A Fast Nearest Neighbor Search Algorithm by Nonlinear Embedding

Yoonho Hwang Bohyung Han Hee-Kap Ahn
Dept. of Computer Science and Engineering
POSTECH, Korea
{cypher, bhhan, heekap}@postech.ac.kr

Abstract

We propose an efficient algorithm to find the exact nearest neighbor based on the Euclidean distance for large-scale computer vision problems. We embed data points nonlinearly onto a low-dimensional space by simple computations and prove that the distance between two points in the embedded space is bounded by the distance in the original space. Instead of computing the distances in the high-dimensional original space to find the nearest neighbor, a lot of candidates are to be rejected based on the distances in the low-dimensional embedded space; due to this property, our algorithm is well-suited for high-dimensional and large-scale problems. We also show that our algorithm is improved further by partitioning input vectors recursively. Contrary to most of existing fast nearest neighbor search algorithms, our technique reports the exact nearest neighbor—not an approximate one—and requires a very simple preprocessing with no sophisticated data structures. We provide the theoretical analysis of our algorithm and evaluate its performance in synthetic and real data.

1. Introduction

The nearest neighbor search is one of the most primitive operations in machine learning, pattern recognition, and information retrieval. It is also a crucial task in a lot of computer vision applications such as image clustering, object classification, image search, visual tracking, computational photography, and so on. Recently, the efficiency of the nearest neighbor search algorithm in both speed and space becomes a critical issue in computer vision research due to the demand for large-scale problems involving high-dimensional multimedia data. Motivated by this fact, we provide a simple but powerful algorithm to find the *exact* nearest neighbor with much less computational cost and memory requirement than existing methods.

The simplest solution for this problem is sequential scan (brute-forth search), which computes the distance from the query point to every point in database sequentially. However, the running time of this technique is proportional to the number and dimensionality of data, and it does not scale well in high-dimensional and large-scale problems even though there exist some heuristics to reduce computational cost.

To make the nearest neighbor search algorithm more efficient, various tree-based data structures have been employed. Typical examples include KD-tree [2], R-tree variations [1, 3, 8], B+-tree [12], and Cover tree [5]. They have the advantage to reduce the number of search candidates based on the tree-structured organization of data. However, they may not be appropriate for the high-dimensional and large-scale data because the construction of such data structures typically takes significant time and requires large memory space [13]. Note that the use of such data structures is particularly undesirable for dynamic data when the number of queries is small compared to the computational cost to build the data structures.

On the other hand, there are several approximate nearest neighbor search algorithms, which compromise the accuracy to improve efficiency in space and time. The Approximate Nearest Neighbor (ANN) search algorithm based on KD-tree [1] presents fairly good performance, but requires a huge amount of preprocessing time and memory. Hashing algorithms such as locality-sensitive hashing [6, 11], and spectral hashing [20], and coherency sensitive hashing [14] project data onto low-dimensional subspaces and maintain hash tables to reduce search space significantly. Hashing techniques are useful to find approximate nearest neighbors, but require substantially additional cost to be implemented for the exact nearest neighbor search.

A similar approach to our algorithm can be found in pattern matching [9], which is a dense nearest neighbor search technique between all pairs of patches in two images. The nearest neighbor search in [9] can be performed efficiently by the linear projection of data onto low-dimensional spaces using Walsh-Hadamard transform [18, 19]. The method in [9] is successfully applied to approximate nearest neighbor search by hashing for patch matching between two images [14]. However, it mainly focuses on the effective elimina-

978-1-4673-1228-8/12/$31.00 ©2012 IEEE

tion of candidates by thresholding with no careful consideration of how to maintain the nearest neighbor; it needs additional procedures to find the exact solution. The performance of several pattern matching algorithms is evaluated thoroughly in [17].

To overcome the limitations of existing techniques, we propose an efficient algorithm to find the exact nearest neighbor through non-linear embedding for large-scale and high-dimensional computer vision problems. We embed data points onto a low-dimensional space and prove that the squared Euclidean distance between two points in the embedded space is bounded by the squared distance in the original space with a constant factor. A lot of nearest neighbor candidates are eliminated by just checking the distances in the low-dimensional embedded space, and the nearest neighbor can be found among a small set of feasible candidates. It is particularly efficient for high-dimensional and large-scale data because the benefit of the dimensionality reduction is huge and we do not need to load full data to compute their distances in the original space. Furthermore, a simple recursive partitioning of input data and query improves the speed of our algorithm. We compare the performance of our technique with other exact and approximate algorithms through various experiments.

The rest of the paper is organized as follows. Section 2 describes our basic nearest search algorithm based on the low-dimensional embedding, and the vector partitioning to further improve the efficiency is presented in Section 3. Section 4 discusses the relation between our algorithm and a linear embedding technique [9]. In Section 5, we evaluate the proposed algorithm through various experiments and illustrate its performance with respect to other exact and approximate algorithms.

2. Nearest Neighbor Search by Embedding

Our algorithm embeds the data points onto a very low-dimensional space and eliminates non-nearest neighbors by comparing the distances in the embedded space. This section describes the theoretical proof of our key idea and how to apply the idea to the efficient nearest neighbor search. The early version of this idea was introduced in [10].

2.1. A Lower Bound on Squared Euclidean Distance

Let $\mathbf{x} = (x_1, \ldots, x_d)^{\mathrm{T}}$ be a d-dimensional vector. Note that $\mathbf{x}$ can represent images or videos in computer vision applications. The mean and standard deviation of the elements in $\mathbf{x} \in \mathbb{R}^d$ are given by $\mu_{\mathbf{x}} = \frac{1}{d}\sum_i^d x_i$ and $\sigma_{\mathbf{x}}^2 = \frac{1}{d}\sum_i^d (x_i - \mu_{\mathbf{x}})^2$, respectively. If another d-dimensional vector $\mathbf{y} = (y_1, \ldots, y_d)^{\mathrm{T}}$ is given, a lower bound on the squared Euclidean distance between $\mathbf{x}$ and $\mathbf{y}$ denoted by $\mathrm{dist}(\mathbf{x}, \mathbf{y})^2$ can be described with respected to $\mu_{\mathbf{x}}$, $\mu_{\mathbf{y}}$, $\sigma_{\mathbf{x}}^2$ and $\sigma_{\mathbf{y}}^2$, which is shown in the following lemma.

