Principal Component Hashing: An Accelerated Approximate Nearest Neighbor Search

Yusuke Matsushita and Toshikazu Wada

Graduate School of Systems Engineering, Wakayama University
930 Sakaedani, Wakayama, 640-8510, Japan
{ymatsushita,twada}@vrl.sys.wakayama-u.ac.jp

Abstract. Nearest Neighbor (NN) search is a basic algorithm for data mining and machine learning applications. However, its acceleration in high dimensional space is a difficult problem. For solving this problem, approximate NN search algorithms have been investigated. Especially, LSH is getting highlighted recently, because it has a clear relationship between relative error ratio and the computational complexity. However, the p-stable LSH computes hash values independent of the data distributions, and hence, sometimes the search fails or consumes considerably long time. For solving this problem, we propose Principal Component Hashing (PCH), which exploits the distribution of the stored data. Through experiments, we confirmed that PCH is faster than ANN and LSH at the same accuracy.

Keywords: Approximate Nearest Neighbor Search, High dimensional space, p-stable Locality Sensitive Hashing.

1 Introduction

Nearest neighbor (NN) search algorithm finds the nearest data to a query from stored data. This algorithm plays important roles in wide varieties of applications, e.g., NN classification [1], stitching geometric objects [2], and so on. For avoiding time consuming exhaustive search, many accelerated algorithms have been proposed, which works well on low dimensional distributions. However, most of them lose effect on high dimensional data distributions, i.e., the computational efficiency decreases almost comparably as the exhaustive search.

For solving this problem, approximated NN search algorithms have been proposed. Approximate Nearest Neighbor (ANN [4, 5]) and Locality Sensitive Hashing (LSH [6, 7]) are the typical examples.

ANN is the k-d tree [3] based search algorithm which first finds an NN candidate by binary tree search and checks other possibilities in the following procedure. This procedure is called priority search. The binary tree corresponds to a box decomposition of the search space, where each box involves a single vector. In the priority search, the algorithm checks the boxes intersecting the hyper sphere whose center is at the query vector and the NN candidate is on its surface. The approximation is reducing the radius of this sphere. Let feasible error and radius ε and r, respectively.

T. Wada, F. Huang, and S. Lin (Eds.): PSIVT 2009, LNCS 5414, pp. 374–385, 2009.

Then the approximation is reducing the radius r to $r/(1+\varepsilon)$. This reduction decreases the number of boxes checked in the priority search, but increases the chance of inaccurate NN search. Of course, $\varepsilon = 0$ corresponds to exact NN search with no errors.

On the other hand, LSH is the hash based approximation of NN search, which has a clear relationship between error ratio and the computational complexity, where

$$ErrorRatio = \frac{\text{distance between query and its approximate NN}}{\text{distance between query and its true NN}}.$$

The basic LSH decomposes the search space into buckets (hash bins), each of which has the same hash value. This algorithm first computes the hash value of the query and finds the NN candidates in the bucket having the same hash value. Finally, it finds approximate NN vector from the candidates. Therefore, a few candidates are preferable for fast search but are not preferable for accurate search.

In the basic LSH [6] and p-stable LSH [7], the hash function is determined without referring the distribution of stored vectors. This causes the following problems:

P1. When the query is given at low density area, the search may fail, because no bucket may have the same hash value with the query.

P2. When the query is given at high density area, the search time may increase, because those buckets usually include more data than low density area.

For solving this problem, we propose Principal Component Hashing (PCH), which exploits the distribution of stored vectors for computing hash function. This NN search algorithm has the following advantages.

- PCH decomposes whole search space into finite buckets involving the same expected number of vectors. This guarantees constant search time independent of query vectors. Also, PCH can find NN candidates for any query vector.
- PCH finds the NN vector from the NN candidates by efficient distance computation on the principal components.

PCH assumes that data distribution obeys Gaussian distribution. However, most practical data distribution does not. Hence, we further extend it to NN search algorithm for general distributions while guaranteeing the above advantages. We call it Adaptive PCH (A-PCH).

2 Approximate Nearest Neighbor Search

Many researches on accelerating NN search have been done before. Through those researches, most algorithms use the following two techniques.

