Nearest-neighbor Search from Large Datasets using Narrow Sketches

Naoya Higuchi[1], Yasunobu Imamura[2], Vladimir Mic[3], Takeshi Shinohara[4], Kouichi Hirata[4] and Tetsuji Kuboyama[5]

[1]*Sojo University, Ikeda 4-22-1, Kumamoto 860-0082, Japan*

[2]*THIRD INC., Shinjuku, Tokyo 160-0004, Japan*

[3]*Masaryk University, Brno, Czech Republic*

[4]*Kyushu Institute of Technology, Kawazu 680-4, Iizuka 820-8502, Japan*

[5]*Gakushuin University, Mejiro 1-5-1, Toshima, Tokyo 171-8588, Japan*

Keywords: Narrow Sketch, Nearest-neighbor Search, Large Dataset, Sketch Enumeration, Partially Restored Distance.

Abstract: We consider the nearest-neighbor search on large-scale high-dimensional datasets that cannot fit in the main memory. Sketches are bit strings that compactly express data points. Although it is usually thought that wide sketches are needed for high-precision searches, we use relatively narrow sketches such as 22-bit or 24-bit, to select a small set of candidates for the search. We use an asymmetric distance between data points and sketches as the criteria for candidate selection, instead of traditionally used Hamming distance. It can be considered a distance partially restoring quantization error. We utilize an efficient one-by-one sketch enumeration in the order of the partially restored distance to realize a fast candidate selection. We use two datasets to demonstrate the effectiveness of the method: YFCC100M-HNfc6 consisting of about 100 million 4,096 dimensional image descriptors and DEEP1B consisting of 1 billion 96 dimensional vectors. Using a standard desktop computer, we conducted a nearest-neighbor search for a query on datasets stored on SSD, where vectors are represented by 8-bit integers. The proposed method executes the search in 5.8 seconds for the 400GB dataset YFCC100M, and 0.24 seconds for the 100GB dataset DEEP1B, while keeping the recall of 90%.

1 INTRODUCTION

We discuss the realization of a fast and accurate nearest-neighbor search (NN search, for short) from large-scale datasets. *Sketches* are bit strings compactly representing high-dimensional data (Wang et al., 2007; Dong et al., 2008; Müller and Shinohara, 2009; Mic et al., 2015; Mic et al., 2016). The mapping of a data point to sketch is considered as a kind of dimension reduction (Higuchi et al., 2018). We have succeeded in effectively searching for the nearest neighbor of a query from high-dimensional datasets that can be read into the main memory in advance (Higuchi et al., 2019a; Higuchi et al., 2019b). We use a filtering algorithm sped up by enumerating narrow sketches, such as 16-bit. In this paper, we show that their search method using narrow sketches also provides an efficient search even for large datasets that do not fit in the main memory.

In this section, we give the outline of our previous study and the contribution of this paper.

1.1 Our Previous Study

Traditionally, the NN search using sketches has been utilizing the Hamming distance between sketches to select the candidates for the nearest neighbor. We use sketches based on the ball partitioning (BP, for short), where each bit of sketch is determined by whether the data point is inside a hypersphere or not. Since BP-based sketches are regarded as quantized images of a dimension reduction such as Simple-Map (Shinohara and Ishizaka, 2002), there is a quantization error in the distance between sketches. We proposed the *partially restored distance* between data points and sketches that provides a more accurate candidate selection than the Hamming distance (Higuchi et al., 2018). Similar distances are also introduced as the asymmetric distance in sketch (Dong et al., 2008) and product quantization (Matsui et al., 2018).

We call a pair (c, r) of a data point c and a real number r a *pivot* to specify the center and the radius of the hypersphere. Clearly, the pivot selection

Higuchi, N., Imamura, Y., Mic, V., Shinohara, T., Hirata, K. and Kuboyama, T.
Nearest-neighbor Search from Large Datasets using Narrow Sketches.
DOI: 10.5220/0010817600003122
In *Proceedings of the 11th International Conference on Pattern Recognition Applications and Methods (ICPRAM 2022)*, pages 401-410
ISBN: 978-989-758-549-4; ISSN: 2184-4313

for sketching is one of the keys to achieving precise filtering. Intuitively, the higher the information preserved by sketches, the more precise the search using sketches. Therefore, we first tried to find a pivot set by which the collision probability is low. This is almost equivalent to the approach by the method of maximizing entropy. We applied a local search method for pivot selection to minimize collision probability. We have experienced the fact that many of the pivots found by the local search lie in the corner of the space. From this, we proposed the method of binary quantization ball partitioning, QBP for short, which restricts the centers of the pivots to corners of the data space. To break the limits of the local search we introduced a method named AIR (Annealing by Increasing Resampling) (Imamura et al., 2017; Higuchi et al., 2020a). Although we can find a set of pivots to provide sketches with a small collision probability by AIR, the precision of the search could not be improved. Recently, we proposed a pivot selection directly using the search precision itself as the objective function of AIR instead of the collision probability (Higuchi et al., 2020b), by which we can find a pivot set to realize efficient sketching transformation.

Furthermore, we succeeded in effectively searching for the nearest neighbor of a query from high-dimensional datasets that can be read into the main memory in advance (Higuchi et al., 2019a; Higuchi et al., 2019b). We use a filtering algorithm sped up by enumerating narrow sketches, such as 16-bit, one-by-one in ascending order of partially restored distance.

1.2 Contribution of this Paper

In this paper, we show that the search method using narrow sketches also provides an efficient search even for large datasets that do not fit in the main memory.