Lemma 1. $\mathrm{dist}(\mathbf{x}, \mathbf{y})^2 \geq d\left((\mu_{\mathbf{x}} - \mu_{\mathbf{y}})^2 + (\sigma_{\mathbf{x}} - \sigma_{\mathbf{y}})^2\right)$.

Proof.

$$\begin{aligned}
\mathrm{dist}(\mathbf{x}, \mathbf{y})^2 &= \sum_{i=1}^{d} (x_i - y_i)^2 \\
&= \sum_{i=1}^{d} \left((\mu_{\mathbf{x}} - \mu_{\mathbf{y}}) + (x_i - \mu_{\mathbf{x}}) - (y_i - \mu_{\mathbf{y}})\right)^2 \\
&= \sum_{i=1}^{d} \left((\mu_{\mathbf{x}} - \mu_{\mathbf{y}})^2 + (x_i - \mu_{\mathbf{x}})^2 + (y_i - \mu_{\mathbf{y}})^2\right) \\
&\quad - 2\sum_{i=1}^{d} (x_i - \mu_x)(y_i - \mu_y)
\end{aligned}$$

The last equality follows from the fact that $\sum_{i=1}^{d}(x_i - \mu_{\mathbf{x}}) = \sum_{i=1}^{d}(y_i - \mu_{\mathbf{y}}) = 0$. The Cauchy-Schwarz inequality is applied to the last term of the above equation as

$$\begin{aligned}
&\mid 2\sum_{i=1}^{d} (x_i - \mu_x)(y_i - \mu_y) \mid \\
&\quad \leq 2\left(\sum_{i=1}^{d} (x_i - \mu_{\mathbf{x}})^2 \sum_{i=1}^{d} (y_i - \mu_{\mathbf{y}})^2\right)^{1/2} \\
&\quad = 2d\sigma_{\mathbf{x}}\sigma_{\mathbf{y}},
\end{aligned}$$

which leads us to

$$\begin{aligned}
\mathrm{dist}(\mathbf{x}, \mathbf{y})^2 &\geq d(\mu_{\mathbf{x}} - \mu_{\mathbf{y}})^2 + d(\sigma_{\mathbf{x}}^2 + \sigma_{\mathbf{y}}^2 - 2\sigma_{\mathbf{x}}\sigma_{\mathbf{y}}) \\
&= d\left((\mu_{\mathbf{x}} - \mu_{\mathbf{y}})^2 + (\sigma_{\mathbf{x}} - \sigma_{\mathbf{y}})^2\right). \qquad \square
\end{aligned}$$

The lower bound of the squared Euclidean distance between $\mathbf{x}$ and $\mathbf{y}$ has a special form; it involves the squared Euclidean distance between $(\mu_{\mathbf{x}}, \sigma_{\mathbf{x}})$ and $(\mu_{\mathbf{y}}, \sigma_{\mathbf{y}})$. In other words, $\mathrm{dist}(\mathbf{x}, \mathbf{y})^2$ in the original dimension is comparable to $\mathrm{dist}((\mu_{\mathbf{x}}, \sigma_{\mathbf{x}}), (\mu_{\mathbf{y}}, \sigma_y))^2$, which is the distance in the two-dimensional nonlinear space based on (μ, σ). It means that the nearest neighbor search can be implemented efficiently by using the distance computations in the low-dimensional space as illustrated in Figure 1. The details are described in the next subsection.

2.2. Efficient nearest neighbor search

We first perform a simple preprocessing step of our algorithm—the computation of the lower bound,

$$\mathrm{LB}(\mathbf{x}, \mathbf{y}) = d\left((\mu_{\mathbf{x}} - \mu_{\mathbf{y}})^2 + (\sigma_{\mathbf{x}} - \sigma_{\mathbf{y}})^2\right). \qquad (1)$$

Given a set $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, we can compute and store $\mu_{\mathbf{x}}$ and $\sigma_{\mathbf{x}}$ for all $\mathbf{x} \in \mathcal{X}$ in $O(nd)$ time using $O(n)$ space. For a query $\mathbf{y}$, we obtain $\mu_{\mathbf{y}}$ and $\sigma_{\mathbf{y}}$ in $O(d)$ time and $\mathrm{LB}(\mathbf{x}, \mathbf{y})$ is computed in $O(1)$ time for any $\mathbf{x} \in X$.

The lower bound is used deliberately to reduce the number of comparisons dramatically. In specific, a lot of the nearest neighbor candidates for a query $\mathbf{y}$ are eliminated by simple comparisons as described in Algorithm 1. Figure 1 illustrates how the Euclidean distances of vectors in the embedded space is used for the filtering. In principle, the lower bound of $\mathrm{dist}(\mathbf{x}_i, \mathbf{y})^2/d$ is equal to the squared distance between $\hat{\mathbf{x}}_i$ and $\hat{\mathbf{y}}$ in the embedded space, and we can reject $\mathbf{x}_i$ if $\mathrm{LB}(\mathbf{x}_i, \mathbf{y}) = d \cdot \mathrm{dist}(\hat{\mathbf{x}}_i, \hat{\mathbf{y}})^2$ is larger than the squared Euclidean distance to the current nearest neighbor from $\mathbf{y}$.

