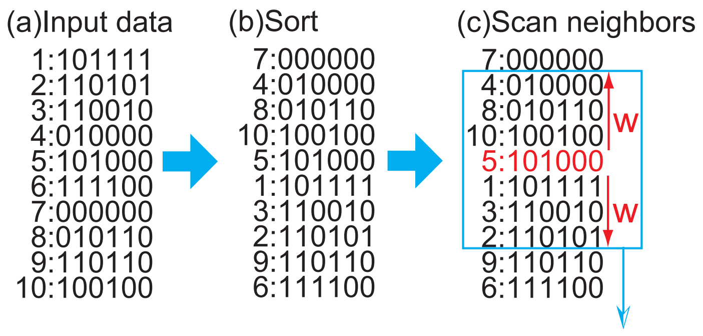

Figure 1: Single sorting method.

dinov and Hinton, 2007), spectral hashing (Weiss et al., 2009), kernelized LSH (Kulis and Grauman, 2009), locality sensitive binary codes for kernels (Raginsky and Lazebnik, 2010). In addition, the mapping can be learned from data (Shakhnarovich et al., 2003; Kulis et al., 2009; Kulis and Darrell, 2010)

In this paper, we deal with the problem of finding all neighbor pairs from sketches, i.e., *all pairs similarity search*. From such pairs, one can build a *neighborhood graph* which is a basis of many different tasks such as manifold modeling (Tennenbaum et al., 2000), semi-supervised learning (Zhou et al., 2004), spectral clustering (Hein et al., 2007), interesting region detection in images (G. Kim and A. Torralba, 2010), and retrieval of protein sequences (Weston et al., 2004). One can solve this problem by building an index from data, and give each point as a query to derive its neighbors (Beygelzimer et al., 2006). Nevertheless, it is often reported that all pairs similarity search methods (Bayardo et al., 2007; Ram et al., 2010) which find similar pairs without any index, are faster and consume less storage.

Charikar (2002) proposed a very simple yet effective method for finding neighbors by sorting (Figure 1). Sketches are lexicographically sorted, and the sorted table is scanned with a small window of height $2w+1$. For all pairs falling into the same window, the distance in the original space is calculated. If the distance is small enough, the pair is regarded as a valid neighbor. Neighbors sharing similar prefix can be found by this method, but some neighbors can be missed if mismatches lie in the beginning. To reduce the number of missing neighbor pairs, random permutation of letters is often introduced (Charikar, 2002). We shall call it "single sorting method" (SSM). The effectiveness of the method has been evidenced in Google news (Das et al., 2007), a computer-generated news site that aggregates news articles from more than 4,500 news sources worldwide. In its recommendation engine, the single sorting method is used to group similar stories together. Then, the story groups are displayed according to reader's personal interests.

A drawback of single sorting is that a large number of distance calculation is necessary for achieving reasonable accuracy. In addition, it is impossible to derive an analytic estimate of the fraction of missing neighbors, even if the probability of mismatch for each letter is given. It is because the probability of neighbors falling into the same window is data-dependent. This unfavorable property forces users to adjust the height w by trial-and-error for each dataset. To cope with the problems above, we propose a novel method called SketchSort that employs the *multiple sorting method* (MSM) (Uno, 2009) as a building block. MSM

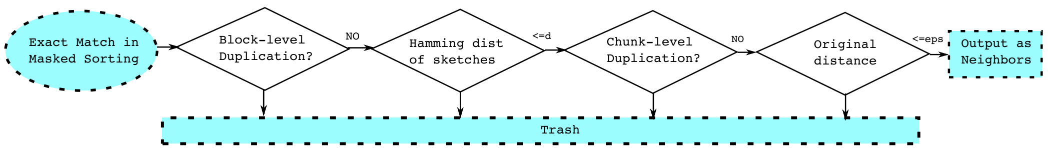

Figure 2: Global flow of our approach.

Table 1: Summary of recent all pairs similarity search methods.

	time comp.	type	detail	data
brute-force	$O(n^2)$	-	exact	real
cover tree (Beygelzimer et al., 2006)	$O(n \ln n)$	partition	exact, index	real
allknn (Ram et al., 2010)	$O(n)$	partition	exact, no index	real
Bayardo et al. (Bayardo et al., 2007)	$O(n)$	inverted index	exact, no index	binary
Lanczos bisection (Chen et al., 2009)	$O(n^t), 1 \leq t \leq 2$	partition	approx., no index	real
Single Sorting (Charikar, 2002)	$O(wn)$	sorting	approx., index/no index	real

has been proposed for detecting pairs of similar strings, but never been used or evaluated for real-valued feature vectors.

In MSM, sketches are divided into blocks. Some blocks are masked and sorting is done with respect to unmasked blocks. It allows us to detect neighbors of dissimilar prefixes as well. By trying all masking patterns, MSM offers a procedure to enumerate all pairs within Hamming distance d, which is beneficial in the following two aspects. 1) The number of distance calculation in the original space is significantly reduced by Hamming distance-based prefiltering and duplication detection (Figure 2). 2) A bound of the expected fraction of missing neighbors can be obtained in a data-independent manner.

As shown in Table 1, most of other state-of-the-art approaches employ “space partitioning” strategies, where the feature space is partitioned into several cells allowing overlap. In cover tree (Beygelzimer et al., 2006), a tree shaped index is constructed to find neighbors of a query quickly. Ram et al. (2010) proved that, based on the cover tree, exact all pairs k-nearest neighbor (NN) search can be solved in linear time. While this result is remarkable, it is often reported that the efficiency of tree-based methods is heavily data dependent (e.g.,Weiss et al. (2009)). This unstable behaviour is backed by the fact that the complexity is sharply dependent on the intrinsic dimensionality of data (Beygelzimer et al., 2006). Lanczos bisection (Chen et al., 2009) is an approximate k-NN search method where the space is partitioned into two halves recursively. If the number of points in each cell falls below a threshold, neighbors are found by brute-force computation. Though theoretical aspects of this method are not clear, high intrinsic dimensionality might pose a problem. Notice that, for extremely sparse binary data, other specialized approaches are possible (e.g.,Bayardo et al. (2007)) but they are not applicable to general data unfortunately.