There are not many large-scale datasets available to the public, while we use two datasets: YFCC100M-HNfc6 (YFCC100M, for short) (Amato et al., 2016a) and DEEP1B (Babenko and Lempitsky, 2016). DEEP1B consists of 1 billion 96 dimensional vectors. Although the number of vectors in YFCC100M is about 100 million and not so large as DEEP1B, the dimensionality is high at 4,096.

In our experiments, vectors are represented by 8-bit integers. The sizes of YFCC100M and DEEP1B are about 400GB and 100GB, respectively. We use a common desktop computer with a standard 8-core processor and M.2 SSD. A naïve NN search by sequential read requires about 300 seconds and 85 seconds for YFCC100M and DEEP1B, respectively, on average for a query. This speed is reasonable from the fact that we use the SSD with a sequential read speed of 2GB per second.

We consider a NN search divided into two stages. At the first stage, the dataset is filtered to select solution candidates using narrow sketches. At the second stage, the nearest neighbor is selected from the candidates. The precision of the search is measured by the *recall*, that is, the ratio that the correct answer is retrieved. The average search time for a query with the recall of 90% is 11 seconds for YFCC100M using 24-bit sketches, and 0.77 seconds for DEEP1B using 22-bit sketches, respectively. If parallel processing with 8 threads is used, it can be shortened to 5.8 seconds and 0.24 seconds, respectively.

The search method used in this paper is based on the fast filtering by sketch enumeration (Higuchi et al., 2019a; Higuchi et al., 2019b). The contribution of this paper is to show practical findings to achieve a high-speed and high-precision search even for a large dataset that cannot be stored in the main memory, and to bridge the gap between the existing method and its practical use. Many useful insights have been gained from the experiments counducted in this paper. In fact, the experimental results imply that the proposed method with narrower sketches outperforms the methods with wider sketches or other filtering methods employed in most NN searches.

2 NEAREST-NEIGHBOR SEARCH USING NARROW SKETCHES

In this paper, we discuss an efficient NN search in a metric space. When the dimensionality of data space is high and the number of the searched data points is large, it is necessary to introduce dimension reduction with an indexing method to be released from so-called "the curses of dimensionality." The *sketches* are bit strings representing data points and can be directly used as indexes of the bucket method. The *width* of a sketch is the length of the bit string. A mapping of a data point to the sketch, which we call *sketching*, is considered as a kind of dimension reduction as long as the width of sketches is narrow. Thus, the sketch-based method we propose provides both efficient indexing and dimension reduction.

The NN search using sketches is performed in two stages. The first stage narrows down the candidates of solution based on the *priorities* of data points defined by sketches, and then, the second stage selects the nearest neighbor from the candidates. We call the first stage *filtering by sketches*. Traditionally, the priority in filtering is defined by the Hamming distance between sketches. The proposed method in this paper uses an asymmetric partially restored distance $\bar{D}_p$ be-

tween data points and sketches to define the priority of the candidate data points (Higuchi et al., 2018). The $\tilde{D}_p(q,\varsigma)$ is defined as the L_p like aggregation of distance lower bounds between q and any x whose sketch is ς. This approach provides more reliable filtering by sketches.

The smaller the number of candidates selected, and the more reliable prioritization in filtering, the faster the search in the second stage we expect. A simple way to reduce the candidate set without degrading filtering performance is to use wide sketches. On the other hand, the wider the sketches in filtering, the higher the cost for filtering. Furthermore, accessing a few data points scattered in the secondary storage becomes slow due to random access. Thus, only simply reducing the number of candidates cannot achieve a fast search.

To increase the speed of the entire search, it is important to balance the number of candidates and the data access efficiency of the second stage search. We showed that the NN search with the narrow 16-bit sketches is faster than that with the wide 32-bit or more sketches (Higuchi et al., 2019b). One of the key methods to realize a fast search is the efficient enumeration of narrow sketches in the order of priorities, by which we can very quickly collect the candidates in filtering.

In (Higuchi et al., 2019a; Higuchi et al., 2019b), the search has been performed *on-memory* with all the data loaded in the main memory. In this paper, we show that the search using narrow sketches is effective even when the data is too large to fit in the main memory.

In the rest of this section, we give a brief introduction to the NN search using sketches. For details, refer to the literature (Higuchi et al., 2019a; Higuchi et al., 2020b).

2.1 Nearest-neighbor in a Metric Space

Let $\mathcal{U}$ be the space of data points to be searched. We assume $\mathcal{U}$ is a metric space where the distance function D is defined. Given a positive integer k and a data point q, a *k-nearest-neighbor search* (*k-NN search* for short) for q from a dataset $ds \subseteq \mathcal{U}$ is the search for the k closest points in ds to q. In this paper, we mainly concern with the NN search, that is, the k-NN search with $k = 1$. We summarize the notations used in this paper in Table 1.

2.2 Sketching by Ball Partitioning

We use the sketching based on the *ball partitioning*, which defines each bit of the sketch to express whether the data point is inside a ball or not. A *pivot* is a pair (c,r) of a data point c and a nonnegative value r which defines the *ball* with the center c and the radius r. Each bit of a sketch is defined by the ball partitioning:

$$B_{(c,r)}(x) = \begin{cases} 0, & \text{if } D(c,x) \leq r, \\ 1, & \text{otherwise.} \end{cases}$$

The sketching σ_Π of *width* w is defined as the bit concatenation using an ordered set $\Pi = \{(c_0,r_0), \ldots, (c_{w-1},r_{w-1})\}$ of w pivots:

$$\sigma_\Pi(x) = B_{(c_0,r_0)}(x) \cdots B_{(c_{w-1},r_{w-1})}(x)$$

The sketch of x is denoted by $\sigma(x)$ when Π is omitted.