Algorithm 1 Fast Nearest Neighbor Search

```
Pick a vector in 𝒳 as the seed x^min
MINDIST ← dist(x^min, y)^2
for i = 1 → n do
  if LB(x_i, y) ≥ MINDIST then
    continue
  end if
  if dist(x_i, y)^2 < MINDIST then
    x^min ← x_i
    MINDIST ← dist(x^min, y)^2
  end if
end for
return {x^min, MINDIST}
```

At each iteration, we maintain the nearest neighbor $\mathbf{x}^{\min}$ among the points that have been considered so far. During the procedure, we can reject $\mathbf{x}_i$ effectively if $\mathrm{dist}(\mathbf{x}_i, \mathbf{y})^2 \geq \mathrm{LB}(\mathbf{x}_i, \mathbf{y}) \geq \mathrm{dist}(\mathbf{x}^{\min}, \mathbf{y})^2$ for $\mathbf{x}_i \in \mathcal{X}, i = 1, \ldots, n$. Note that for each vector, the filtering is done in $O(1)$ time while computing the exact Euclidean distance takes $O(d)$ time; we eliminate a lot of non-nearest neighbors in constant time instead of linear time. It is a huge reduction in computational cost especially when the data are high-dimensional.

Our algorithm requires the computation of the Euclidean distance in the embedded space as well as the original space if the lower bound is smaller than the distance to the current nearest neighbor. In the worst case—when the input vectors are given in the decreasing order based on their lower bounds and the lower bounds are always smaller than the distance to the current nearest neighbor, the algorithm takes $O(nd)$ time to find the nearest neighbor because the algorithm computes both the exact Euclidean distance and the lower bound for every input. On the other hand, the algorithm achieves its best running time, $O(n)$ if $\mathrm{LB}(\mathbf{x}, \mathbf{y}) \geq \mathrm{dist}(\mathbf{x}^{\min}, \mathbf{y})^2$ holds for every vector $\mathbf{x}$, except the first input vector. Such extreme cases seems unlikely to happen; using the lower bound typically eliminates unnecessary computation and saves much time to find the nearest neighbor in average case, especially on high-dimensional vector set.

Although our algorithm is efficient to compute the nearest neighbor, there may still exist many non-nearest neighbors that fail to be rejected by the method described in Algorithm 1. To improve the performance of our algorithm by filtering more candidates, we should be able to compute a tighter lower bound. It can be achieved by a simple partitioning of vectors, and the technique including theoretical proof is presented in the next section.

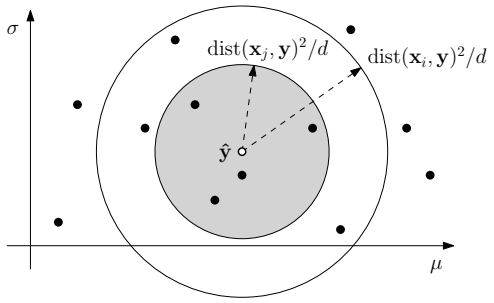


Figure 1: The vectors in $\mathcal{X}$ and the query $\mathbf{y}$ are embedded in the two-dimensional space based on their mean and standard deviation of elements. The squared Euclidean distance from a vector $\hat{\mathbf{x}} = (\mu_{\mathbf{x}}, \sigma_{\mathbf{x}})$ to the query $\hat{\mathbf{y}} = (\mu_{\mathbf{y}}, \sigma_{\mathbf{y}})$ in the embedded space is equal to $\mathrm{LB}(\mathbf{x}, \mathbf{y})/d$. If the current nearest neighbor is $\mathbf{x}_i$, the points lying outside the larger disk have their LB values bigger than $\mathrm{dist}(\mathbf{x}_i, \mathbf{y})^2$ and cannot be the nearest neighbor. If a better candidate $\mathbf{x}_j$—$\hat{\mathbf{x}}_j$ should be inside the larger disk—is found, we can further reject the points lying in the annulus between the two disks since their LB values are bigger than $\mathrm{dist}(\mathbf{x}_j, \mathbf{y})^2$.

3. Improving the Lower Bound

Suppose that we partition a d-dimensional vector $\mathbf{x}$ into two disjoint parts and obtain two vectors with half sizes, *i.e.*, $\mathbf{x}_{1,2} = (x_1, x_2, \ldots, x_{\lceil d/2 \rceil})^{\mathrm{T}}$ and $\mathbf{x}_{2,2} = (x_{\lceil d/2 \rceil + 1}, x_{\lceil d/2 \rceil + 2}, \ldots, x_d)^{\mathrm{T}}$. If we denote by $\mathbf{y}_{1,2}$ and $\mathbf{y}_{2,2}$ the first $\lceil d/2 \rceil$-dimensional vector and the second $\lfloor d/2 \rfloor$-dimensional vector of $\mathbf{y}$, respectively, a new lower bound of $\mathrm{dist}(\mathbf{x}, \mathbf{y})^2$ is given by the following lemma.

Lemma 2. $\mathrm{dist}(\mathbf{x}, \mathbf{y})^2 \geq \mathrm{LB}_2(\mathbf{x}, \mathbf{y})$, *where* $\mathrm{LB}_2(\mathbf{x}, \mathbf{y}) = \mathrm{LB}(\mathbf{x}_{1,2}, \mathbf{y}_{1,2}) + \mathrm{LB}(\mathbf{x}_{2,2}, \mathbf{y}_{2,2})$.

Proof. We have

$$\begin{aligned} \mathrm{dist}(\mathbf{x}, \mathbf{y})^2 &= \sum_{i=1}^{\lceil d/2 \rceil} (x_i - y_i)^2 + \sum_{i=\lceil d/2 \rceil + 1}^{d} (x_i - y_i)^2 \\ &= \mathrm{dist}(\mathbf{x}_{1,2}, \mathbf{y}_{1,2})^2 + \mathrm{dist}(\mathbf{x}_{2,2}, \mathbf{y}_{2,2})^2. \end{aligned}$$

The claim holds since $\mathrm{dist}(\mathbf{x}_{1,2}, \mathbf{y}_{1,2})^2 \geq \mathrm{LB}(\mathbf{x}_{1,2}, \mathbf{y}_{1,2})$ and $\mathrm{dist}(\mathbf{x}_{2,2}, \mathbf{y}_{2,2})^2 \geq \mathrm{LB}(\mathbf{x}_{2,2}, \mathbf{y}_{2,2})$. □

The next lemma reads that $\mathrm{LB}_2(\mathbf{x}, \mathbf{y})$ is a better lower bound than $\mathrm{LB}(\mathbf{x}, \mathbf{y})$. Its proof contains long and tedious derivations and we put it in Appendix A.