In experiments, we first show, in near duplication detection experiments, that SketchSort is generally faster than single sorting at the same accuracy level. It is mainly due to the reduction of original distance calculations. Next, we compare SketchSort with recent space partitioning methods in near duplicate detection and k-NN search settings. SketchSort was faster in near duplicate detection by orders of magnitude. To our surprise, it performed competitively in k-nearest neighbor discovery as well.

This paper is organized as follows. Section 2 reviews the multiple sorting method for strings. The extension to real-valued data is explained in Section 3. A refinement procedure

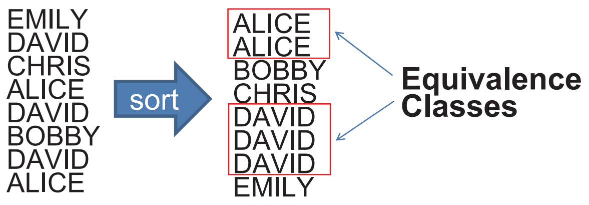

Figure 3: Sorting and equivalence classes.

of our method for finding k-nearest neighbor is presented in Section 4. Extensive empirical evaluations and comparisons are presented in Section 5. Finally concluding remarks are provided in Section 6.

2. Multiple Sorting Method

As a building block of our method, we need a fast method to detect near duplicates in short strings of equal length ℓ. The problem is formulated as follows: Given a string pool $S = \{s_1, \ldots, s_n\}$, find all neighbor pairs (i, j), $i < j$ whose Hamming distance is at most d, $HamDist(s_i, s_j) \leq d$. In the following, the number of neighbor pairs in data is described as m. Among several methods available (Abrahamson, 1987; Muthukrishnan and Sahinalp, 2000), we choose the multiple sorting method (MSM) (Uno, 2009) due to superior speed and memory efficiency.

2.1 Basic Idea

To explain the idea of MSM intuitively, let us start from the special case $d = 0$, that is, enumerating exactly same string pairs. In that case, the problem is solved by sorting the strings and scanning the sorted list to detect equivalence classes, each of which consists of more than 2 strings (Figure 3). Then, for each equivalence class, edges are built between all pairs. Using radix sort, sorting takes only $O(n)$ time. The edge building takes $O(m)$ time, where m is the number of all pairs within Hamming distance d. So the overall complexity is $O(n + m)$.

Even if $d > 0$, we can enumerate neighbor pairs by applying radix sort multiple times (Figure 4b). Let C denote a set of $\ell - d$ distinct integers taken from $\{1, \ldots, \ell\}$. Denote by s_i^C the i-th string whose characters at positions in C are concatenated. Thus, the positions not in C are masked. Obviously, the following two statements are equivalent.

- There exists C such that $s_i^C = s_j^C$, $|C| = \ell - d$.
- $HamDist(s_i, s_j) \leq d$.

Therefore, the neighbor pairs can be enumerated by trying every possible C of size d and sorting the masked strings. It takes $\begin{pmatrix} \ell \\ d \end{pmatrix}$ times sorting, hence the time complexity is polynomial to ℓ and exponential to d. Nevertheless, in terms of n and m, the time complexity stays linear, yielding overall complexity $O(n + m)$.

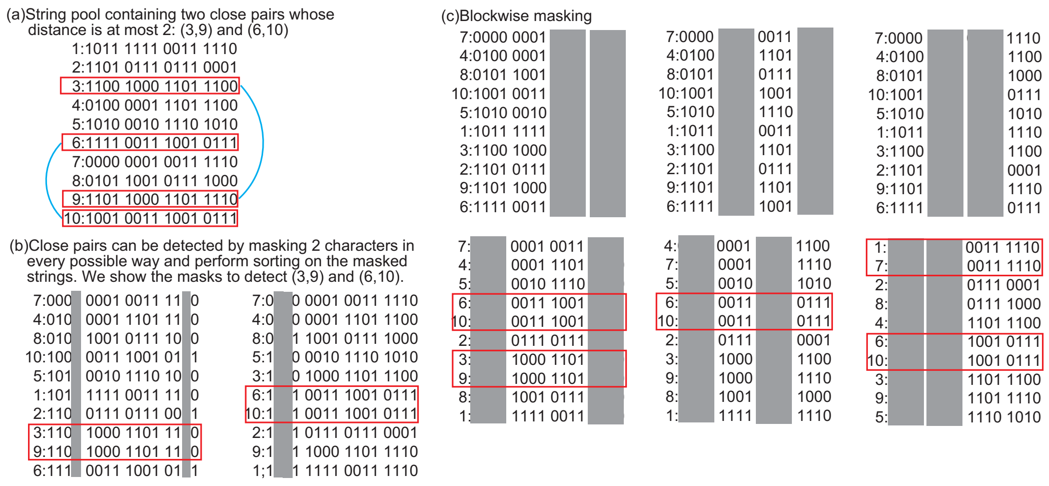
(a)String pool containing two close pairs whose distance is at most 2: (3,9) and (6,10)
1:1011 1111 0011 1110
2:1101 0111 0111 0001
3:1100 1000 1101 1100
4:0100 0001 1101 1100
5:1010 0010 1110 1010
6:1111 0011 1001 0111
7:0000 0001 0011 1110
8:0101 1001 0111 1000
9:1101 1000 1101 1110
10:1001 0011 1001 0111
(b)Close pairs can be detected by masking 2 characters in every possible way and perform sorting on the masked strings. We show the masks to detect (3,9) and (6,10).
(c)Blockwise masking

Figure 4: Multiple sorting method.