[Reducing the number of distance computation]. The NN candidates for distance computation are narrowed based on the triangular inequality [8, 9, 10] or the space decomposition [4, 11, 12].

[Pruning of distance computation]. The pruning stops distance computation when halfway distance exceeds given tentative distance [4].

This research field has been regarded matured, because many researchers spent long time and some accelerated search algorithms have been produced. However, their computational efficiency decreases almost comparably as the exhaustive search.

For solving this problem, approximated NN search algorithms, e.g., ANN and LSH, have been proposed. Especially, LSH is getting highlighted recently, because it has a clear relationship between relative error ratio and the computational complexity.

2.1 $(R,c)-$Nearest Neighbor Problem

Suppose X is a metric space and $\boldsymbol{x}_1, \boldsymbol{x}_2 \in X$. Let $D(\boldsymbol{x}_1, \boldsymbol{x}_2)$ be the distance between $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, $S\ (\subset X)$ be the stored vector set, $\boldsymbol{q}$ be a query, and $\boldsymbol{NN}(\boldsymbol{q}) \in S$ be the nearest vector to $\boldsymbol{q}$ within S. Then, (R,c)-NN problem is to find an approximate nearest neighbor vector $\boldsymbol{NN}'(\boldsymbol{q})$ satisfying

$$D(\boldsymbol{q}, \boldsymbol{NN}'(\boldsymbol{q})) \le cD(\boldsymbol{q}, \boldsymbol{NN}(\boldsymbol{q})), \tag{1}$$

where $c(\ge 1)$ is called error ratio.

For solving this problem, we define the following hash function:

Definition 1. *Let U be a set of hash values, $h(x): X \to U$ be the hash function, locality-sensitive hash function satisfies the conditions below:*

- *if $D(\boldsymbol{v},\boldsymbol{q}) \le r_1$ then $\Pr[h(\boldsymbol{q}) = h(\boldsymbol{v})] \ge p_1$,*
- *if $D(\boldsymbol{v},\boldsymbol{q}) > r_2$ then $\Pr[h(\boldsymbol{q}) = h(\boldsymbol{v})] < p_2$,*

where $p_2 \le p_1$ and $r_2 = cr_1$.

By using those hash functions satisfying this definition, we can realize (R,c)-NN search based on the following theorem:

Theorem 1. *Let $h_1, h_2, \cdots$ be hash functions, n be the number of vectors in the dataset, and $\rho(c) = \ln p_1 / \ln p_2$. Then, it is possible to find $\boldsymbol{NN}'(\boldsymbol{q})$ satisfying Equation* (1) *by $L = n^{\rho(c)}$ times bucket search with constant probability.*

LSH is an approximate NN search algorithm based on this theorem, whose efficiency is characterized by $\rho(c)$. For realizing better search algorithm, which finds approximate NN vector with high accuracy ($|c-1|$ is small) within short time ($\rho(c)$ is small), $\rho(c)$ should decrease quickly. Various researches are being conducted about what kind of hash function brings good $\rho(c)$.

2.2 P-Stable LSH

P-stable LSH is an example of practical LSH, which finds approximate NN vector in Euclidean distance. Suppose $\boldsymbol{q}$ is a query, $\boldsymbol{a}$ is a vector, b and ω are constants. Then the p-stable hash function is defined by the following formula.

$$h_{\boldsymbol{a},b}(\boldsymbol{q}) = \left\lfloor \frac{\boldsymbol{a} \cdot \boldsymbol{q} + b}{\omega} \right\rfloor, \tag{2}$$

where $\lfloor . \rfloor$ is floor function.

This hash function projects vectors onto the vector $\boldsymbol{a}$ and quantize the axis with the interval ω, which is decided based on the distribution width of the inner product $\boldsymbol{a} \cdot \boldsymbol{q}$. In this sense, the parameter b can be regarded as adjusting the bias, which is chosen uniformly from the range $[0, \omega]$. $\boldsymbol{a}$ is sampled from a p-stable distribution, for example, isotropic Gaussian. Depending on the property of p-stable distribution, it can be proven that the hash function achieves $\rho(c) \leq 1/c$ [6].

