This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2017.2748131, IEEE Transactions on Computers

Principal Component Analysis based Filtering for Scalable, High Precision k-NN Search

Huan Feng, David Eyers, Steven Mills, Yongwei Wu, *Member, IEEE,* and Zhiyi Huang,

Abstract—Approximate k Nearest Neighbours (AkNN) search is widely used in domains such as computer vision and machine learning. However, AkNN search in high-dimensional datasets does not scale well on multicore platforms, due to its large memory footprint. Parallel AkNN search using space subdivision for filtering helps reduce the memory footprint, but its loss of precision is unstable. In this paper, we propose a new data filtering method—PCAF—for parallel AkNN search based on principal component analysis. PCAF improves on previous methods, demonstrating sustained, high scalability for a wide range of high-dimensional datasets on both Intel and AMD multicore platforms. Moreover, PCAF maintains highly precise AkNN search results.

Index Terms—K nearest neighbours, approximate knn search, parallel algorithms, multicore, principal component analysis, data filtering, scalability.

1 Introduction

Wide use of k Nearest Neighbours (k-NN) search is made in domains such as bioinformatics [1], data analysis [2], machine learning [3], computer vision [4] and handwriting recognition [5]. Given query data points, k-NN finds the k data items within a database that are most similar to the query data, where the similarity is often measured by Euclidean Distance. In general, a data point $\mathbf{p}$ can be defined as a D-dimensional vector, $\mathbf{p} = \begin{bmatrix} e_1 & e_2 & \dots & e_D \end{bmatrix}^\top$. The database is a set of N data points, $\mathcal{R} = \{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_N\}$. Based on the definitions, the k-NN problem can be formally described as: given a query point $\mathbf{q}$, find the k data points in $\mathcal{R}$ that have the shortest (Euclidean) distances to $\mathbf{q}$.

To address the rapidly increasing amounts of data being produced for processing, many Approximate k Nearest Neighbours (AkNN) algorithms have been proposed [6], [7], [8], [9], [10], [11]. Instead of returning the actual k-NN, they return k results that are highly likely to be the k-NN. Although AkNN algorithms are significantly more efficient, search precision becomes a concern [12], [13], [14].

There are two main strategies in AkNN for finding approximate nearest neighbours: *data selection* and *data filtering*. The data selection strategy tries to find candidate points that are most likely to be the precise k nearest neighbours. Most AkNN algorithms adopt this strategy [6], [7], [9]. However, this strategy incurs a large memory footprint and requires a large number of random memory accesses. This leads to many cache misses, and therefore poor scalability on multicore systems [8], [15], [16].

The data filtering strategy [16] instead excludes unlikely data points based on distance estimation between the query and the data points. If they have a high filtering rate, much computation and many memory accesses can be avoided. Typically, the scalability of AkNN can be greatly improved on multicore systems by using a filtering strategy.

Subspace Clustering for Filtering (SCF) [8] is a state-of-the-art approach to AkNN that uses data filtering. It greatly improves the scalability of AkNN algorithms. However, its search precision is unstable and depends on the nature of the datasets. We will discuss this challenge in detail in the next section.

In this paper, we propose a parallel AkNN algorithm called PCAF which uses Principal Component Analysis (PCA) [17] to estimate the rank of distance between the query and the data points. PCAF uses data filtering to exclude those data points that are not likely to be k-NN results according to the PCA estimation. It has high scalability on multicore systems with stable, high search precision on high-dimensional datasets (our experiments include datasets with up to 960 dimensions).

The remainder of this paper is organised as follows. Section 2 describes the motivation of our research. Section 3 presents our PCAF algorithm, and Section 4 describes its technical implementation details. Section 5 provides a performance evaluation against four widely-used k-NN algorithms. Section 6 assesses the impact of different parameter selections on precision and performance, and gives appropriate choices of parameters for six example datasets. Section 7 discusses related research work. Finally, Section 8 summarises the contributions of this paper.