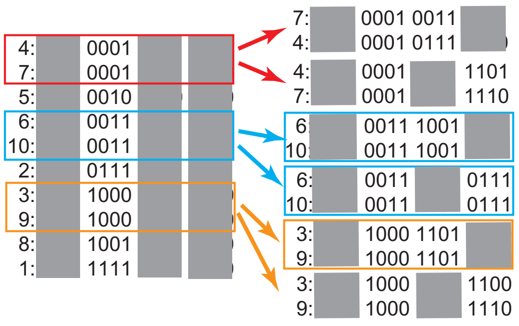

Figure 5: Updating equivalence classes in block concatenation. Strings in a block are sorted and equivalence classes (shown as square frames) are detected. A next block is concatenated to each equivalence class and sorted again.

2.2 Blockwise Masking

Empirical computation time of the above method is not optimal because a large number of sorting operations are necessary. To reduce the number of sorting operations, *blockwise masking* comes in useful.

Let us divide the strings into k blocks of approximately equal length as in Figure 4c. Define B as a set of $k-d$ distinct integers taken from $\{1,\ldots,k\}$. Denote by s_i^B the i-th string whose blocks listed in B are concatenated. The blocks not listed in B are masked. If $HamDist(s_i,s_j) \leq d$, then there exists B such that $s_i^B = s_j^B$. However, the inverse does not hold. When pairs are enumerated by trying every possible B and sorting the masked strings as before, the solution set contains all neighbor pairs as well as a certain number of non-neighbor pairs. To filter out non-neighbor pairs, we need to calculate the actual Hamming distances. Since distance calculation is done only for pairs falling into an equivalence class, the number of distance calculations is much smaller than exhaustive comparison.

2.3 Recursive Algorithm

Algorithm 1 Multiple Sorting Method. d: Hamming distance threshold, k: number of blocks.

```
function MultipleSortingMethod
    $I \leftarrow \{1, ..., n\}$
    $B \leftarrow \phi$
    Recursion($I$, $B$)
    return
end function
function Recursion($I$, $B$)
    if $|B| = k - d$ then
        for $(i, j) \in I \times I, i < j$ do
            if $s_i^b \neq s_j^b$ for all $b < \max(B), b \notin B$ then
                if $HamDist(s_i, s_j) \leq d$ then
                    Report $(i, j)$ to output file
                end if
            end if
        end for
        return
    end if
    for $b$ in $(\max(B) + 1)..(d + |B| + 1)$ do
        $J \leftarrow$ Sorted indices based on $b$-th block $\{s_i^b\}_{i \in I}$
        $T \leftarrow$ Intervals of equivalence classes in $\{s_j^b\}_{j \in J}$
        for each interval $(x, y) \in T$ do
            Recursion($J[x : y], B \cup b$)
        end for
    end for
    return
end function
```

In blockwise masking, one needs to detect equivalence classes in the concatenation of unmasked blocks. If $k = 4$ and $d = 2$, we need to traverse the following concatenation of blocks: (1,2), (1,3), (1,4), (2,3), (2,4), (3,4), and detect all equivalence classes in them. For efficient traversal, we adopt a recursive algorithm shown in Algorithm 1. First, it sorts strings in a block, and detects all equivalence classes in this block. The strings not included in any equivalence class are removed at this point. Then, sorting of the next block is done only for the remaining strings, yielding equivalence classes of concatenated blocks (Figure 5). The recursive structure of the algorithm allows us to avoid unnecessary computation. For example, for (1,2), (1,3), (1,4), the equivalence classes of the first block are reused.

For efficiency, it is very important not to duplicate the same pairs in output. To ensure that each pair is reported only once, we introduce *canonicity* in pairs of concatenated blocks. Let us define the set of all block combinations,

$$\mathcal{Z} = \{(i_1, \ldots, i_{k-d}) \mid 1 \leq i_1 < i_2 < \ldots < i_{k-d} \leq k\},$$

and introduce lexicographical order ('<') among elements of $\mathcal{Z}$. Consider two identical concatenated blocks s_i^B and s_j^B, $s_i^B = s_j^B$. If $s_i^Z \neq s_j^Z$ for all $Z < B$, $Z \in \mathcal{Z}$, this pair is called *canonical*. By reporting canonical pairs only, duplication can be avoided. Fortunately, it is not necessary to check all combinations due to the following property: the pair (s_i^B, s_j^B)

is canonical, iff $s_i^b \neq s_j^b$ for all $b < \max(B), b \notin B$. So canonicity check can be done by simply checking $\max(B) - |B|$ blocks in $O(\ell)$ time (line 10).

In radix sort at line 19, the number of executed operations is proportional to $c\ell/k$, where c is the number of strings of the equivalence class. We call it *sorting volume* of this radix sort. The total sorting volume in a whole run will be used as a measure of complexity later in experiments.

The worst-case complexity of blockwise sorting is worse than the complexity of letter masking $O(n+m)$, because the number of duplication checks can be larger than m. So the complexity of MSM is $O(n+m)$ which is achieved at $k = \ell$ (Uno, 2009).

3. SketchSort

To describe our method, we start from reviewing locality sensitive hashing. Then, the methodology to exploit MSM for real-valued vectors is explained, together with a bound on the fraction of missing neighbors.