Recently, [8] claims that $\rho(c) = 1/c^2$ can be achieved by using Voronoi decomposition of search space. However, this is impractical in high dimensional space, because the computational complexity of the Voronoi decomposition over n samples in d dimensional space is $O(n^{\lfloor d/2 \rfloor})$.

3 Principal Component Hashing

Here we describe the algorithm of PCH. This algorithm performs 1) hash value computation, 2) NN candidate generation, 3) refinement of NN candidate to find approximate NN. We will explain these three processes and some tips for improving the performance.

3.1 Hash Functions

In the p-stable LSH, parameters of the hash function are determined independent of the data distribution. However, the accuracy and the efficiency can vary depending on the data distribution. For example, we can easily generate the data distribution and query that causes problems P.1 and P.2 described in section 1. This means $\rho(c)$ does not guarantee the actual performance but just illustrates the trend of accuracy versus speed independent of the data distribution.

Our basic idea is to use the data distribution for designing the hash function. In practice, we use the principal components of the distribution instead of $\boldsymbol{a}$. This is because the standard deviation of the projected vectors is maximized when vectors are projected to the principal component. This implies projected vectors are widely distributed on the principal component.

Once vectors are projected, we have to segment the projection axis into buckets. Of course, optimally segmented buckets should involve the same number of vectors. If we know the probabilistic data distribution $p(x)$ on the projection axis, we can compute cumulative probability distribution $P(x)$ as

$$P(x) = \int_{-\infty}^{x} p(\xi) d\xi \tag{3}$$

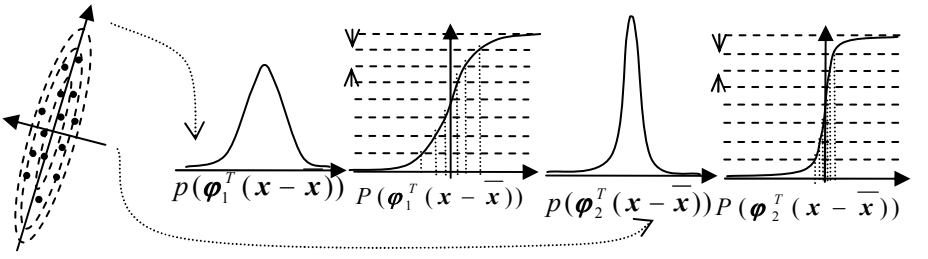

Fig. 1. The hash function and bucket division **in PCH**

$P(x)$ is monotonically increasing. Also, its domain and the range are $(-\infty,+\infty)$ and $[0,1]$, respectively. This implies that there is an inverse mapping $P^{-1}:[0,1]\mapsto(-\infty,+\infty)$.

Hence, by dividing the range of $P(x)$ into $n+1$ uniform intervals $[0,\Delta]$, $(\Delta,2\Delta]$, $\cdots,(n\Delta,1]$, the whole projection axis can be decomposed into $n+1$ disjoint buckets: $(-\infty,P^{-1}(\Delta)]$, $(P^{-1}(\Delta),P^{-1}(2\Delta)]$, $\cdots$, $(P^{-1}(n\Delta),+\infty)$ as shown in Fig. 1. This disjoint decomposition guarantees

- Every query must fall into a bucket.
- Every bucket involves the same expected number of vectors.

These facts are most suitable for approximate NN search. In p-stable LSH, queries provided at low density area can easily fail, but PCH never fails without adding exception handling code. Also, expected number of vectors contained by a bucket directly influences the efficiency of the search. Then, equal expected number implies constant search time.

For the realization of this idea, we introduce an assumption:

Assumption 1. *The distribution of the stored vector is Gaussian.*

Assuming this, we can say that the projected vectors to a principal component also obey Gaussian distribution. Then we can fit Gaussian $p(x)$ to the projected vectors.

In practice, Equation (3) should not be computed when performing search, because it consumes considerably long time. In this research, since $p(x)$ is a Gaussian distribution, Equation (3) is approximated by the sigmoid function shown below.

$$P(x) \cong P_s(x) = 1/(1+e^{-x/\sigma}) \,. \tag{4}$$

This approximation is for designing a fast hash function. When the i-th principal component φ_i is used, the hash function is expressed as

$$h_i(\boldsymbol{x}) = \left\lfloor P_s(\varphi_i^T(\boldsymbol{x}-\bar{\boldsymbol{x}}))/\Delta \right\rfloor , \tag{5}$$

where Δ is the interval.