Lemma 3. $\mathrm{LB}(\mathbf{x},\mathbf{y}) \le \mathrm{LB}_2(\mathbf{x},\mathbf{y})$.

Lemma 3 is powerful because it does not assume any data distributions. Moreover, this lower bound can be improved further by additional partitionings. For example, if $\mathbf{x}_{1,2}$ is divided into two $d/4$-dimensional vectors, $\mathbf{x}_{1,4}$ and $\mathbf{x}_{2,4}$, we can derive $\mathrm{LB}(\mathbf{x}_{1,4},\mathbf{y}_{1,4}) + \mathrm{LB}(\mathbf{x}_{2,4},\mathbf{y}_{2,4}) \ge \mathrm{LB}(\mathbf{x}_{1,2},\mathbf{y}_{1,2})$ by applying Lemma 3 to this subdivision. In a similar way, we can prove $\mathrm{LB}_4(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{4} \mathrm{LB}(\mathbf{x}_{i,4},\mathbf{y}_{i,4}) \ge \mathrm{LB}_2(\mathbf{x},\mathbf{y})$ and $\mathrm{LB}_{16}(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{16} \mathrm{LB}(\mathbf{x}_{i,16},\mathbf{y}_{i,16}) \ge \mathrm{LB}_4(\mathbf{x},\mathbf{y})$ by applying Lemma 3 recursively. In summary, we have

$$\mathrm{LB}_2(\mathbf{x},\mathbf{y}) \le \mathrm{LB}_4(\mathbf{x},\mathbf{y}) \le \mathrm{LB}_{16}(\mathbf{x},\mathbf{y}) \le \mathrm{dist}(\mathbf{x},\mathbf{y})^2. \quad (2)$$

These relations are more intuitive in image and video data because the standard deviation in a small patch tends to be smaller than the one in a large patch due to spatial coherency. As shown in the proof of Lemma 1, the difference between the distance of two points $\mathbf{x}$ and $\mathbf{y}$ in the original space and its lower bound is proportional to $\sigma_{\mathbf{x}}\sigma_{\mathbf{y}}$. It suggests that we can improve the lower bound by reducing $\sigma_{\mathbf{x}}$ and $\sigma_{\mathbf{y}}$, which is often achieved by subdividing a patch based on the spatial locality.

The lower bound $\mathrm{LB}_k(\mathbf{x},\mathbf{y})$ consists of k lower bounds and we require the original data and the query should be embedded in $2k$-dimensional space. It incurs additional computational cost compared to 2-dimensional embedding based only on $\mathrm{LB}(\mathbf{x},\mathbf{y})$. However, each partition may still be very high-dimensional and slightly higher dimensional embedding—but significantly lower dimensional representation compared to the original data—is advantageous. Also, note that many data points are typically eliminated by the lower dimensional embeddings and that there remain few data to be embedded onto higher dimensional subspaces, statistically.

The pseudocode in Algorithm 2 summarizes our nearest neighbor search algorithm using the recursive partitioning.[1] The nested **for** loop handles the case that $\mathrm{LB}_{16}(\mathbf{x},\mathbf{y})$ is smaller than the current minimum distance; instead of computing $\mathrm{dist}(\mathbf{x},\mathbf{y})^2$ in $O(d)$ time, we gradually improve the bound by replacing $\mathrm{LB}(\mathbf{x}_{i,16},\mathbf{y}_{i,16})$ with $\mathrm{dist}(\mathbf{x}_{i,16},\mathbf{y}_{i,16})^2$ one by one until the bound becomes larger than or equal to the current minimum distance. Each iteration takes $O(d/16)$ time, so this **for** loop saves time if the bound reaches the current nearest neighbor before iterating all subdivisions. In practice for images or videos, we subdivide regions, as presented in Figure 2.

4. Relation to Linear Projection Algorithm

Our work is closely related to [9], where input vectors are projected onto a set of orthogonal basis vectors and eliminated based on the distance in each one-dimensional subspace. A critical difference between our algorithm and [9] is embedding strategy; our algorithm employs nonlinear embedding but [9] uses linear embedding. The main idea in [9] uses the following property:

$$\mathrm{dist}(\mathbf{x},\mathbf{y})^2 \ge (\mathbf{u}_i^{\mathrm{T}}\mathbf{x} - \mathbf{u}_i^{\mathrm{T}}\mathbf{y})^2, \quad (3)$$

where $\mathbf{u}_i^{\mathrm{T}}\mathbf{x}$ and $\mathbf{u}_i^{\mathrm{T}}\mathbf{y}$ are the projections of $\mathbf{x}$ and $\mathbf{y}$ onto a unit vector $\mathbf{u}_i$, respectively. It means the lower bound of the squared Euclidean distance between $\mathbf{x}$ and $\mathbf{y}$ is given by the distance in the projected space. Using this property, the lower bound of $\mathrm{dist}(\mathbf{x},\mathbf{y})^2$ can be improved progressively

[1] Note that we introduced LB_2 just for the convenience to describe our idea and did not use it in our algorithm as shown in Figure 2.