2 Motivation

As discussed above, data filtering in AkNN greatly improves its parallel performance on multicore systems. We have previously proposed a parallel AkNN algorithm called SCF using data filtering to exclude unlikely k-NN points. Before searching, SCF must build an index over the dataset, as needed in all AkNN algorithms. SCF divides the data points into a number of subspaces with low-dimensionality

- *H. Feng and Y. Wu are with the Department of Computer Science and Technology, Tsinghua National Laboratory for Information Science and Technology (TNLIST), Tsinghua University, Beijing 100084, China; Research Institute of Tsinghua University in Shenzhen, Shenzhen 518057, China. E-mail: see http://madsys.cs.tsinghua.edu.cn/*
- *D. Eyers, S. Mills and Z. Huang are with the Department of Computer Science, University of Otago, New Zealand.*

Manuscript received December 05, 2016; revised August 3, 2017.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2017.2748131, IEEE Transactions on Computers

TABLE 1
The rank estimation using SCF with different formations of subspaces and PCAF in an example.

(a) A 4-dimensional example with query, q, and points, A, B, C, D. RED and the exact rank of points are listed.

	e_1	e_2	e_3	e_4	RED	Rank
q	1	1	1	1	-	-
A	1	1	14	15	19.10	4
B	2	3	7	11	11.87	2
C	4	5	5	5	7.55	1
D	5	6	12	10	14.90	3

(b) EED and rank of using SCF with different formations of subspaces: $[e_1\ e_2]$ $[e_3\ e_4]$ and $[e_1\ e_3]$ $[e_2\ e_4]$.

SCF	$[e_1\ e_2]$ $[e_3\ e_4]$		$[e_1\ e_3]$ $[e_2\ e_4]$	
	EED	Rank	EED	Rank
A	16.66	3	17.12	4
B	8.67	1	13.19	2
C	10.32	2	9.57	1
D	17.57	4	14.51	3

(c) EED and rank using PCAF with only 1 principal component. p_1 lists the projections of all of the points

PCAF	p_1	EED	Rank
q	-10.77	-	-
A	7.29	18.06	4
B	-0.73	10.04	2
C	-7.05	3.72	1
D	0.48	11.25	3

in order to alleviate the problem of the *curse of dimensionality* [2], [18]. Then, in each subspace, SCF uses k-means [9], [19] clustering to divide the points into clusters. The centre of each cluster is used to estimate the distance between the query and the points of the cluster in the subspace. Finally the distance between the query and any given point in the original high-dimensional space is estimated by summing up the distance between the query and the point's projection within each subspace.

Table 1a gives an example of k-NN search with one query and four points in a 4-dimensional space. The Real Euclidean Distance (RED) and the rank based on it are also listed in the table. We use only one query and four points here, but in real applications, there will typically be tens of thousands of queries and data points in each pair of images and thousands of images to be searched and matched pair by pair. This domain thus needs high-performance parallel computing if results are to be calculated rapidly.

To demonstrate how SCF works, suppose SCF divides the original space into two subspaces: the first two dimensions form one subspace and the remaining two form the other. In each subspace, two clusters are formed and the centre of each cluster is used to estimate the distance between the query and the points of the cluster. Based on that, the Estimated Euclidean Distance (EED), which is calculated by summing up distances from the query to each group centres, is shown in Table 1b (for more details about how the EED is calculated refer to [8]).

From the table, the rank of the points is B, C, A, D according to the EED. However, according to the RED in Table 1a, the rank is C, B, D, A. Based on the estimation, if only one nearest neighbour is requested, in this case, the precision of the result using SCF is 0—the matched point is not actually the closest.

However, if we change the partitioning of dimensions for the subspaces, SCF may produce the correct results. For example, we instead choose the first and third dimensions to form the first subspace and the remainder to form the second. Likewise we create two clusters in each subspace. According to the EED calculated by SCF listed in Table 1b, the rank of the points is C, B, D, A, which is exactly the same as their rank in RED. Therefore, the precision of each search of k-NN for $k \in [1, 4]$ using SCF would be 100%.