The series of independent hash functions can easily be created by using orthonormal bases φ_i $(i=1,\cdots,M)$ obtained by performing PCA on the given dataset. These hash functions corresponds to a lattice decomposition of the whole search space.

By using above hash functions, each bucket on an axis i has a single hash value H. Hereafter, we denote this bucket B_{iH}. That is,

$$B_{iH} = \{\boldsymbol{x} \mid \boldsymbol{x} \in S, h_i(\boldsymbol{x}) = H\}, \tag{6}$$

where S represents the search space.

3.2 Generation of NN Candidates

According to the discussion above, when the hash values of a query $\boldsymbol{q}$ are $h_i(\boldsymbol{q})$ $(i = 1, \cdots, m)$, we should find the candidates in $\bigcap_{i=1}^{m} B_{ih_i(\boldsymbol{q})}$. This strategy drastically reduces the number of NN candidates, however, it may produce empty set of candidates and may produce erroneous search results when the query is located near the boundary between buckets.

Hence, the candidates should be in those buckets which have at least one hash value $h_i(\boldsymbol{q})$. This means initial estimate of candidate set $C_0(\boldsymbol{q})$ for query $\boldsymbol{q}$ should be the union of $B_{ih_i(\boldsymbol{q})}$:

$$C_0(\boldsymbol{q}) = \bigcup_{i=1}^{m} B_{ih_i(\boldsymbol{q})}. \tag{7}$$

The problem remaining here is the candidates in $C_0(\boldsymbol{q})$ are still too many for distance computation. For reducing the number of candidates, PCH performs "refinement of candidates".

3.3 Refinement of NN Candidates

When performing the hashing, we can count the frequency of hits for each stored vector $\boldsymbol{x}$, i.e., how many times hash values match. We represent this frequency $w(\boldsymbol{x})$. According to this value, we can select a tentative NN vector $\boldsymbol{NN}^0(\boldsymbol{q})$:

$$\boldsymbol{NN}^0(\boldsymbol{q}) = \arg\max_{\boldsymbol{x} \in C_0(\boldsymbol{q})} w(\boldsymbol{x}). \tag{8}$$

Then the tentative distance z can be expressed as

$$z = D(\boldsymbol{q}, \boldsymbol{NN}^0(\boldsymbol{q})). \tag{9}$$

This tentative distance is used for pruning the distance computation, i.e., while computing the distance between $\boldsymbol{q}$ and a stored vector $\boldsymbol{x}$, whenever the halfway distance grows bigger than z, the distance computation can be terminated.

This type of pruning is also employed in ANN [4], however, the pruning in PCH is much more efficient. This is because the distance computation can be done on the principal axes.

In the PCH, we first apply PCA to stored vectors and all vectors are projected onto the principal axes $\boldsymbol{\varphi}_i$ $(i = 1, \cdots, M)$, i.e., orthonormal bases. In this case, L_p distance $D(\boldsymbol{x}_1, \boldsymbol{x}_2)$ between $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ can be expressed as

$$D(\boldsymbol{x}_1, \boldsymbol{x}_2) = \sqrt[p]{\sum_{i=1}^{M} \left| x_{1i} - x_{2i} \right|^p} = \sqrt[p]{\sum_{i=1}^{M} \left| \boldsymbol{\varphi}_i^T (\boldsymbol{x}_1 - \boldsymbol{x}_2) \right|^p} . \tag{10}$$

This is based on the Parseval's identity.

If $\boldsymbol{\varphi}_i$ is sorted in descent order of eigen values, projection to $\boldsymbol{\varphi}_1$ has the biggest deviation. This implies that many candidates can be pruned only by comparing $\left| \boldsymbol{\varphi}_1^T \boldsymbol{q} - \boldsymbol{\varphi}_1^T \boldsymbol{x} \right|$ with z, i.e., if $\left| \boldsymbol{\varphi}_1^T \boldsymbol{q} - \boldsymbol{\varphi}_1^T \boldsymbol{x} \right|$ is bigger than z then $\boldsymbol{x}$ can not be a candidate of $\boldsymbol{NN}'(\boldsymbol{q})$. This pruning can be generalized using multiple bases as below.