Algorithm 2 Improved Fast Nearest Neighbor Search

Pick a vector in $\mathcal{X}$ as the seed $\mathbf{x}^{\min}$
MINDIST $\leftarrow \mathrm{dist}(\mathbf{x}^{\min},\mathbf{y})^2$
for all $\mathbf{x} \in \mathcal{X}$ **do**
  if $\mathrm{LB}(\mathbf{x},\mathbf{y}) \ge$ MINDIST **then**
    continue
  end if
  if $\mathrm{LB}_4(\mathbf{x},\mathbf{y}) \ge$ MINDIST **then**
    continue
  end if
  if $\mathrm{LB}_{16}(\mathbf{x},\mathbf{y}) \ge$ MINDIST **then**
    continue
  end if
  BND $\leftarrow \mathrm{LB}_{16}(\mathbf{x},\mathbf{y})$
  for $i = 1 \rightarrow 16$ **do**
    BND $\leftarrow$ BND $+ \mathrm{dist}(\mathbf{x}_{i,16},\mathbf{y}_{i,16})^2 - \mathrm{LB}(\mathbf{x}_{i,16},\mathbf{y}_{i,16})$
    if BND $\ge$ MINDIST **then**
      break
    end if
  end for
  if BND $<$ MINDIST **then**
    $\mathbf{x}^{\min} \leftarrow \mathbf{x}$
    MINDIST $\leftarrow$ BND
  end if
end for
return $\{\mathbf{x}^{\min}$, MINDIST$\}$

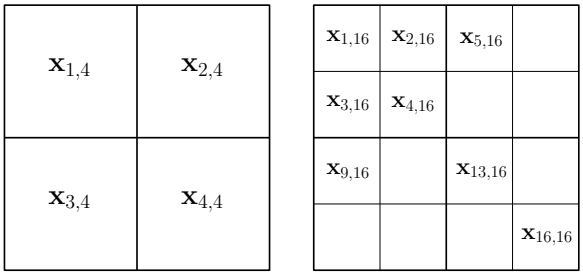


Figure 2: Recursive subdivision of a d-dimensional vector $\mathbf{x}$ into vectors in smaller dimensions.

by the projections of $\mathbf{x}$ and $\mathbf{y}$ onto additional orthogonal basis vectors as

$$\begin{aligned}\mathrm{LB}'_p &\equiv \sum_{j=1}^{p}(\mathbf{u}_j^{\mathrm{T}}\mathbf{x} - \mathbf{u}_j^{\mathrm{T}}\mathbf{y})^2 \\ &= \sum_{j=1}^{p-1}(\mathbf{u}_j^{\mathrm{T}}\mathbf{x} - \mathbf{u}_j^{\mathrm{T}}\mathbf{y})^2 + (\mathbf{u}_p^{\mathrm{T}}\mathbf{x} - \mathbf{u}_p^{\mathrm{T}}\mathbf{y})^2. \qquad (4)\end{aligned}$$

This implies that, at the pth projection, it updates its lower bound by adding the squared difference between $\mathbf{x}$ and $\mathbf{y}$ along their projections onto $\mathbf{u}_p$ to its previous lower bound LB'_{p-1}. For efficient computation of the projections, they employed the Walsh-Hadamard basis vectors, which are binary (with ± 1 elements) and mutually orthogonal [18, 19].

An issue in [9] is the absence of filtering criteria. They rely on the predefined threshold instead of the distance to the current nearest neighbor from the query. Depending on the threshold, many candidates may remain after all available projections or the nearest neighbor may be rejected at an early stage; threshold selection and nearest neighbor search are chicken-and-egg problems. It is possible to maintain the current nearest neighbor in their framework as in our algorithm. In this case, the linear projection approach is almost equivalent to the optimized brute-force search algorithm described in Algorithm 3.

Algorithm 3 Optimized Brute-Force Search

```
Pick a vector in 𝒳 as the seed x^min
MINDIST ← dist(x^min, y)^2
for all x ∈ 𝒳 do
  BND = 0
  for j = 1 → d do
    BND = BND + (x_j − y_j)^2
    if BND ≥ MINDIST then
      break
    end if
  end for
  if BND < MINDIST then
    x^min ← x
    MINDIST ← BND
  end if
end for
return {x^min, MINDIST}
```

Figure 3 illustrates the equivalence of the brute-force search and the linear projection algorithm. Suppose that the ith coordinate of a vector $\mathbf{x}$ is the value of its projection onto $\mathbf{u}_i$.[2] Then, the total computation for filtering with the first k projections is the same with that of the optimized brute-force search up to the first k dimensions; the computational costs of both strategies would be statistically equivalent.

[2]The choice of basis vectors are different from [9]. However, this choice is better than Walsh-Hadamard basis because we do not need to compute projections because they are already *given*.

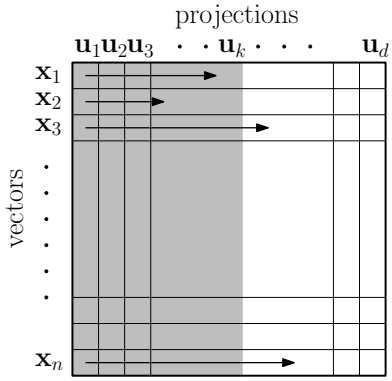


Figure 3: The equivalence of the optimized brute-force search and the linear projection algorithm.

5. Experiment

In this section, we first discuss an important implementation issue of our algorithm—seed selection. Then, we present the performance of our algorithm in several scenarios frequently observed in computer vision problems.

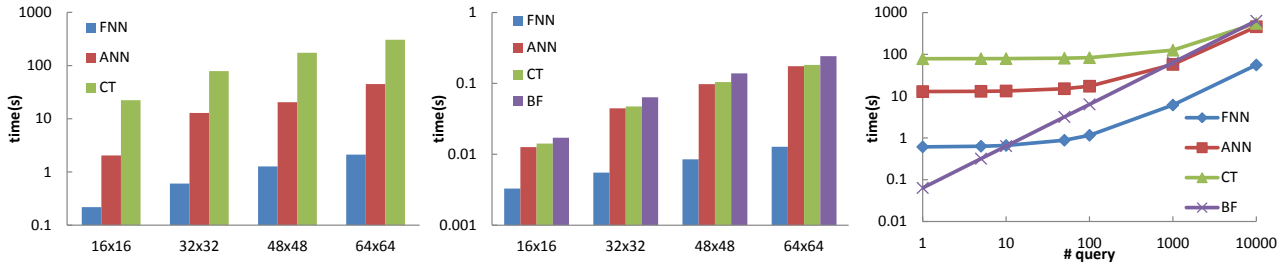
5.1. Seed selection

The performance of our algorithm depends on the ratio of the candidates rejected in the embedded space, and it is directly related to the quality of the *seed* in Algorithm 2. The best choice of the seed would be the point closest to query, but it is equivalent to the nearest neighbor search problem. Another naïve choice is to use a random point as the seed, but it may not be effective to eliminate nearest neighbor candidates.