Table 1: Notations.

Notation	Description
$\mathcal{U}$	the data space
$x, y, x_0, \ldots$	data points in $\mathcal{U}$
$D(x,y)$	the distance between x and y
ds	dataset $\{x_0, x_1, \ldots, x_{n-1}\}$ indexed by numbers
n	the number of data points in ds
k	the number of data points in the answer by the k-NN search
k'	the number of candidates selected by sketch filtering
k''	the number of sketches to select k' candidates
$q \in \mathcal{U}$	a query point
w	the width (length) of sketches
$\sigma(x)$	the sketch of x
ς	a sketch of unspecified data point
$\tilde{D}_p(x,\varsigma)$	the partially restored distance between x and ς
b	the number of nonempty buckets when using the sketches as keys, which is the same as the number of sketches in $\{\sigma(x) \mid x \in ds\}$

2.3 Prioritization by Partially Restored Distances

Since sketches can hold only partial information about data points, we perform the NN search using sketches in two stages to give an approximated result. The first stage, called *filtering by sketches*, selects a small subset of data points as the candidates of the answer to the query. The second stage selects the nearest neighbor from the candidates. The *recall* of the NN search is the ratio that the correct answer is retrieved, that is, the ratio that the correct answer is included in candidates selected through the filtering by sketches.

The filtering by sketches is based on the *priority* of sketches. Traditionally, the priority is given by the Hamming distance between sketches defined as the number of different bits. In this paper, we use $\tilde{D}_p$ that is an asymmetric distance function between sketches and data points, introduced in (Higuchi et al., 2018). In our previous papers, we use a notation $score_p$ for the asymmetric distance. To avoid misunderstanding and clarify the meaning of distance, we change the notation to $\tilde{D}_p$. The smaller the Hamming distance or $\tilde{D}_p$, the higher the priority to pass through the filtering.

Sketches by ball partitioning are considered bit strings consisting of quantized distances. Therefore, the distance between sketches has a quantization error. Here, we explain the partially restoration of quantization error. Let $x = 0.4$ and $y = 0.6$. Then quantized values x' and y' of them by round operation are 0 and 1, respectively. Since $D(x,y) = |0.4-0.6| = 0.2$ and $D(x',y') = |0-1| = 1$, the quantization error in distance is 0.8. When one of the raw values is used by the distance calculation, $D(x,y') = |0.4-1| = D(x',y) = |0-0.6| = 0.6$ and the error is reduced to 0.4. We call the asymmetric distance between the raw value and the quantized value the *partially restored distance* and denote it by $\tilde{D}$.

One of the keys to speeding up the NN search by sketches is the fast filtering, where data points are compressed into bit strings to avoid costly data access and distance calculation. The filtering based only on sketches is influenced by the error due to the quantization of data points to sketches. In the filtering stage, while uncompressed data points in the dataset should not be accessed, as for query both uncompressed and compressed points are available. Therefore, we can improve the filtering performance by using the partially restored distance between the uncompressed query point and compressed data points.

For a point x and a pivot (c,r), we define $e_{(c,r)}(x)$ as the minimum distance from x to the boundary of partitioning by $B_{(c,r)}$, that is,

$$e_{(c,r)}(x) = |D(c,x) - r|.$$

Suppose any point x and y are on the different side of ball partitioning by a pivot (c,r). Then, we can prove that the distance $D(x,y)$ is not less than $e_{(c,r)}(x)$ by triangular inequality. Thus, we obtain a lower bound $\delta_{(c,r)}(x,s)$ on $D(x,y)$ for any y such that $B_{(c,r)}(y) = s$ as follows:

$$\delta_{(c,r)}(x,s) = \begin{cases} 0, & \text{if } B_{(c,r)}(x) = s, \\ e_{(c,r)}(x), & \text{otherwise.} \end{cases}$$

For $p \geq 1$, we use an asymmetric distance $\tilde{D}_p$ defined by a L_p like aggregation of the distance lower bounds $\delta_{(c_i,r_i)}(x,\varsigma_i)$ as the criteria to select candidates in the first stage.

$$\tilde{D}_p(x,\varsigma) = \sqrt[p]{\sum_{i=0}^{w-1} (\delta_{(c_i,r_i)}(x,\varsigma_i))^p},$$

where (c_i,r_i) is the i-th pivot and ς_i the i-th bit of ς.

Here, we should note that $\tilde{D}_p$ is guaranteed to give a distance lower bound only if $p = \infty$. However, as we will see in the experiments, $\tilde{D}_1$ and $\tilde{D}_2$ may provide more accurate priorities for candidate selection than $\tilde{D}_\infty$.

2.4 Filtering by Sketches

We employ two methods of filtering by sketches: *full scan* and *sketch enumeration*.

Full scan
Compute $\tilde{D}_p(q,\sigma(x_i))$ for $i = 0, \ldots, n-1$, and select top-k' data points with the smallest $\tilde{D}_p$.

Sketch enumeration
Using one-by-one sketch enumeration in increasing order of $\tilde{D}_p(q,\varsigma)$, select the beginning k'' sketches from the enumeration.