3.1 Sparse Cosine LSH

In this section, we review exsting locality sensitive hashing (LSH) methods for nearest neighbor discovery. For efficiency, we employ the very sparse random projection proposed by Li et al. (2006) instead of the Gaussian-based dense matrix commonly used in literature (Gionis et al., 1999). Denote n data points in $\Re^D$ by $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$. The cosine distance is defined as :

$$\Delta(\boldsymbol{x}_i, \boldsymbol{x}_j) \quad = \quad 1 - \frac{\boldsymbol{x}_i^\top \boldsymbol{x}_j}{||\boldsymbol{x}_i||||\boldsymbol{x}_j||}. \tag{1}$$

Let $R \in \Re^{D \times \ell}$ be a random matrix defined as follows:

$$r_{ij} \quad = \quad \sqrt{t} \begin{cases} 1 \text{ with prob. } \frac{1}{2t}, \\ 0 \text{ with prob. } 1 - \frac{1}{t}, \\ -1 \text{ with prob. } \frac{1}{2t}, \end{cases} \tag{2}$$

where $t = \sqrt{D}$. Let $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n$ be bit strings of length ℓ. The projection is defined as

$$s_{ik} = \text{sign}(\boldsymbol{r}_k^\top \boldsymbol{x}_i), \tag{3}$$

where s_{ik} is the k-th letter of the i-th string, $\boldsymbol{r}_k$ is the k-th column of R and sign(t) produces 1 if $t > 0$ and 0 otherwise. The cosine distance is approximately preserved as the Hamming distance, because of the following relationship:

$$Pr[s_{ik} \neq s_{jk}] = \frac{\theta_{ij}}{\pi}, \ \forall k, \tag{4}$$

where θ_{ij} is the angle between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$:

$$\theta_{ij} = \arccos\left(\frac{\boldsymbol{x}_i^\top \boldsymbol{x}_j}{||\boldsymbol{x}_i||||\boldsymbol{x}_j||}\right). \tag{5}$$

This relationship guarantees that the expected value of the Hamming distance is a monotonically increasing function of the cosine distance. Furthermore, in the limit $\ell \to \infty$, the Hamming distance between two bit strings converges to the angle of the original vectors.

$$\lim_{\ell \to \infty} \frac{1}{\ell} HamDist(s_i, s_j) = \frac{\theta_{ij}}{\pi}. \tag{6}$$

When ℓ is finite, $HamDist(s_i, s_j)$ is subject to the binomial distribution Binom(ℓ, $\frac{\theta_{ij}}{\pi}$).

3.2 Method

Our basic idea is to map the data points to strings by LSH, and enumerate pairs of similar strings by MSM. However, considering the fact that the computational complexity of MSM is a polynomial function of the string length, it is not a good strategy to create long strings and process them by MSM at once. Thus, we employ the following chunking strategy: First, long sketches of $Q\ell$ bits are created from data points by LSH. Then, it is divided into Q chunks of short strings of length ℓ. Denote by $\boldsymbol{s}_{qi}$ the q-th string corresponding to $\boldsymbol{x}_i$. Basically, we would like to create the union of all MSM results, $E = \bigcup_{q=1}^{Q} P_q$, where P_q is the set of MSM pairs of chunk q. Then, we finally report the pairs in E whose cosine distance is small.

In straightforward implementation, MSM is called for each chunk separately, the derived pairs are stored in memory and integrated into E. However, it is not optimal due to $O(n^2)$ memory requirement. Instead, we take linear-memory implementation shown in Algorithm 2. Here, MSM is repeated Q times for each chunk and two additional checks are performed to remove unnecessary pairs. The first check (line 15) resolves *chunk-level duplication*: When a pair (i, j) is found at the q-th chunk, the Hamming distances of chunks $r = 1, \ldots, q-1$ are calculated. The pair survives if all the distances are beyond the threshold d. In the final check (line 16), the actual cosine distance of original vectors is calculated, and qualifying pairs are reported to an output file.

3.3 Missing Edge Ratio

Given the true set of neighbors E^*

$$E^* = \{(i, j) \mid \Delta(\boldsymbol{x}_i, \boldsymbol{x}_j) \leq \epsilon\},$$

and the union of all MSM results E, there are two kinds of error.

- Type-I error (false positive): A non-neighbor pair has a Hamming distance within d in at least one chunk.

$$F_1 = \{(i, j) \mid (i, j) \in E, (i, j) \notin E^*\}.$$

- Type-II error (false negative): A neighbor pair has a Hamming distance larger than d in all chunks.

$$F_2 = \{(i, j) \mid (i, j) \notin E, (i, j) \in E^*\}.$$

Algorithm 2 SketchSort for near duplicate detection. Q: number of chunks, d: Hamming distance threshold, k: number of blocks, ϵ: cosine distance threshold.

```
1: function SketchSort($\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$)
2:     Use LSH to obtain sketches $\{s_{1i}, \ldots, s_{qi}\}_{i=1}^n$ from data $\{\boldsymbol{x}_i\}_{i=1}^n$
3:     $I \leftarrow \{1, ..., n\}$
4:     for $q = 1 : Q$ do
5:         $B \leftarrow \phi$
6:         Recursion($I$, $B$, q)
7:     end for
8:     return
9: end function
10: function Recursion($I$, $B$, q)
11:     if $|B| = k - d$ then
12:         for $(i, j) \in I \times I, i < j$ do
13:             if $s_{qi}^b \neq s_{qj}^b$ for all $b < \max(B), b \notin B$ then
14:                 if $HamDist(s_{qi}, s_{qj}) \leq d$ then
15:                     if $HamDist(s_{ri}, s_{rj}) > d$ for all $r < q$ then
16:                         if $\Delta(\boldsymbol{x}_i, \boldsymbol{x}_j) \leq \epsilon$ then
17:                             Report $(i, j)$ to output file
18:                         end if
19:                     end if
20:                 end if
21:             end if
22:         end for
23:         return
24:     end if
25:     for $b$ in $(\max(B) + 1)..(d + |B| + 1)$ do
26:         $J \leftarrow$ Sorted indices based on $b$-th block $\{s_{qi}^b\}_{i \in I}$
27:         $T \leftarrow$ Intervals of equivalence classes in $\{s_{qj}^b\}_{j \in J}$
28:         for each interval $(x, y) \in T$ do
29:             Recursion($J[x : y], B \cup b$, q)
30:         end for
31:     end for
32:     return
33: end function
```