Suppose $m \le M$ and $\boldsymbol{x} \in C_0(\boldsymbol{q})$, if the following inequality is satisfied, $\boldsymbol{x}$ cannot be a candidate of $\boldsymbol{NN}'(\boldsymbol{q})$.

$$D^p(\boldsymbol{q}, \boldsymbol{x}) = \sum_{i=1}^{m} \left| \boldsymbol{\varphi}_i^T \boldsymbol{q} - \boldsymbol{\varphi}_i^T \boldsymbol{x} \right|^p > z^p . \tag{11}$$

In practice, this pruning does not require special computation. For computing hash function $h_i(\boldsymbol{q})$, $\boldsymbol{\varphi}_i^T \boldsymbol{x}$ is also obtained. Just by using this value, we can prune the distance computation and refine the candidate based on the inequality (11). This is because $\boldsymbol{\varphi}_i^T \boldsymbol{x}$ is already computed when vectors are stored. We show an algorithm of "refinement the NN candidates" below.

This algorithm computes $\sum_{i=1}^{A} \left| \boldsymbol{\varphi}_i^T \boldsymbol{q} - \boldsymbol{\varphi}_i^T \boldsymbol{x} \right|^p$ and check the inequality (11) within a range $1 \le i \le A (<< m)$. If the inequality (11) is not satisfied with $i = A$, compute actual distance $D(\boldsymbol{x}, \boldsymbol{q})$ and compare $D(\boldsymbol{x}, \boldsymbol{q})$ with z. If $D(\boldsymbol{x}, \boldsymbol{q}) < z$ then z is updated, otherwise $\boldsymbol{x}$ is excluded from the candidates. We perform this processing for all NN candidates in $C_0(\boldsymbol{q})$.

This algorithm directly finds NN vector from $C_0(\boldsymbol{q})$ without generating series of candidates and decreases the chances of actual distance computations by updating of z accelerates the pruning of distance computation. Since the performance depends on the parameter A, we have to find the best parameter for each problem.

3.4 Tips for Improving the Performance

In this section, we describe two tips for improving the performance of PCH to achieve higher accuracy and faster speed for practical use.

3.4.1 Bucket Overlapping

It is important for accurate search to select buckets including true NN $\boldsymbol{NN}(\boldsymbol{q})$. However, when the query is given near the boundary between buckets, $\boldsymbol{NN}(\boldsymbol{q})$ may not be in the bucket. The essential problem is not the size but the disjoint arrangement of the buckets. Then, we introduce the overlapped arrangement of buckets as follows.

$$B_{iH}^{\delta} = \{ \boldsymbol{x} \mid \boldsymbol{x} \in S, H - \delta \le h_i(\boldsymbol{x}) \le H + \delta \} . \tag{12}$$

By employing this arrangement, $\Pr[\boldsymbol{NN}(\boldsymbol{q}) \in B_{iH}^{\delta}] \geq \Pr[\boldsymbol{NN}(\boldsymbol{q}) \in B_{iH}]$ holds. δ is called a margin. Bigger δ produces more accurate search results.

3.4.2 Cutoff the NN Candidates

When we employ the bucket overlapping, the number of candidates will increase. In spite of the efficient pruning of the distance computation, too many candidates slow down the search speed. In such cases, we can reduce the number of candidates while keeping the accuracy according to the hit frequency $w(\boldsymbol{x})$, because the data having bigger $w(\boldsymbol{x})$ can have bigger chance to become $\boldsymbol{NN}(\boldsymbol{q})$. In practice, candidates $C_0(\boldsymbol{q})$ are sorted in the descent order of $w(\boldsymbol{x})$, and top b % of sorted candidates are extracted as reduced NN candidates $C_0'(\boldsymbol{q})$. This b is named *cutoff ratio*.

3.5 Extension of PCH to General Distribution

For constructing the basic PCH algorithm, we introduced **ASSUMPTION 1** that stored data obeys Gaussian distribution. From this distribution model, cumulative distribution model is approximated by the sigmoid function, and by segmenting the cumulative probability (vertical axis) uniformly, we get non-uniform buckets on the projection axis. However, once we get this non-uniform bucket decomposition, it can be stored in a tree structure, which can be utilized in the search process. This implies we don't have to rely on the **ASSUMPTION 1.**

In this section, we extend the PCH to Adaptive PCH (A-PCH), which can be applied to general distributions.