The full scan is a naïve method to calculate the priorities of all data points and select the top-k' priority ones. Given a query q, it first computes $\tilde{D}_p(q,\sigma(x_i))$ for all $i = 0, \ldots, n-1$. The Hamming distance, traditionally used as a sketch priority, can be calculated very quickly. On the other hand, partially restored distances are expensive to calculate in a simple way. First, we get distance lower bounds $e_{(c_j,r_j)}(q)$ for all $j = 0, \ldots, w-1$. Next, we aggregate lower bounds corresponding to the part where the bit of $\sigma(q)$ XOR $\sigma(x_i)$ is 1. However, since the full scan calculates multiple partially restored distances, $\tilde{D}_p(q,\sigma(x_0)), \tilde{D}_p(q,\sigma(x_1)), \ldots$ for a common q, it is sufficient to calculate the lower bounds only once for q. Furthermore, a table function can speed up the aggregation. Therefore, the full scan can calculate priorities by partially restored distances at the almost same cost as for the Hamming distance. An algorithm known as QUICKSELECT selects the top-k portion from n ordered data in $O(n + k\log k)$ time. The time complexity of the full scan can be regarded as $O(n)$ because the number of candidates is very smaller than the number of searched data points, that is, $k' \ll n$.

The sketch enumeration is a method introduced in (Higuchi et al., 2019a), which uses *one-by-one* sketch enumeration in the order of priority and uses only the beginning of the enumeration. Assuming that the number of sketches enumerated before obtaining k' candidates is k'', the sketch enumeration takes a

```
function BATCHNNSEARCH1
    for i = 0 to t − 1 do
        nearest[i] ← ∞;
    for j = 0 to n − 1 do
        read x_j from ds;
        for i = 0 to t − 1 do
            if D(q_i, x_j) < nearest[i] then
                nearest[t] ← D(q_i, x_j);
                answer[i] ← j;
    return answer;
function BATCHNNSEARCH2
    for i = 0 to t − 1 do
        nearest[i] ← ∞;
        for j = 0 to n − 1 do
            read x_j from ds;
            if D(q_i, x_j) < nearest[i] then
                nearest[t] ← D(q_i, x_j);
                answer[i] ← j;
    return answer;
```

Algorithm 1: Two batch methods of NN search.

calculation time proportional to $k'' \log k''$. If the total number 2^w of w-bit sketches is small compared to n, the time required for the sketch enumeration is negligible because k'' is small and only a short beginning of the enumeration is used. On the other hand, since increasing w implies decreasing the average number of data points with the same sketch, the required computation time is increasing exponentially with respect to w. When the sketch enumeration is used, $\tilde{D}_p$ is not needed to be explicitly calculated.

2.5 Speed-up of NN Search by Parallel Processing

The main purpose of this paper is to speed up individual query processing. In experiments, we run the NN search for multiple queries to measure the average search time. We compare the search times with and without parallel processing.

Given multiple queries at a time, we consider a method of speeding up the search by batch processing. Assume that the dataset $ds = \{x_0, \ldots, x_{n-1}\}$ is stored in the secondary memory, whereas all of the t query points $q_0, \ldots, q_{t-1}$ are in the main memory. Consider the two outlines of the NN search algorithm by a double loop in Algorithm 1. Here, when the nearest neighbor to q_i is x_j, the algorithm returns the array such that $answer[i] = j$ as the query answer.

BATCHNNSEARCH1 uses the outer loop as the data loop and the inner loop to process multiple queries for each data. Since the data needs to be read from the secondary storage only once, the cost of data access decreases relatively as the number of queries increases. In addition, the internal loop of query processing can be easily parallelized with multiple threads. If this method is executed in parallel processing with 8 threads, the ground truth of 1,000 queries for YFCC100M can be obtained in about 2 hours. This is about 7 seconds per query. This speed becomes faster as the number of questions increases. However, this method is only valid for batch processing and should not be used to measure search time.

In our experiment, we run the batch process as BATCHNNSEARCH2 and measure the search time for multiple queries, where parallelization is only used within individual searches.

3 EXPERIMENTS

In experiments, we use a standard desktop computer: AMD Ryzen 7 3700X 8-Core Processor, 64GB RAM (limited for experiments to reduce the impact of disk cache), 2TB Intel 665p M.2 SSD, Ubuntu 20.04.2 LTS (Windows WSL1.0, where the file system of Windows is used). We use GCC with OpenMP to compile programs for multi-thread processing.

We use AIR to select the pivot sets for sketches narrower than 26-bit where the search precision for a small sample subset of the dataset is used as the objective function to maximize (Higuchi et al., 2020b). The current version of the pivot selection by AIR is applicable only to narrow sketches. For wide sketches, such as 48-bit and 192-bit, we use the pivot sets selected based on QBP with collision probability to minimize (Higuchi et al., 2018).

It is important to take the disk cache into account when considering the speed of retrieval from datasets stored in secondary storage. In our experiments, most of the candidate sets to achieve the recall of 90% contain more than 1 million data points. The intersection of the two candidate sets of individual queries is very small.

Since each data point of the YFCC100M is 4KB in size, each query process should read at least 4GB of data points as a potential solution. This is a small subset of a dataset of about 400GB. Therefore, in experiments measuring the search time from the YFCC100M, we limit the main memory to the size required by the program plus 4GB to reduce the effect of the disk cache to a negligible extent.

Since the size of the data point of DEEP1B is 100 bytes, it is difficult to reduce the influence of the disk cache by the same method as for YFCC100M. If the

Table 2: The searched datasets.

dataset	#p	dim	size	#q
YFCC100M	10^8	4,096	400GB	1,000
DEEP1B	10^9	96	100GB	1,000

memory is made too small, the speed of the entire search slows down. Therefore, we limit the memory to the size required by the program plus 4GB so that the processing speed is not slow, although the effect of the disk cache remains to some extent.