The type-II errors are more critical in our method because the type-I errors are eventually filtered out by calculating the cosine distances. The fraction of missing neighbors is defined as $|F_2|/|E^*|$, whose expectation is bounded as follows.

$$E\left[\frac{|F_2|}{|E^*|}\right] \leq \left(1 - \sum_{k=0}^{\lfloor d \rfloor} \binom{\ell}{k} p^k (1-p)^{\ell-k}\right)^Q, \tag{7}$$

where p is an upper bound of the non-collision probability (4) for neighbors. For the cosine LSH, p is set as follows,

$$p = \frac{\arccos(1-\epsilon)}{\pi}.$$

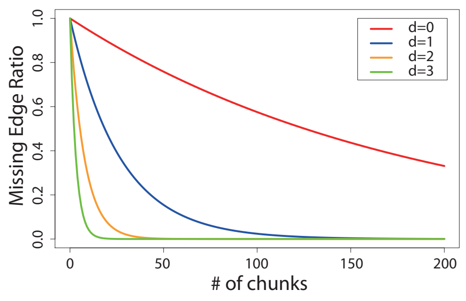

Figure 6: Bound of expected missing edge ratio against the number of chunks.

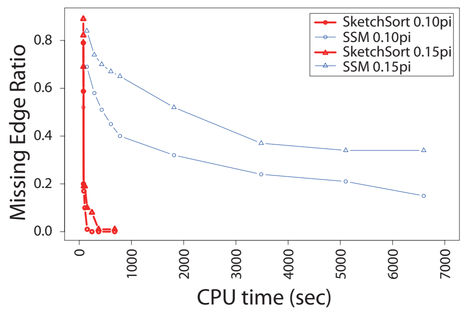

Figure 7: Comparison of SketchSort and single sorting on single 32 bit sketch.

We call this value *missing edge ratio*, because it amounts to the fraction of missing edges in the resultant neighborhood graph.

Figure 6 depicts the missing edge ratio as a function of Q for different values of d. We used the cosine LSH where the radius is set such that $p = 0.15$ and $\ell = 32$. As d increases, the number of required chunks reduces remarkably. To improve missing edge ratio, there are two solutions, (i) increase the number of chunks Q, (ii) increase the Hamming distance threshold d. The former has weaker effect, but the complexity grows only linearly. The latter leads to exponential growth in MSM's computational time, but has much stronger effect.

3.4 Complexity of SketchSort

The complexity of SketchSort is $O(n+m')$, where m' is the number of pairs whose Hamming distance is at most d in at least one chunk,

$$m' = \{(i,j) \mid \min_q(HamDist(s_{qi}, s_{qj})) \le d\}.$$

It is derived from the fact that MSM's complexity is $O(n+m)$. Memory requirement is $O(n)$ which makes SketchSort particularly attractive for large datasets.

In general, the worst case analysis of all pairs similarity search is tricky, because if all the entries are identical, we need $O(n^2)$ time simply to output all pairs. To avoid this vacuous bound, we need assumptions on data such as intrinsic dimensionality (Beygelzimer et al., 2006). The linear time result in (Ram et al., 2010) assumes that intrinsic dimensionality stays constant as n grows, which might not be true in reality.

4. K-Nearest Neighbor Search

In the last section, we presented SketchSort for near duplicate detection (i.e., finding all pairs within distance ϵ). To make SketchSort applicable to k-NN pairs search, the following modification is done.

Instead of cutting off at the threshold at line 16, we maintain the best-h neighbors of each point as a linked list. If the distance of a new pair is smaller than the current h-best pair, it is replaced by the new one. After all computation is done, a neighborhood

Table 2: SketchSort and single sorting (SSM) on MNIST dataset on single 32-bit sketch.

SketchSort d	MER (0.10π)	MER (0.15π)	sorting volume	# of HamDist	# of cosdist	time (sec)
1	0.79	0.89	3,897,528	244,050	114,830	78.29
2	0.52	0.81	7,053,642	892,193	236,551	79.22
3	0.20	0.69	10,675,856	4,199,689	615,035	81.00
4	0.17	0.19	14,699,954	18,266,846	3,593,240	88.00
5	0.10	0.19	19,920,000	42,609,173	3,693,937	107.88
6	0.01	0.10	24,660,000	110,253,094	7,963,091	152.61
7	0.01	0.08	30,540,000	224,195,158	16,768,097	243.54
8	0.00	0.01	36,720,000	368,724,926	27,866,624	370.95
9	0.00	0.01	44,580,000	611,893,757	55,695,612	679.80

SSM w	MER (0.10π)	MER (0.15π)	sorting volume	# of cosdist	time (sec)
100	0.69	0.84	19,200,000	11,983,539	145.81
300	0.58	0.74	19,200,000	35,901,099	289.21
500	0.51	0.70	19,200,000	59,739,467	430.72
750	0.45	0.67	19,200,000	89,425,328	605.90
1000	0.40	0.65	19,200,000	118,986,462	780.62
2500	0.32	0.52	19,200,000	293,733,201	1,813.78
5000	0.24	0.37	19,200,000	574,979,323	3,484.08
7500	0.21	0.34	19,200,000	843,726,162	5,102.95
10000	0.15	0.29	19,200,000	1,099,972,420	6,595.80

graph is constructed, where each node has h adjacent nodes. For each node, we seek k neighbors in terms of cosine distance from the surrounding nodes within graph distance 2. It requires at most $O(h^2)$ computation of cosine distances per node, but increases the accuracy significantly. Notice that this refinement procedure is employed in (Chen et al., 2009) as well.