3.5.1 A-PCH: Extension to General Distributions

In the data storing stage, all vectors are projected on a principal axis. Then, we can generate cumulative histogram $H(x)$. By scanning this histogram, the domain can be decomposed into non-uniform buckets involving the same number of projected vectors. Suppose that the intervals $(-\infty, H^{-1}(\Delta)]$, $(H^{-1}(\Delta), H^{-1}(2\Delta)], \cdots$, $(H^{-1}(n\Delta), +\infty)$ represent the bucket decomposition, then the tree structure can be constructed as below.

The root node of the tree has a threshold $I(\lfloor (n+1)/2 \rfloor) = H^{-1}(\lfloor \Delta(n+1)/2 \rfloor)$. The left descendant node has $I(\lfloor (n+1)/4 \rfloor)$ and right node has $I(\lfloor 3(n+1)/4 \rfloor)$. By recursively applying this rule, we can generate a balanced binary search tree.

In the search stage, A-PCH performs binary search using this tree structure for finding the bucket where the query falls in. The rest of the process is the same as PCH.

4 Experiments

For evaluating PCH and A-PCH, we conducted the following experiments:

4.1 Performance comparison among A-PCH, ANN and p-stable LSH.
4.2 Performance comparison between PCH and A-PCH.

The specification of the platform PC is: CPU (Intel Xeon 3.72GHz x 2), Memory (32 GB), OS (x86_64 Linux Kernel Version 2.6.20), Compiler (gcc 4.1.2-13 built for x86_64).

For the fair evaluation, we have to take account of the balance between the accuracy and the speed. Then, we employ "time versus error ratio curve". Curves passing closer to left-bottom corner have better performance.

4.1 Comparison among A-PCH, ANN, and LSH

The purpose of this experiment is to compare the properties of A-PCH, LSH, and ANN under special and realistic conditions.

As for the special condition, we use the following dataset and queries:

> Dataset: 5000 vectors sampled from 3000-dimensional isotropic Gaussian distribution. Queries: 1000 uniformly distributed queries within a hyper cube $(-3\sigma, 3\sigma)^{3000}$.

This is the most difficult dataset for A-PCH, because the distribution essentially has no principal component, and hence, PCA does not play important role. One may think that a priori knowledge on the distribution is useless in this case.

For the realistic condition, we use the following dataset and queries:

> Dataset: 10000 vectors sampled from 4096-dimensional (64x64) monochrome images (CASPEAL). Queries: 1000 images independent of stored images.

This is a suitable dataset for A-PCH, because the distribution may be biased, and hence, the knowledge on the distribution can be utilized for generating hash functions and pruning the candidates.

For both datasets, all parameters in each NN search algorithm are changed as possible as we can:

- ANN: Feasible error ε is changed within the range $[1,100]$.
- LSH: Number of projections per hash value k is changed within the range $[2,40]$. Number of hash tables L within the range $[3,171]$.
- A-PCH: Number of a principal axis A within the range $[5,100]$, Number of buckets on an axis $1/\Delta$ within the range $[5,100]$, Bucket overlapping δ within the range $[0,2]$, Cutoff ratio b within the range $[20,80]$.

Fig. 2 and 3 show the "time versus error ratio curves" using simulated data and CASPEAL image dataset, respectively. In these graphs, horizontal axis represents time [s] and vertical axis is error ratio. P-stable LSH can generate "search failures", which means no NN vector is found. In this case, the graph is plotted using successful search results. ANN and A-PCH do not generate any search failure. This means the plot of p-stable LSH overestimates its performance.