The above example demonstrates that the precision of SCF is seriously affected by how the subspaces are formed. Forming the optimal subspaces to achieve the highest precision is data dependent. It is difficult to determine how much it will depend on the nature of a given dataset, and thus may vary significantly between applications.

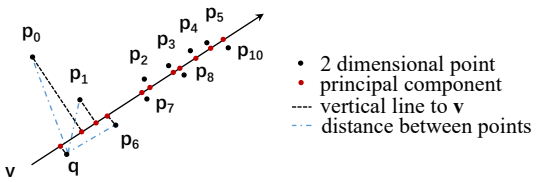


Fig. 1. The first principal direction $\mathbf{v}$ of the dataset having $\mathbf{p}_i, i \in [0, 10]$, where all points lie close to $\mathbf{v}$ except $\mathbf{p}_0$

In the following section, we propose a PCA-based filtering method called PCAF for AkNN search. It has the same advantages of using the data filtering strategy: high scalability, small memory footprint, and reduced computational overhead. More importantly, compared with SCF, PCAF has stable, higher precision k-NN results due to its accurate rank estimation of Euclidean Distance using the principal components of the dataset points.

3 PCA-based Filtering (PCAF)

Principal Component Analysis (PCA) is a popular algorithm used to reduce dimensionality, that can alleviate the *curse of dimensionality* in some contexts. It uses an orthogonal transformation, $\varphi : V \rightarrow V'$, to convert a set of data values of possibly correlated variables denoted as $\mathbf{p}$ into a set of data values of linearly uncorrelated variables $\varphi(\mathbf{p})$ called *principal components*. The D-dimensional space V is the original space containing $\mathbf{p}$, while V' is called the *PCA space* holding the principal components $\varphi(\mathbf{p})$—the projections of $\mathbf{p}$ in V'. According to the definition of orthogonal transformations, it preserves distance in Euclidean space.

$$\begin{aligned} \forall \mathbf{p}_i, \mathbf{p}_j \in V \rightarrow \exists \varphi(\mathbf{p}_i), \varphi(\mathbf{p}_j) \in V' \text{and} \\ \|\varphi(\mathbf{p}_i), \varphi(\mathbf{p}_j)\| = \|\mathbf{p}_i, \mathbf{p}_j\| \\ \text{where } \varphi(\mathbf{p}) = \mathbb{U}^\top \cdot \mathbf{p} \end{aligned} \quad (1)$$

So that the rank of distance to a query in V is maintained in the D-dimensional PCA space V'.

For a high-dimensional dataset, if the correlation between the dimensions is strong, the information of the dataset can be represented by a small number of principal components, which taken together have lower dimensionality. Using this small set of principal components denoted as $\tilde{\varphi}(\mathbf{p})$, the data set can be projected into a lower-dimensional space while preserving most of the original information *i.e.* the distance rank between $\forall \mathbf{p} \in V$. The projection $\tilde{\varphi}(\mathbf{p})$ is calculated as follows, where $\mathbb{U}$ is split into the submatrix $\tilde{\mathbb{U}}$ that we care about, and unused columns $\mathbb{A}$,

$$\tilde{\varphi}(\mathbf{p}) = \tilde{\mathbb{U}}^\top \cdot \mathbf{p}, \ \mathbb{U} = [\tilde{\mathbb{U}} \mid \mathbb{A}] \quad (2)$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2017.2748131, IEEE Transactions on Computers