3.1 YFCC100M-HNfc6 and DEEP1B

We conduct experiments on the datasets YFCC100M-HNfc6 (Amato et al., 2016a) and DEEP1B (Babenko and Lempitsky, 2016).

YFCC100M consists of about 1 million data points, which are image visual features consisting of 4,096 nonnegative 32-bit floating points and normalized to unit vectors (length = 1.0) in the Euclidean space. In our experiments, we convert floating-point data into unsigned 8-bit integers after multiplying them by 1,000. The total size of YFCC100M is about 400GB. As query points, we extract 97 sets consisting of 1,000 data points. We mainly use the first set in experiments. The other 96 sets of queries are used to confirm the robustness of the quality of our search method. The rest is used as the data points to be searched. The error introduced by our conversion has a negligible effect on the NN search results.

DEEP1B consists of 1 billion data points and 10,000 query points. Data points are 96 dimensional unit vectors represented by 32-bit floating points. In our experiments, we convert them into signed 8-bit integers after multiplying 255. The total size of DEEP1B is about 100GB. We mainly use the first 1,000 queries in experiments.

Table 2 summarizes the datasets used in experiments, where #p, dim, size, and #q are the number of data points, the dimensionality of data, the size of the dataset, and the number of queries, respectively.

3.2 Scanning Speed of Naïve NN Search

At first, we present the scanning speed of the naïve NN search in Table 3, as a basis for comparison of search speeds. The values in the rows *serial* and *parallel* are microseconds needed to scan each data point by single thread processing and parallel processing with 8 threads, respectively. The columns seq, bulk, and rand include the scanning speed by sequantial read, bulk read, and random read, respectively. In bulk read, multiple consecutive data points are read together.

Table 3: Scanning speed of naïve NN search.

dataset	process	scanning speed (μs/d.p.)		
		seq	bulk	rand
YFCC100M	serial	5.2	3.0	140
	parallel	4.1	2.7	33
DEEP1B	serial	3.4	0.085	136
	parallel	0.93	0.066	19

Using these scanning speeds, we can estimate the time of the naïve NN search for YFCC100M at 520 seconds, which is shortened to 270 seconds by parallel processing with 8 threads and bulk read. On the other hand, for DEEP1B, the search for a query is estimated to take 66 seconds with 8-thread parallel processing and bulk read, which is about 50 times faster than 3,400 seconds by single thread processing without bulk read. The reason why the speedup for DEEP1B is larger than that for YFCC100M is explained by the difference in the dimensionalities and the block size of SSD. Data points of DEEP1B are represented by 96 bytes, which can be packed together in a 4KB block of SSD for every 40 data points. The size of data point of YFCC100M is the same as the block size. Scanning speeds by random reads are the basis for the second stage search because candidates after filtering by sketches may be scattered in the storage.

3.3 Priorities of Sketches

We regard a priority as *reliable* so that the recall of the NN search using the priority is high. Alternatively, we consider the priority as reliable if the number of candidates required to achieve the target recall rate is reduced. We introduced asymmetric partially restored distances, $\tilde{D}_\infty$, $\tilde{D}_1$, and $\tilde{D}_2$ (Higuchi et al., 2018), all of which are more reliable priorities than the Hamming distance. In this paper we compare the priorities on YFCC100M and DEEP1B.

Table 4 provides the reliability of priorities by 24-bit sketching as the number of candidates k' required to achieve the recall of 90%. Clearly, the three asymmetric distances are all superior to the Hamming distance. In particular, $\tilde{D}_1$ has the highest reliability. For the later experiments, we adopt $\tilde{D}_1$.

3.4 Filtering Cost

We can run the first stage filtering completely in the main memory even for the large datases such as

Table 4: Reliability of priority by 24-bit sketches: the number of candidates k' to achieve recall 90%.

dataset	Ham	$\tilde{D}_1$	$\tilde{D}_2$	$\tilde{D}_\infty$
YFCC100M	3.2M	1.4M	1.5M	2.5M
DEEP1B	9.5M	1.3M	1.7M	2.4M

Table 5: Filtering cost (sec/q).

dataset	w	k'	enum.	full scan	
				ser.	para.
YFCC100M	20	2.3M	0.0016	1.0	0.18
	22	1.8M	0.0022	1.1	0.18
	24	1.4M	0.0050	1.1	0.18
	26	1.2M	0.0150	1.0	0.18
DEEP1B	20	2.6M	0.0017	11	2.3
	22	1.6M	0.0012	9.5	2.3
	24	1.3M	0.0014	9.9	2.3
	26	0.90M	0.0020	12	2.4

Table 6: Search time (sec/q).

dataset	w	n/b	k'	ser.	para.
YFCC100M	20	110	2.3M	10	8.3
	22	37	1.8M	**9.4**	6.8
	24	15	1.4M	11	**5.8**
	26	6.8	1.2M	21	6.9
DEEP1B	20	950	2.6M	0.83	0.41
	22	240	1.6M	**0.77**	**0.24**
	24	66	1.3M	0.94	0.26
	26	23	0.90M	1.7	0.44

YFCC100M and DEEP1B. We prepare all sketches for the searched data points in advance. Table 5 summarizes the filtering costs. The number of candidates k' is set to achieve the recall of 90% for each sketch width w. The column *enum* provides the costs by the filtering by sketch enumeration. The columns of *fulls can* provide the costs by serial processing and 8-thread parallel processing. At this moment, we have no implementation of sketch enumeration in a parallel environment. For all w in Table 5, the full scan costs almost independently on w and can be accelerated by multi-thread processing. The filtering by sketch enumeration has a very small cost compared to the full scan.