5. Experiments

In our experiments, we used two image datasets. One is MNIST handwritten digit recognition dataset of 60000 data and 748 dimensions (LeCun and Cortes, 2000). The other is the tiny image dataset collected by Torralba et al. (2008), which will be referred to as TinyImage later on. TinyImage has 80 million images in total, but we used a smaller version containing 1.6 million images, which was immediately downloadable[1]. Using GIST descriptors (Douze et al., 2009), a 960 dimensional feature vector was made for each image. While our method is based on the cosine distance, several methods used in our experiments are based on the Euclidean distance. For a fair comparison, features are centralized and normalized to norm 1. All experiments are performed on a linux machine on an AMD Opteron Processor^{TM} 854 2.8GHz with 64GB memory.

5.1 Near Duplication Detection

Here, MSM is compared to SSM (Charikar, 2002; Ravichandran et al., 2005) with respect to missing edge ratio (MER) and execution time in near duplication detection. We tried two different values of cosine distance radius $\epsilon = 0.0489$ and 0.109 which translate to 0.10π and 0.15π in terms of angle, respectively. The results for one chunk of 32 bits are presented

1. `http://people.csail.mit.edu/torralba/tinyimages/`

in Table 2. Here, the number of blocks k are always set to $d + 3$. As shown in comparative plots in Figure 7, SketchSort is faster by orders of magnitude at most levels of missing edge ratio. This is mainly due to the reduced number of cosine distance calculations. The sorting volume is constant for SSM and variable in MSM. For large d, the sorting volume of MSM exceeds SSM, but it did not have large impact on the computational time.

We also conducted similar experiments using multiple chunks on the two datasets, MNIST and TinyImage. TinyImage is downsampled to 100,000 points, because the ground-truth result have to be obtained by brute-force computation. For SketchSort, the parameters (d, k) is fixed either to $(2, 5)$ or $(3, 6)$, and the number of chunks Q is varied as $2, 6, 10, \ldots, 50$. For SSM, the window height w is fixed to 50 or 100 and the number of chunks is varied in the same manner. For comparison, an engineered version of Lanczos bisection Chen et al. (2009) is included. In the original version, Lanczos bisection detects k-nearest neighbors at the end of recursive bisection. Here we modified the source code[2], so that near duplicates are detected instead. Lanczos bisection has two types, Lanczos-glue and Lanczos-overlap. They are different with respect to the way of dividing regions (see Chen et al., 2009). Both types have a parameter p to control the maximum number of data points for brute-force search, which is varied as $10, 100, 200, \ldots, 1000$. The results for MNIST and TinyImage datasets are shown in Figure 8. Our method was significantly faster than SSM and Lanczos bisection methods at the same level of missing edge ratio. In TinyImage dataset, the difference is clearer, due to the larger number of data points and higher dimensionality.

5.2 *k*-Nearest Neighbor Search

We evaluated SketchSort's empirical performance on the k-nearest neighbor search task. Our method is extended as stated in Section 4, setting h=25, and compared with Lanczos bisection (Chen et al., 2009), cover tree (Beygelzimer et al., 2006) and allknn (Ram et al., 2010). For cover tree and allknn, we used the source codes from `http://hunch.net/~jl/projects/cover_tree/cover_tree_2.tar.gz` and `http://mloss.org/software/view/152/`, respectively. The error rate of a k-nearest neighbor graph G' against the true graph G is defined as

$$\text{error-rate}(G') = 1 - \frac{|E(G') \cap E(G)|}{|E(G)|},$$

where $E(\cdot)$ denotes the set of edges in the graph.

Figure 9 shows that our method is significantly faster than the exact methods, cover tree and allknn. In MNIST, it showed speed-accuracy trade-off comparable to Lanczos bisection. In TinyImage, SketchSort was faster by substantial margin. Space-partitioning methods are considered as good at finding nearest neighbors which lie far away from each other (Chen et al., 2009), whereas SketchSort is designed for finding close neighbors in principle. Nevertheless, this result shows that SketchSort can be a viable alternative in k-nearest neighbor search as well.

5.3 Near Duplicate Detection in 1.6 Million Images

To demonstrate SketchSort's scalability, we conducted near duplicate detection experiments using the full set of TinyImage at three angle thresholds 0.05π, 0.10π and 0.15π which cor-