In our implementation, we pick k vectors at random and use the one with the smallest Euclidean distance to query $\mathbf{y}$ as seed. The randomness improves the quality of seed and filtering performance statistically. Note that the first order statistic of a random sample is equal to its smallest value [7]. This implies that there are roughly n/k vectors whose Euclidean distances to $\mathbf{y}$ are at most the distance of the random seed to $\mathbf{y}$, and therefore quite a large number of vectors are expected to be rejected in average by filtering with the selected seed.

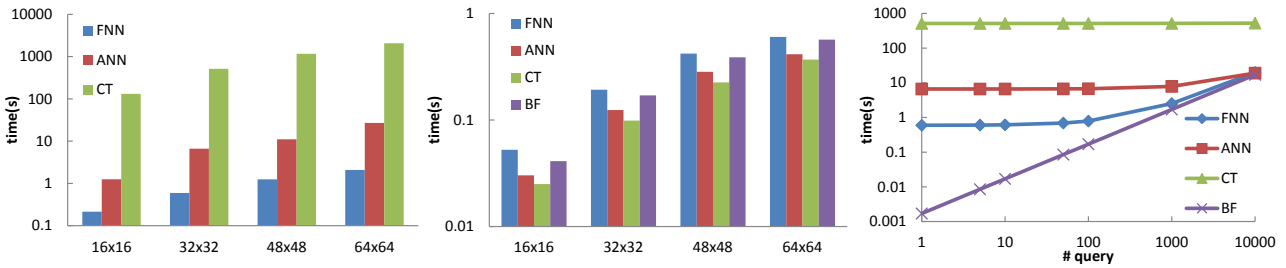
5.2. Nearest Neighbor Search

For the evaluation of our nearest neighbor search algorithm, we used two datasets—the CIFAR-10 and a synthetic dataset. The CIFAR dataset [15] consist of 6 (5 training and 1 test) batches of 10,000 color images of 32×32. 100 query images are randomly sampled from the test batch, and the



(a) Preprocessing (b) A query (c) Preprocessing and queries on 32×32 images

Figure 4: Performance for the exact nearest neighbor search on the CIFAR dataset. ($\log_{10}$ scale)



(a) Preprocessing (b) A query (c) Preprocessing and queries on 32×32 images

Figure 5: Performance for the exact nearest neighbor queries on a random dataset. ($\log_{10}$ scale)

running time for the nearest neighbor search is measured for each training batch. We resize images in the dataset to change the dimensionality of data. The synthetic dataset consists of 10,000 points of three different sizes, 16×16, 32×32 and 64×64, where the elements in each vector are randomly chosen integers in $[0, 255]$.

Four different algorithms are implemented and tested in our experiment—our algorithm (FNN) in Algorithm 2, the optimized Brute-Force search (BF) in Algorithm 3, the Approximate Nearest Neighbor search (ANN) [1, 16], and the Cover Tree (CT) [5, 4].

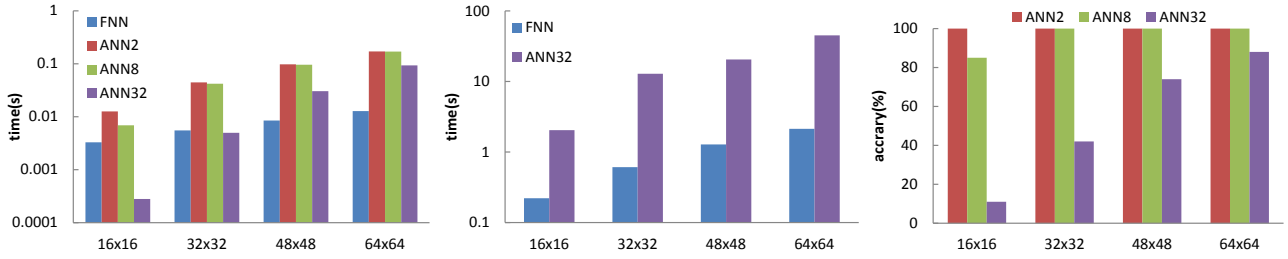
Exact nearest neighbor search. Figure 4 illustrates the performance of the four algorithms for the exact nearest neighbor search on the CIFAR dataset, where both preprocessing and query time are compared. Note that BF does not need preprocessing and ANN is set to return the exact nearest neighbor in this case. The preprocessing of FNN is very efficient—approximately 1 second at most, but ANN and CT require significant amount of time for preprocessing, which takes up to 100 times more than FNN. In terms of the query processing time, FNN outperforms others approximately by one order of magnitude while the query times of the other three algorithms are comparable.

For the synthetic dataset, FNN gains no benefit of fast filtering unfortunately, which suggests that filtering technique is more appropriate for natural images with spatial locality than random data. The performance of the four algorithms are all comparable; ANN and CT takes much more time in preprocessing, but is marginally faster in query processing, which is presented in Figure 5.

The results of FNN with random data are not as good as natural image data because the two factors to determine the lower bound of the squared Euclidean distance—the differences of two element means and standard deviations—become small statistically when the dimensionality is sufficiently high and each element in a vector is *iid*. The effect of the characteristics is actually found in Figure 5b, where query time is not improved since all the pixel values are obtained from the same uniform distribution independently. However, natural image and video data are not *iid* and the differences of the two means and standard deviations may not decrease even in high dimensional data. Also, note that our algorithm utilizes the property that the standard deviations of each subregion decreases by partitioning due to spatial locality of image and video data.

Approximate Nearest Neighbor Search. Our exact algorithm is also compared with the approximate nearest neighbor search algorithm based on ANN.[3] Figure 6 presents the

[3]BF is not straightforward to be implemented for approximate nearest neighbor search.