As we will see in Sect. 3.5, the NN search requires more than 5.8 seconds per query for YFCC100M and 0.24 seconds for DEEP1B. The search times include the whole query processing. Therefore, the costs of filtering by sketch enumeration are very small compared with the search times by narrow sketches.

3.5 Search Time

As shown in Table 3, the difference of the speed between sequential and random read in the secondary storages is big. Therefore, we can expect that sorting data points in the sketch order can speed up the search. To observe the effect by sorting, we first run the NN searches on the unsorted dataset. Search times are given by the average seconds per query, sec/q. On an unsorted dataset of YFCC100M, the NN search time with k'= 1M was almost constant regardless of w, 150 sec/q by 1-thread and 36 sec/q by 8-thread. They are roughly equal to the scanning speed by the random read of 140 µs/data and 33 µs/data, respectively. Similarly, the search times on the unsorted dataset of DEEP1B are equal to the scanning speed by random read. Therefore, we can see that the data access in the search using the unsorted dataset is almost random.

Table 6 summarizes the search times on the sorted datasets, where the number of candidates k' is set to achieve the recall of 90% for each width. The sketch enumeration is used in the filtering. The ratio n/b is the average number of data points in nonempty buckets. When n/b is large, multiple data points with a common sketch can be efficiently read from the sorted dataset stored in the secondary storage. The columns *ser.* and *para.* represent the search times by serial processing and 8-thread parallel processing. For each dataset, the shortest search time is shown in bold with and without parallel processing.

For YFCC100M, the shortest search time is 9.4 seconds per query when 22-bit sketches are used by single-thread, and 5.8 seconds per query when 24-bit sketches by 8-thread processing. The speed-up ratio by sketches is 300 / 9.4 = 32 and 270 / 5.8 = 47 for single-thread and 8-thread, respectively.

For DEEP1B, the fastest search is realized by 22-bit sketches, regardless of whether parallel computing is used, and they are 0.77 and 0.24 seconds per query for single-thread and 8-thread processing, respectively. The speed-up ratio is 85 / 0.77 = 110 and 66 / 0.24 = 280 for single-thread and 8-thread, respectively.

The wider the sketches, the smaller k' and the larger n/b. Thus, there is a trade-off between k' and n/b. For example, the 2nd stage search of YFCC100M by 1-thread requires 10 seconds to scan 2.3M candidates, which corresponds scanning speed 4.3 µs/d.p. Table 7 summarizes the scanning speeds in the 2nd stage of the search by narrow sketches. There is a clear tendency that the larger the width, the slower the scanning speed. The fastest search is achieved by 24-bit sketches for YFCC100M and by the 22-bit sketches for DEEP1B.

Table 7: Scanning speed of 2nd stage search (μs/d.p.).

dataset	w	n/b	ser.	para.
YFCC100M	20	110	4.3	3.6
	22	37	5.2	3.8
	24	15	7.9	4.1
	26	6.8	18	5.8
DEEP1B	20	950	0.29	0.11
	22	240	0.48	0.15
	24	66	0.72	0.20
	26	23	1.9	0.49

3.6 Comparison of Narrow Sketches with Wide Sketches

To compare the performance of narrow sketches with wide sketches, we execute additional experiments. 256-bit, 512-bit and 1024-bit sketches were used for YFCC100M, and 96-bit, 192-bit and 384-bit for DEEP1B. Filtering with wide sketches is done by the full scan, since the filtering by sketch enumeration works only for narrow sketches. Table 8 summarizes the results by 8-thread parallel processing, where the columns *filt.* and *2nd* include the filtering cost and the cost of the second stage search, respectively. The number of candidates k' is set to achieve the recall of 90% for each w.

The wider the sketches, the smaller the number of candidates and the larger the filtering cost. Furthermore, the scanning cost of the second stage of the search becomes large as by random read. For example, for YFCC100M, the 2nd stage of the search with 256-bit sketches requires 19 seconds to scan 450K candidates, which corresponds to 43 μs/d.p., so it is a bit slower than the scanning speed by random read shown in Table 3. Thus, it is hard for wide sketches to provide a faster search than narrow sketches.

For YFCC100M, 512-bit and 1024-bit sketches provide a bit faster search than narrow sketches. However, wider sketches than 1024-bit cannot provide a faster search. In contrast, no wide sketches provide a faster search for DEEP1B. The difference between YFCC100M and DEEP1B is explained by the scanning speed. For YFCC100M, scanning speed by bulk sequential read is about 10 times faster as by random read. For DEEP1B, the speed ratio of bulk sequential read to random read is more than 250.

4 DISCUSSION

Here, we discuss our achievements in comparison with related work.

Table 8: Search time by wide sketches (sec/q).

dataset	w	k'	filt.	2nd	total
YFCC100M	256	450K	0.25	19	19
	512	90K	0.50	4.0	**4.5**
	1,024	77K	1.5	3.6	5.1
DEEP1B	96	25K	2.0	0.59	2.6
	192	2,600	2.1	0.06	**2.1**
	384	400	3.0	0.01	3.0

4.1 Related Work

Before giving an overview of related work, we notice that there are two confusing notations concerning the accuracy of the search. We adopt the same notation as Amato et al. (Amato et al., 2016b), i.e. *recall@k* to denote the ratio of correct results for a given k-NN query in the top-k results returned. Formally,

$$recall@k = \frac{|E(q,k) \cap A(q,k)|}{|E(q,k)|},$$

where $|\cdot|$ denotes the number of elements in a set, $E(q,k)$ and $A(q,k)$ are the top-k sets obtained by the exact and approximate k-NN searches for a query q, respectively. This represents the quality of the entire search.