We use Figure 1 to illustrate the distance rank relationship between V and $\tilde{\varphi}(\mathbf{p})$. We suppose there is a group of points $\mathbf{p}_i, i \in [1, 10]$ in a 2-dimensional space V. The first principal direction found by using PCA is represented by the arrow-pointed vector $\mathbf{v}$. Since most of the points lie close to $\mathbf{v}$, the distance rank of the points $\mathbf{p}_i, i \in [1, 10]$ is preserved in the principal component $\tilde{\varphi}(\mathbf{p})$—the projection of the points on $\mathbf{v}$, in the PCA space. As shown in Figure 1, except for $\mathbf{p}_0$, the distances of all points to $\mathbf{q}$ have the same rank in both V and the PCA space. More detailed theoretical analysis is beyond the scope of this paper and can be found in [20], [21], [22]. Based on this property of PCA, PCAF uses the projections/principal components to estimate the rank of distances between the query and the points. Note that different from most AkNN algorithms, what we estimate is the rank of distances instead of the real distances, because the distances calculated with projections are in a transformed low-dimensional space, which is different from the original, high-dimensional space. From Table 1c, we can see that the rank of distances between the query and the points in the 1-dimensional PCA space presented by the principal components p_1 is C, B, D, A, which is exactly the same as the rank of those in the original space.

In many real-world domains, the dimensions are more or less correlated [23]. For example, SIFT features [24], which are popular in computer vision applications, have many dimensions that are correlated with each other [25], [26], [27]. From our experiments, often we only need around 10 dimensions in the PCA space projected from the space of hundreds of dimensions. This shows us that we could possibly use very low-dimensional PCA-converted points' projections to accurately estimate the rank of distances between the query and the points. This approach can save a large amount of computation by calculating the distances in a very low-dimensional surrogate space.

In PCAF, before searching, PCA is applied to the centred dataset to find the principal axes in the PCA space. Then, the points are projected into the surrogate PCA space, and the query is projected into the same PCA space. PCAF maintains two heaps for the current k nearest neighbours. The *main heap* contains the current k nearest neighbours ranked by the distance of the original space. The *filter heap* contains the current k nearest neighbours ranked by the distance of the PCA space.

During search, the distance between a point and the query in the PCA space is calculated first. If the distance is larger than the largest distance in the filter heap, we simply drop the point as it is unlikely to be one of the k nearest neighbours. In this way, we can filter out many points with a low computational overhead. For example, in Figure 1, for 2-NN search in the projected PCA space, 9 points ($\mathbf{p}_i, i \in [2, 10]$) out of 11 will be filtered.

However, as the reader can notice, the false 2-NN result $\mathbf{p}_0$ is kept as a candidate but one true 2-NN $\mathbf{p_6}$ is incorrectly filtered. The reason is that the difference between the query $\mathbf{q}$ and $\mathbf{p}_0$ mainly lies along the other directions rather than the principal directions. Therefore, the projection of $\mathbf{p}_0$ to the principal direction is very close to the projection of $\mathbf{q}$. Fortunately, according to the principle of PCA, statistically most points should lie around the principal directions and the cases of $\mathbf{p}_0$ are very rare [20]. To counteract the effect of the rare cases of $\mathbf{p}_0$, PCAF uses a scale factor to enlarge the selection scope of k-NN in the PCA space and verifies the selected potential k-NN points using the real distances. That is, the filter heap is larger than the main heap and when the distance is smaller than the largest distance in the filter heap, PCAF calculates the distance between the point and the query in the original space, V. If the distance is larger than the largest distance in the main heap, we drop the point; otherwise, the distance of the point is inserted into the main heap and the corresponding distance in the PCA space is inserted into the filter heap. In this way, points like $\mathbf{p}_0$ are excluded while the true candidates are included. After each point is processed as described above, the k nearest neighbours are in both heaps.

In PCAF, there are two overheads that are related to PCA. The first is the PCA transformation applied to the dataset, which projects the points into the PCA space. This is a one-off cost, and is similar to the overhead of building indices in other AkNN algorithms according to our experiments. Moreover, if necessary, this PCA process could be parallelised to further reduce the overhead, as many parallel PCA algorithms have been proposed already [28], [29], [30].