(a) A single query (b) Preprocessing and a query (c) Accuracy

Figure 6: Performance for the approximate nearest neighbor queries on the CIFAR dataset. ($\log_{10}$ scale)

performance of FNN and three versions of ANN—ANN2, 8, and ANN32—in the CIFAR dataset. ANNi returns a point whose distance is within i times of the optimal one. Note that all three approximate ANN algorithms including the exact version involve the exactly same preprocessing. The preprocessing of FNN is obviously much more efficient than ANN as illustrated in Figure 4a, and FNN is also faster than ANN in query processing except ANN32 (high approximation ratio) in a relatively low-dimensional data (16×16) as in Figure 6a. Since ANN32 is the faster than the other two versions, we only compared the preprocessing and query time of FNN and ANN32 in Figure 6b. As expected, FNN is much more efficient than ANN32, especially in high-dimensional cases. Figure 6c presents the accuracy of the approximate nearest neighbor search methods, where ANN2 always returns the exact solution over all dimensions in practice; the accuracy of ANN32 gradually improves with dimensionality increase and reaches around 80% in 64×64 images. It is probably because ANN is more effective in a high-dimensional space due to sparsity of data.

Memory requirement. The memory requirement is a critical issue to handle large-scale and high-dimensional data. While FNN needs only $O(n)$ memory space to store μ and σ values of n vectors in addition to $O(nd)$ memory space for the input, ANN and CT requires additional $O(nd)$ space to construct the tree data structures.

6. Conclusion

We proposed an efficient algorithm to compute the nearest neighbor by embedding the original data onto low dimensional nonlinear subspaces. Our algorithm maintains the lower bound of the squared Euclidean distance between a point in the dataset and the query using the distances in the subspaces in order to reject candidate points. Due to the effectiveness of the rejection strategy based on the low dimensional distances, our algorithm is much faster than the state-of-the-art nearest neighbor search algorithms. The performance of our algorithm is tested and verified in synthetic data and real images.

Acknowledgement

We give special thanks to Dr. Hyo-Sil Kim and Wanbin Son for their helpful discussion. This research was supported in part by Basic Science Research Program (2011-0005749) and in part by SRC-GAIA (2011-0030044) through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology.

References

[1] S. Arya, D. Mount, N. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM*, 45(6):891–923, 1998. 1, 6

[2] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of ACM*, 18(9):509–517, September 1975. 1

[3] S. Berchtold, D. A. Keim, and H. P. Kriegel. The x-tree: An index structure for high-dimensional data. In *Proc. VLDB '96*, pages 28–39, 1996. 1

[4] A. Beygelzimer, S. Kakade, and J. Langford. Cover tree implementation. `http://hunch.net/~jl/projects/cover_tree/cover_tree.html`. Online. 6

[5] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *Proc. ICML '06*, pages 97–104. ACM, 2006. 1, 6

[6] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In T. Darrell, P. Indyk, and G. Shakhnarovich, editors, *Proc. SoCG'04*, pages 253–262. ACM, 2004. 1

[7] H. A. David and H. N. Nagaraja. *Order Statistics (3rd edition)*. Wiley, 2003. 5

[8] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. SIGMOD '84*, pages 47–57. ACM, 1984. 1

[9] Y. Hel-Or and H. Hel-Or. Real-time pattern matching using projection kernels. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(9):1430–1445, 2005. 1, 2, 4, 5

[10] Y. Hwang and H. Ahn. Convergent bounds on the euclidean distance. In *Proc. NIPS '11*, 2011. 2

[11] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proc. STOC'98*, pages 604–613, 1998. 1

[12] H. V. Jagadish, B. C. Ooi, K.-L. Tan, C. Yu, and R. Zhang. iDistance: An adaptive B+-tree based indexing method for nearest neighbor search. *ACM Trans. Database Syst.*, 30(2):364–397, June 2005. 1

[13] D. R. Karger and M. Ruhl. Finding nearest neighbors in growth-restricted metrics. In *Proc. STOC'02*, pages 741–750, 2002. 1

[14] S. Korman and S. Avidan. Coherency sensitive hashing. In *Proc. ICCV'11*. IEEE, 2011. 1

[15] A. Krizhevsky. Learning multiple layers of features from tiny images. http://www.cs.toronto.edu/~kriz/cifar.html. Online. 5

[16] D. M. Mount and S. Arya. A library for approximate nearest neighbor searching. http://www.cs.umd.edu/~mount/ANN/. Online. 6

[17] W. Ouyang, F. Tombari, S. Mattoccia, L. Stefano, and W.-K. Cham. Performance evaluation of full search equivalent pattern matching algorithms. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2011. 2

[18] J. Shanks. Computation of the fast walsh-fourier transform. *IEEE Trans. Computers*, C-18(5):457–459, 1969. 1, 5

[19] D. Sundararajan and M. Ahmad. Fast computationa of the discrete walsh and hadamard transforms. *IEEE Trans. Image Processing*, 7(6):898–904, 1998. 1, 5

[20] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *Proc. NIPS'08*, pages 1753–1760, 2008. 1

A. Proof of Lemma 3

We want to show $\text{LB}(\mathbf{x},\mathbf{y}) - \text{LB}_2(\mathbf{x},\mathbf{y}) \leq 0$.