2. `http://www-users.cs.umn.edu/~saad/software/knn.tar.gz`

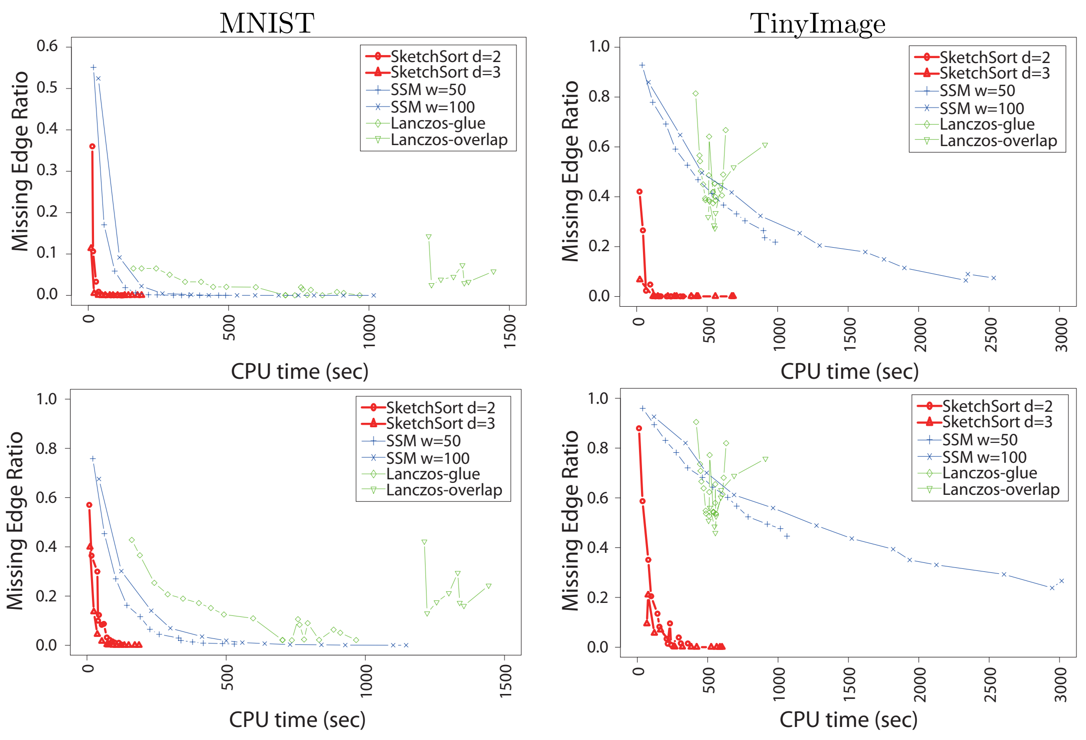

Figure 8: Near duplicate detection on MNIST and TinyImage datasets for cosine distance thresholds 0.10π (top) and 0.15π (bottom).

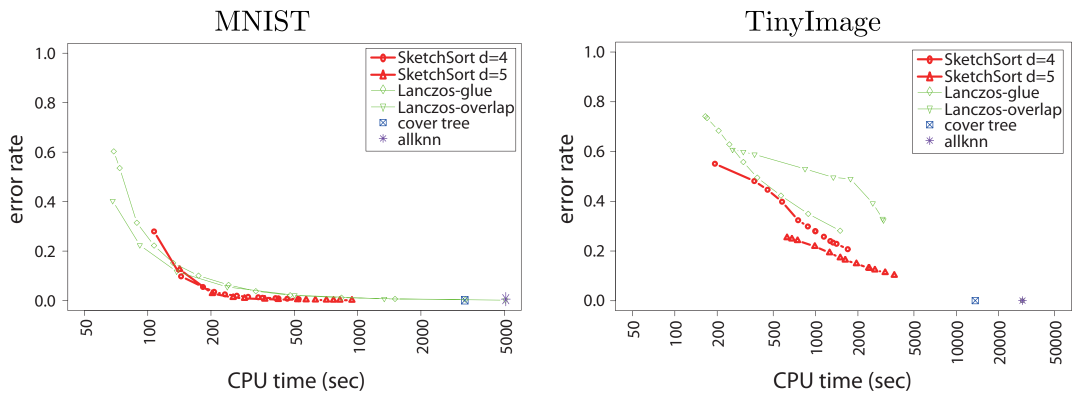

Figure 9: Error rate for 5-nearest neighbor search on MNIST and TinyImage datasets.

respond to cosine distance radiuses 0.0123, 0.0489 and 0.109, respectively. The parameters (d, k) are fixed to (2, 6), and Q is set such that the missing edge ratio bound is approximately 1.0×10^{-6} to make error rate negligibly small. To observe the growth rate, we also generated smaller datasets of different sizes by talking the first n records. Figure 10 shows the efficiency results in comparison with exact methods, cover tree and brute force. Within

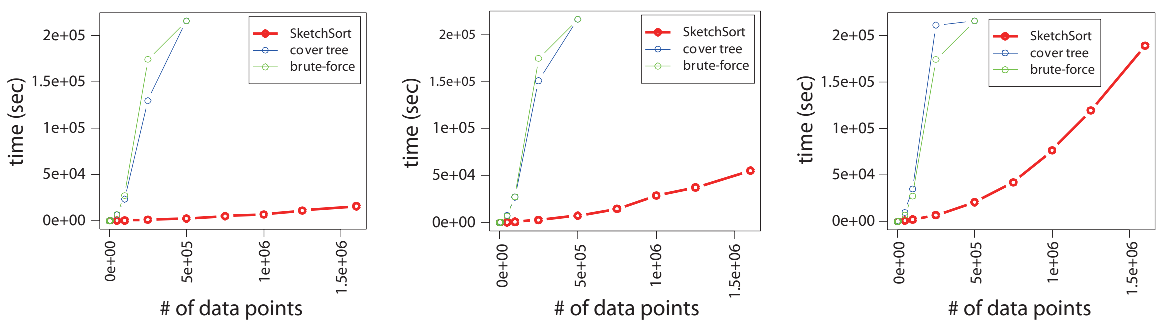

Figure 10: Near duplicate detection in up to 1.6 million images at thresholds 0.05π (left), 0.10π (middle) and 0.15π (right).

the time limit of 60 hours, cover tree could scale only up to 500,000 samples even at the smallest threshold 0.05π. On the other hand, SketchSort processed the full set in 4.3 hours.