For the simulated data, the performance seems almost the same, but p-stable LSH generates a lot of search failures (from 0 to 997 failures per 1000 queries). The maximum number of failures are observed at K=40 and L=3, which is plotted at (0.000379[s], 1.0298 error ratio). This implies the leftmost area of p-stable LSH

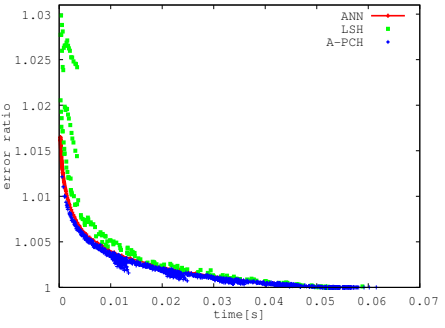

Fig. 2. Isotropic Gaussian

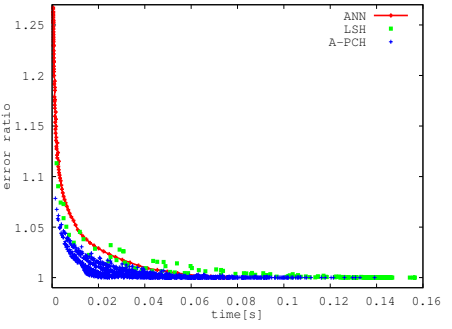

Fig. 3. CAS-PEAL image database

plotting is not reliable. ANN performs better than p-stable LSH, because p-stable LSH does not make initial guess but ANN makes initial guess by k-d tree search and approximated priority search accelerates the search keeping the accuracy. In spite that this is the most difficult case for A-PCH, it performs better than others. This is because the initial guess using the hit frequency and the pruning work better than ANN.

For the real data, LSH produces less search failures (from 0 to 42 failures per 1000 queries) than the simulated data and seems better than ANN. Comparing with them, A-PCH outperforms. This is because the performance of A-PCH mainly depends on the dimensionality of the data distribution, however, others depends on the dimensionalities of search space as well as distribution.

4.2 Comparison between PCH and A-PCH

The purpose of this experiment is to compare A-PCH with PCH, for clarifying the effect of the extension described in section 3.5.

PCH cannot guarantee "each bucket includes the same expected number of vectors" when data distribution does not obey Gaussian. But, A-PCH can guarantee that in every case. This may make some differences.

Also, we will not use empty buckets for avoiding search failure in PCH and A-PCH. Then, the maximum number of buckets along each projection axis in PCH is less than that of A-PCH. This also makes some differences.

We conducted comparative experiments with PCH and A-PCH. In the experiment, we applied them to the data used in section 4.1 and the following additional data:

> Dataset: 10000 data sampled from mixture of two isotropic 3000-dimensional Gaussians 2σ apart each other, where σ represents the standard deviation of each Gaussian. Query: Sampled data from this distribution but independent of the stored vectors.

This data is for simulating non-Gaussian distribution.

The parameter ranges of PCH and A-PCH are same as that of A-PCH in section 4.1 except the number of buckets is limited not to produce empty buckets.

Fig. 4, 5, and 6 shows the comparison experiment result of A-PCH and PCH using the isotropic distribution, CASPEAL image dataset, and the data sampled from a

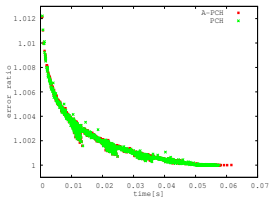

Fig. 4. Isotropic Gaussian: A-PCH vs. PCH.

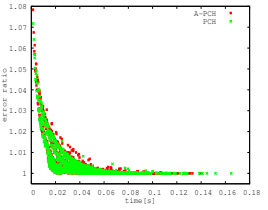

Fig. 5. CASPEAL Image database: A-PCH vs. PCH.

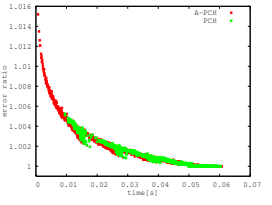

Fig. 6. Mixture of Gaussians: A-PCH vs. PCH.

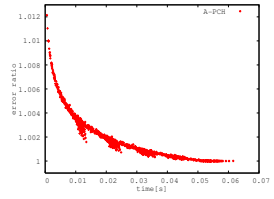

Fig. 7. Isotropic Gaussian. Result of A-PCH.

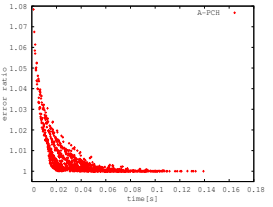

Fig. 8. CASPEAL Image database. Result of A-PCH.