After expanding the inequality

$$d((\mu_{\mathbf{x}}-\mu_{\mathbf{y}})^2+(\sigma_x-\sigma_y)^2) - \sum_{i=1}^{2}\frac{d}{2}((\mu_{\mathbf{x}_{i,2}}-\mu_{\mathbf{y}_{i,2}})^2+(\sigma_{\mathbf{x}_{i,2}}-\sigma_{\mathbf{y}_{i,2}})^2) \leq 0. \quad (5)$$

By multiplying $4/d$ to both sides of the inequality, we are left with

$$4(\mu_{\mathbf{x}}-\mu_{\mathbf{y}})^2 - 2\sum_{i=1}^{2}(\mu_{\mathbf{x}_{i,2}}-\mu_{\mathbf{y}_{i,2}})^2 + 4(\sigma_x-\sigma_y)^2 - 2\sum_{i=1}^{2}(\sigma_{\mathbf{x}_{i,2}}-\sigma_{\mathbf{y}_{i,2}})^2 \leq 0. \quad (6)$$

By substituting $\mu_{\mathbf{x}}$ and $\mu_{\mathbf{y}}$ by $(\mu_{\mathbf{x}_{1,2}}+\mu_{\mathbf{x}_{2,2}})/2$ and $(\mu_{\mathbf{y}_{1,2}}+\mu_{\mathbf{y}_{2,2}})/2$, respectively, the first two terms in Eq. (6) are rearranged as follows:

$$-((\mu_{\mathbf{x}_{1,2}}-\mu_{\mathbf{x}_{2,2}})-(\mu_{\mathbf{y}_{1,2}}-\mu_{\mathbf{y}_{2,2}}))^2. \quad (7)$$

To expand the last two terms in Eq. (6), we use the following equation.

$$\begin{aligned}
&\sigma_{\mathbf{x}}^2 = \mu_{\mathbf{x}^2} - \mu_{\mathbf{x}}^2 \\
=&\frac{1}{2}(\mu_{\mathbf{x}_{1,2}^2}+\mu_{\mathbf{x}_{2,2}^2}) - \frac{1}{4}(\mu_{\mathbf{x}_{1,2}}+\mu_{\mathbf{x}_{2,2}})^2 \\
=&\frac{1}{4}(2\sigma_{\mathbf{x}_{1,2}^2}+2\sigma_{\mathbf{x}_{2,2}^2}+\mu_{\mathbf{x}_{1,2}}^2+\mu_{\mathbf{x}_{2,2}}^2-2\mu_{\mathbf{x}_{1,2}}\mu_{\mathbf{x}_{2,2}}) \\
=&\frac{1}{4}(2\sigma_{\mathbf{x}_{1,2}^2}+2\sigma_{\mathbf{x}_{2,2}^2}+A^2), \quad (8)
\end{aligned}$$

where $A = \mu_{\mathbf{x}_{1,2}} - \mu_{\mathbf{x}_{2,2}}$ and $B = \mu_{\mathbf{y}_{1,2}} - \mu_{\mathbf{y}_{2,2}}$. Now we expand the terms in the second line of Eq. (6) using Equation (8).

$$A^2 + B^2 - 8\sigma_{\mathbf{x}}\sigma_{\mathbf{y}} + 4\sum_{i=1}^{2}\sigma_{\mathbf{x}_{i,2}}\sigma_{\mathbf{y}_{i,2}}. \quad (9)$$

By using Eq. (7) and (9), Eq. (6) is rewritten as follows.

$$\begin{aligned}
&-(A-B)^2 + A^2 + B^2 - 8\sigma_{\mathbf{x}}\sigma_{\mathbf{y}} + 4\sum_{i=1}^{2}\sigma_{\mathbf{x}_{i,2}}\sigma_{\mathbf{y}_{i,2}} \\
&= 2AB - 8\sigma_{\mathbf{x}}\sigma_{\mathbf{y}} + 4\sum_{i=1}^{2}\sigma_{\mathbf{x}_{i,2}}\sigma_{\mathbf{y}_{i,2}} \leq 0. \quad (10)
\end{aligned}$$

Since σ values are always nonnegative, $\sum_{i=1}^{2}\sigma_{\mathbf{x}_{i,2}}\sigma_{\mathbf{y}_{i,2}} \geq 0$ and $\sigma_{\mathbf{x}}\sigma_{\mathbf{y}} \geq 0$. By Eq. (8), we have $(2AB)^2 \leq (8\sigma_{\mathbf{x}}\sigma_{\mathbf{y}})^2$, which implies $2AB - 8\sigma_{\mathbf{x}}\sigma_{\mathbf{y}} \leq 0$. Therefore, we show the following is at most zero.

$$\begin{aligned}
&(2\sum_{i=1}^{2}\sigma_{\mathbf{x}_{i,2}}\sigma_{\mathbf{y}_{i,2}})^2 - (AB-4\sigma_{\mathbf{x}}\sigma_{\mathbf{y}})^2 \\
=&(2\sum_{i=1}^{2}\sigma_{\mathbf{x}_{i,2}}\sigma_{\mathbf{y}_{i,2}})^2 - A^2B^2 - 16\sigma_{\mathbf{x}}^2\sigma_{\mathbf{y}}^2 + 8AB\sigma_{\mathbf{x}}\sigma_{\mathbf{y}} \\
=&-2A^2B^2 + 8AB\sigma_{\mathbf{x}}\sigma_{\mathbf{y}} \\
&-A^2(2\sigma_{\mathbf{y}_{1,2}}^2+2\sigma_{\mathbf{y}_{2,2}}^2+B^2) - A^2B^2 \\
&-B^2(2\sigma_{\mathbf{x}_{1,2}}^2+2\sigma_{\mathbf{x}_{2,2}}^2+A^2) - A^2B^2 \\
&-4\sigma_{\mathbf{x}_{1,2}}^2\sigma_{\mathbf{y}_{2,2}}^2 - 4\sigma_{\mathbf{x}_{2,2}}^2\sigma_{\mathbf{y}_{1,2}}^2 + 8\sigma_{\mathbf{x}_{1,2}}\sigma_{\mathbf{y}_{1,2}}\sigma_{\mathbf{x}_{2,2}}\sigma_{\mathbf{y}_{2,2}} \\
=&-4(A\sigma_{\mathbf{y}}-B\sigma_{\mathbf{x}})^2 - 4(\sigma_{\mathbf{x}_{1,2}}\sigma_{\mathbf{y}_{2,2}}-\sigma_{\mathbf{x}_{2,2}}\sigma_{\mathbf{y}_{1,2}})^2 \leq 0
\end{aligned}$$

The last two equalities are also derived from Eq. (8). Therefore, $\text{LB}(\mathbf{x},\mathbf{y}) - \text{LB}_2(\mathbf{x},\mathbf{y}) \leq 0$ holds.