6. Conclusions

In this paper, we proposed a novel algorithm for all pairs similarity search. Single sorting is used in diverse applications probably due to its simplicity. However, we have shown that a more deliberate algorithm can achieve significant improvement in speed. In experiments, we solely used the locality sensitive hashing, but our method is applicable to other non-random sketches such as spectral hashing (Weiss et al., 2009) and semantic hashing (Salakhutdinov and Hinton, 2007).

In future work, we would like to explore the following questions. 1) Is it really necessary to map the vectors to discrete symbols? Apart from locality sensitive hashing, there are a variety of random projection algorithms that maps a vector to a real-value (Li et al., 2006). Can they be used for better accuracy? Sorting continuous values takes more time than radix sort, but it is possible in $O(n \log n)$ time. 2) Hierarchical organization of the multiple sorting method. Our current algorithm has a two-level structure that Q chunks are created, each of which is divided into k blocks. We are not yet sure if it is the optimal architecture. Further efficiency could possibly be achieved by organizing the algorithm in a hierarchy of more than two levels. 3) Implementation of SketchSort in a many-core processor. Though we performed all sorting operations serially, our method is inherently amenable to parallelization.

7. Acknowledgments

This work is partly supported by MEXT Kakenhi 21680025, the research fellowship from the Japan Society for the Promotion of Science (JSPS) for Young Scientists, the FIRST program, and Synthesis of Knowledge for Information Oriented Society of the PRESTO program (JST).

References

K. Abrahamson. Generalized string matching. *SIAM Journal on Computing*, 16:1039–1051, 1987.

J.R. Bayardo, Y. Ma, and R. Srikant. Scaling Up All Pairs Similarity Search. In *Proceedings of the 16th International Conference on World Wide Web*, 2007.

A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *International Conference on Machine Learning*, pages 97–104, 2006.

Moses Charikar. Similarity Estimation Techniques from Rounding Algorithms. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, 2002.

J. Chen, H. Fang, and Y. Saad. Fast Approximate kNN Graph Construction for High Dimensional Data via Recursive Lanczos Bisection. *Journal of Machine Learning Research*, 10:1989–2012, 2009.

A.S. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th International Conference on World Wide Web*, page 280. ACM, 2007.

M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni. Locality sensitive hashing scheme based on p-stable distributions. In *Proceedings of the ACM Symposium on Computational Geometry*, 2004.

M. Douze, H. Jégou, H. Sandhawalia, L. Amsaleg, and C. Schmid. Evaluation of GIST descriptors for web-scale image search. In *ACM International Conference on Image and Video Retrieval*, 2009.

G. Kim and A. Torralba. Unsupervised Detection of Regions of Interest Using Iterative Link Analysis. In *Advances in Neural Information Processing Systems*, 2010.

A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Base*, 1999.

M.X. Goemans and D.P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42 (6):1115–1145, 1995.

M. Hein, J.-Y. Audibert, and U. von Luxburg. Graph Laplacians and their convergence on random neighborhood graphs. *Journal of Machine Learning Research*, 8:1325–1368, 2007.

B. Kulis and T. Darrell. Learning to Hash with Binary Reconstructive Embeddings. In *Advances in Neural Information Processing Systems*, 2010.

B. Kulis and K. Grauman. Kernelized Locality-Sensitive Hashing for Scalable Image Search. In *International Conference on Computer Vision*, 2009.

B. Kulis, P. Jain, and K. Grauman. Fast Similarity Search for Learned Metrics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(12):2143, 2009.

Y. LeCun and C. Cortes. MNIST database. *http://yann.lecun.com/exdb/mnist/*, 2000.

P. Li, T.J. Hastie, and K.W. Church. Very Sparse Random Projections. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, page 296. ACM, 2006.

S. Muthukrishnan and S.C. Sahinalp. Approximate nearest neighbors and sequence comparison with block operations. In *Proceedings of 32nd annual ACM Symposium on Theory of Computing*, pages 416–424, 2000.

M. Raginsky and S. Lazebnik. Locality-Sensitive Binary Codes from Shift-Invariant Kernels. In *Advances in Neural Information Processing Systems*, 2010.

P. Ram, D. Lee, W. March, and A. Gray. Linear-time Algorithms for Pairwise Statistical Problems. In *Advances in Neural Information Processing Systems*, 2010.

D. Ravichandran, P. Pantel, and E. Hovy. Randomized Algorithms and NLP: Using Locality Sensitive Hash Function for High Speed Noun Clustering. In *Annual Meeting of the Association for Computational Linguistics*, pages 345–356, 2005.

R.R. Salakhutdinov and G.E. Hinton. Semantic hashing. In *SIGIR Workshop on Information Retrieval and Applications of Graphical Models*, 2007.

G. Shakhnarovich, P. Viola, and T. Darrell. Fast pose estimation with parameter-sensitive hashing. In *Proceedings of the Ninth IEEE International Conference on Computer Vision*, page 750, 2003.

J.B. Tennenbaum, V. de Silva, and J.C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290:2319–2323, 2000.

A. Torralba, R. Fergus, and W.T. Freeman. 80 million tiny images; a large dataset for non-parametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(11):1958–1970, 2008.

T. Uno. Multi-sorting algorithm for finding pairs of similar short substrings from large-scale string data. *Knowledge and Information Systems*, 2009. published online.

Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *Advances in Neural Information Processing Systems*, 2009.

J. Weston, A. Elisseeff, D. Zhou, C. Leslie, and W.S. Noble. Protein ranking: from local to global structure in the protein similarity network. *Proceedings of the National Academy of Sciences of USA*, 101(17):6559–6563, 2004.

D. Zhou, O. Bousquet, J. Weston, and B. Schölkopf. Learning with local and global consistency. In *Advances in Neural Information Processing Systems*, pages 321–328. MIT Press, 2004.