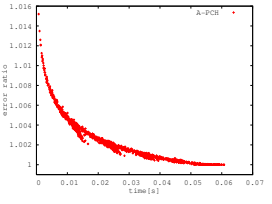

Fig. 9. Mixture of Gaussians. Result of A-PCH.

mixture of Gaussians, respectively. The experiment result of A-PCH is hidden behind that of PCH. So it is shown in Fig. 7, 8, and 9 independently.

In Fig. 4, the performances seem almost the same between PCH and A-PCH. This is because the data distribution in Fig. 4 is a perfect Gaussian. In Fig. 5, the performances seem slight difference but the best performances between PCH and A-PCH are almost the same.

In Fig. 6, the best performance of A-PCH is slightly better than PCH. This trend can be observed from 0.01[s] to 0.035[s] in search time. This is because the non-uniform numbers of vectors involved in buckets.

Also, in this figure, in the very short time area less than 0.008[s] no PCH plot can be found. This is because the maximum bucket number of A-PCH is bigger than PCH for avoiding the search failures.

5 Conclusions

This paper presents Principal Component Hashing (PCH) and its extension Adaptive PCH (A-PCH). Both of them exploit the properties of distributions of stored data. PCH projects data to principal axes, where each axis is decomposed into disjoint buckets involving the same number of stored data. This disjoint decomposition guarantees 1) no search failure and 2) constant search time. In the search stage, a NN candidate set is extracted using hash functions and the hit frequency of hash values is utilized for making initial guess of the tentative NN candidate. By using this initial guess and NN candidates, we can efficiently pick up the approximate NN by pruning the distance computations. PCH assumes that the stored data obey a Gaussian

distribution. For removing this assumption, we extended PCH to A-PCH, which can be applied to dataset obeying wide varieties of distributions.

Through extensive experiments, we confirmed that PCH and A-PCH perform better than ANN and standard p-stable LSH without producing search failures. Also A-PCH performs better than PCH for non-Gaussian distributions.

References

1. Cover, T.M., Hart, P.E.: Nearest neighbor pattern classification. IEEE Transactions on Information Theory IT-13(1), 21–27 (1967)
2. Zhang, Z.: Iterative Point Matching for Registration of Free-Form Curves and Surfaces. Tech. Report INRIA, No 1658 (1992)
3. Bentley, J.L.: Multidimensional binary search trees used for associative searching. Commun. ACM 18(9), 509–517 (1975)
4. Arya, S., Mount, D.M., Netanyahu, N.S., Silverman, R., Wu, A.Y.: An optimal algorithm for approximate nearest neighbor searching. Journal of the ACM 45, 891–923 (1998)
5. ANN: Library for Approximate Nearest Neighbor Searching, `http://www.cs.umd.edu/~mount/ANN/`
6. Indyk, P., Motwani, R.: Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In: Proceedings of the 30th ACM Symposium on Theory of Computing (STOC 1998), pp. 604–613 (May 1998)
7. Datar, M., Indyk, P., Immorlica, N., Mirrokni, V.: Locality-Sensitive Hashing Scheme Based on p-Stable Distributions. In: Proceedings of the 20th Annual Symposium on Computational Geometry (SCG 2004) (June 2004)
8. Andoni, A., Indyk, P.: Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. In: Proc. of FOCS 2006, pp. 459–468 (2006)
9. Vidal, R.: An algorithm for finding nearest neighbor in (approximately) constant average time. Pattern Recognition Letters 4, 145–158 (1986)
10. Mico, L., Oncina, J., Vidal, E.: A new version of the nearest-neighbor approximating and eliminating search algorithm (AESA) with linear preprocessing time and memory requirements. Pattern Recognition Letters 15, 9–17 (1994)
11. Brin, S.: Near neighbor search in large metric spaces. In: Proc. of 21st Conf. on very large database (VLDB), Zurich, Switzerland, pp. 574–584 (1995)
12. Yianilos, P.Y.: Data structures and algorithms for nearest neighbor search in general metric spaces. In: Proc. of the Fourth Annual ACM-SIAM Symp. on Discrete Algorithms, Austin, TX, pp. 311–321 (1993)