On the other hand, in (Babenko and Lempitsky, 2016; Johnson et al., 2021), they use the notation *Recall@R* to denote the rate of queries for which the true nearest neighbor is included in a set of candidates whose number of elements is R. Their interests are only in the computational cost to get the set of R candidates. *Recall@R* is considered as the quality of the indexing and does not represent the quality of the entire search since we need to select the final answer out of the candidates unless $R = 1$.

Amato et al. well explain the difficulty of NN search on YFCC100M in (Amato et al., 2016b). However, they focus strictly on the search with a very small number of candidates by sacrificing a significant portion of quality for achieving a quick search. In the paper (Amato et al., 2016a), they showed experimental results using a set of 4,096-bit strings named *Binary*. They employed the traditional Hamming distances between binary strings to select candidates and reported *recall*@100 of 75% with 7,000 candidates. Long bit strings, such as 4,096-bit, require a lot of time and space just to get the candidates.

In (Babenko and Lempitsky, 2016) that introduced DEEP1B, it is reported that the NN search with the recall of 45% can be executed in 20 milliseconds.

Both of YFCC100M and DEEP1B are also used in experiments using GPUs (Johnson et al., 2021).

Table 9: Recall@k of k-NN search.

dataset	k=1	k=10	k=20	k=100
YFCC100M	90%	86%	84%	80%
DEEP1B	90%	86%	84%	79%

However, they execute the search on-memory using compressed datasets and do not consider the cost of the final selection using the uncompressed datasets. Probably, direct handling large-scale dataset is not appropriate for the computation with GPUs. In fact, for example, they use a YFCC100M dataset of reduced 128 dimension by PCA.

4.2 Recall of the k-NN Search

We have discussed the NN search targeting the recall of 90%. Table 9 summarizes the recall@k with the same settings as the NN search. The k-NN search speed is almost the same as the NN search. From this, it can be confirmed that the proposed method has a high performance even for the k-NN search.

4.3 Comparison with Related Work

We cannot directly compare the efficiency of our method with those used in related work, since there is presented no concrete time for the search or only time to get the candidates by on-memory processing with the recall lower than 90%. Therefore, here, we compare them using the experimental results under a different setting from Section 3.

In our experimental environment, we cannot run the search using *Binary* stored in the main memory, whose size is 50GB. By a rough estimation in the converted YFCC100M of 400GB on SSD, the search will run in 6 seconds to get candidates using *Binary* and 0.2 seconds to select the answer from the 7,000 candidates. In contrast, our method executes a 100-NN query with $recall@100 = 80\%$ in almost the same time 6 seconds as a NN query with the recall of 90%. Thus, our method runs at the same speed as the method reported by Amato et al. for YFCC100M with a bit higher recall and less memory consumption.

It is a bit difficult to compare our method with the methods by Babenko et al., since the computer environment such as processor and memory specification is not shown in the literature. With the recall of 45%, even if the dataset is stored in secondary memory, using our method with the 8-thread parallel processing, the search can run in almost the same time. In our computer environment, it is impossible to read the entire dataset into the main memory and execute it. When the dataset Deep500M of half size is used, the search can run on memory. If the recall rate is 45%, we can run the search for Deep500M in 1.5 milliseconds with a single thread. Therefore, under an environment with sufficient memory, we expect that the search for DEEP1B with a recall rate of 45% can be run in 3 milliseconds by our method.

We would like to emphasize that this paper focuses on rather higher recall rates since we aim to provide new insights for practical databses queries. The subject in this paper is to realize a fast search from a large-scale dataset stored in the secondary memory. Here, for reference, we show the search speed of the proposed method when the reduced dataset is loaded in the main memory. For the datasets YFCC10M and Deep100M which are subsets of one tenth size, we can run the search on memory. For YFCC10M, using 24-bit sketches, the average search time for a query is 21, 30, 47, and 120 milliseconds with recall 80%, 85%, 90%, and 95%, respectively. For Deep100M, using 24-bit sketches, the average search time for a query is 1.4, 1.9, 3.2, and 6.0 milliseconds with recall 80%, 85%, 90%, and 95%, respectively. Even for Deep500M, 3.0, 4.1, 6.8, and 14 milliseconds, respectively.

Thus, according to our best knowledge, the method used in this paper exceeds the current search engines in speed for the large datasets keeping a high recall rate.

4.4 Comparison of Sketching with Indexing Structures

Babenko et al. compared two commonly used indexing structures IVFADC and IMI with two methods NO-IMI and GNO-IMI introduced by them. The sketch-based methods are also considered indexing structures. We compare wide sketches of 48, 96, and 192-bit with the four structures. Table 10 displays the number k' of candidates to achieve the recall of 90% in searching the dataset DEEP1B. The priority of sketches is measured by the partially restored distance $\tilde{D}_1$. The values for structures 1 to 4 are taken from the paper by Babenko et al. (Babenko and Lempitsky, 2016).

We can observe that the indexing by 96-bit sketches has almost the same quality as GNO-IMI. Furthermore, the indexing by 192-bit sketches, which are only 24-byte in size, significantly outperforms GNO-IMI. However, as already seen in the preceding subsections, the fastest search with the recall 90% is provided by narrow sketches. Thus, not only the indexing quality but also the speed of selecting answers from the candidates are important.

Table 10: Comparison of indexing structures.

indexing	1	2	3	4	5	6	7
$k'(\times 10^3)$	120	60	25	15	240	25	3

k': the number of candidates to achieve the recall 90% for DEEP1B. 1. INVADC, 2. IMI, 3. NO-IMI, 4. GNO-IMI, 5. 48-bit, 6. 96-bit, 7. 192-bit

5 CONCLUSION

Using two datasets YFCC100M-HNfc6 and DEEP1B consisting of 100 million and 1 billion vectors, respectively, we have demonstrated that narrow 22-bit and 24-bit sketches provide fast and accurate NN search. We have also described how difficult it is for the wide sketches to outperform narrow sketches in search speed. The key to fast search is to keep the ability to narrow the candidates, as well as to speed up the filtering.

One of the most important future tasks is the revision of the pivot selection to make the sketching more reliable. Since the current version of pivot selection by AIR can treat only narrow sketches, we have to modify it. As we have observed in Section 4.4, that 96-bit and 192-bit sketches provide the indexing structures with high quality without using AIR. Therefore, we can expect that AIR can find indexing structures with higher quality by not so wide sketches.

We also plan to build a search engine based on double-filtering with narrow and wide sketches which can be expected to search even larger datasets than YFCC100M-HNfc6 and DEEP1B.

ACKNOWLEDGMENTS

This research was partly supported by ERDF "CyberSecurity, CyberCrime and Critical Information Infrastructures Center of Excellence" (No. CZ.02.1.01/0.0/0.0/16 019/0000822), and also by JSPS KAKENHI Grant Numbers 19H01133, 19K12125, 20H00595, 20H05962, 21H03559 and 20K20509.

REFERENCES

Amato, G., Falchi, F., Gennaro, C., and Rabitti, F. (2016a). YFCC100M-HNfc6: A large-scale deep features benchmark for similarity search. In *Proc. SISAP'16, LNCS 9939, Springer*, pages 196–209.

Amato, G., Falchi, F., Gennaro, C., and Vadicamo, L. (2016b). Deep permutations: Deep convolutional neural networks and permutation-based indexing. In *Proc. SISAP'16, LNCS 9939, Springer*, pages 93–106.

Babenko, A. and Lempitsky, V. (2016). Efficient indexing of billion-scale datasets of deep descriptors. In *Proc. CVPR'16, IEEE Computer Society*, pages 2055–2063.

Dong, W., Charikar, M., and Li, K. (2008). Asymmetric distance estimation with sketches for similarity search in high-dimensional spaces. In *Proc. ACM SIGIR'08*, pages 123–130.

Higuchi, N., Imamura, Y., Kuboyama, T., Hirata, K., and Shinohara, T. (2018). Nearest neighbor search using sketches as quantized images of dimension reduction. In *Proc. ICPRAM'18*, pages 356–363.

Higuchi, N., Imamura, Y., Kuboyama, T., Hirata, K., and Shinohara, T. (2019a). Fast filtering for nearest neighbor search by sketch enumeration without using matching. In *Proc. AI 2019, LNAI 11919, Springer*, pages 240–252.

Higuchi, N., Imamura, Y., Kuboyama, T., Hirata, K., and Shinohara, T. (2019b). Fast nearest neighbor search with narrow 16-bit sketch. In *Proc. ICPRAM'19*, pages 540–547.

Higuchi, N., Imamura, Y., Kuboyama, T., Hirata, K., and Shinohara, T. (2020a). Annealing by increasing resampling. In *Revised Selected Papers, ICPRAM 2019, LNCS 11996, Springer*, pages 71–92.

Higuchi, N., Imamura, Y., Mic, V., Shinohara, T., Kuboyama, T., and Hirata, K. (2020b). Pivot selection for narrow sketches by optimization algorithms. In *Proc. SISAP'20, LNCS 12440, Springer*, pages 33–46.

Imamura, Y., Higuchi, N., Kuboyama, T., Hirata, K., and Shinohara, T. (2017). Pivot selection for dimension reduction using annealing by increasing resampling. In *Proc. LWDA'17*, pages 15–24.

Johnson, J., Douze, M., and Jégou, H. (2021). Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547.

Matsui, Y., Uchida, Y., Jégou, H., and Satoh, S. (2018). A survey of product quantization. *ITE Transactions on Media Technology and Applications*, 6(1):2–10.

Mic, V., Novak, D., and Zezula, P. (2015). Improving sketches for similarity search. In *Proc. MEMICS'15*, pages 45–57.

Mic, V., Novak, D., and Zezula, P. (2016). Speeding up similarity search by sketches. In *Proc. SISAP'16*, pages 250–258.

Müller, A. and Shinohara, T. (2009). Efficient similarity search by reducing I/O with compressed sketches. In *Proc. SISAP'09*, pages 30–38.

Shinohara, T. and Ishizaka, H. (2002). On dimension reduction mappings for approximate retrieval of multi-dimensional data. In *Progress of Discovery Science, LNCS 2281, Springer*, pages 89–94.

Wang, Z., Dong, W., Josephson, W., Q. Lv, M. C., and Li, K. (2007). Sizing sketches: A rank-based analysis for similarity search. In *Proc. ACM SIGMETRICS'07*, pages 157–168.