The second overhead is the projection of the query into the PCA space. It involves a multiplication between a $d \times D$ matrix and a vector of size D, where D is the dimensionality of the original space and d is the dimensionality of the PCA space. This one-off cost is amortised over the distance computation against tens of thousands of points, as the same projected query will be reused by each point in the dataset. In order to simulate a real k-NN problem, we execute the query projection process as part of the searching in the current algorithm. Thus our performance evaluation includes the aforementioned overhead. It is worth noting that, if necessary, this overhead could be preprocessed in most practical applications and easily parallelised by using multiple CPU cores or SIMD floating point units (*e.g.* SSE), but it is a relatively small cost in any case.

The advantages of PCAF can be summarised as follows.

- It replaces the distance calculations between points in a high-dimensional space with the calculations in a surrogate low-dimensional space. The computational overhead is substantially reduced.
- The memory footprint is greatly reduced as only the low-dimensional projections are accessed most of time. This is extremely helpful for multicore systems that suffer from memory bottlenecks.
- The precision of k-NN results is significantly improved compared to other AkNN algorithms due to the use of principal components for rank estimation.

4 Implementation of PCAF

The main idea of PCAF is to use the distance rank in the surrogate PCA space to filter out the points that are unlikely to be in the k-NN set. In this section, we will discuss: (i) the rank estimation in the PCA space, (ii) the filtering algorithm in detail, and (iii) our fine-grained data parallelism in PCAF.

Note that, like other AkNN algorithms, to reduce computational overhead, we use the squared Euclidean distance, $\|\mathbf{p}_i - \mathbf{p}_j\|^2$, to measure the distance between two points $\mathbf{p}_i$ and $\mathbf{p}_j$ in the rest of the paper.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2017.2748131, IEEE Transactions on Computers

4.1 Rank Estimation with PCA

We use the Singular Value Decomposition (SVD) [31] to find the principal components of the dataset after it is centred. The principal axes in the PCA space are represented by a $d \times D$ matrix denoted as $\tilde{\mathbb{U}}^\top$, where D is the dimensionality of the original space, and $d \ll D$ is the dimensionality of the PCA space or the number of principal components. Having computed $\tilde{\mathbb{U}}^\top$ for the dataset $\mathcal{R} = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_N\}$, we need to project everything into the PCA space using

$$\begin{aligned}\mathcal{R}' &= \{\mathbf{p}'_1, \ldots, \mathbf{p}'_N\} \\ &= \{\tilde{\mathbb{U}}^\top(\mathbf{p}_i - \bar{\mathbf{p}}), \ldots, \tilde{\mathbb{U}}^\top(\mathbf{p}_N - \bar{\mathbf{p}})\} \quad (3)\\ \mathbf{q}' &= \tilde{\mathbb{U}}^\top(\mathbf{q} - \bar{\mathbf{p}}), \quad (4)\end{aligned}$$

where $\bar{\mathbf{p}}$ is the mean of the points in $\mathcal{R}$, that is $\bar{\mathbf{p}} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{p}_i$. These processes can be accelerated through parallel computing since either parallelisation of SVD or matrix-vector structured multiplication has been well-studied [28], [32], [33].

As $\tilde{\mathbb{U}}^\top$ and $\mathcal{R}'$ are used by all queries, the computation of these two matrixes is one-off preprocessing. Although the complexity of this overhead is $O(D^3)$, other AkNN algorithms have similar overhead for building search indices. According to our tests, using a single core of the Intel Xeon processor in our experimental environment, this one-off time overhead of PCAF is between 0.01s and 4.5s. This time can be further shortened if only necessary components are computed during the SVD decomposition process [34]. In contrast, the index building overhead of other AkNN algorithms is between 0.01s and 70.5s depending on precision (for detailed experimental results refer to [35]). However, this one-off overhead is amortised over large numbers (thousands) of queries.

After the projection, the distance in the PCA space between the query projection $\mathbf{q}'$ and each projected point, $\mathbf{p}'$ in set $\mathcal{R}'$ is calculated. This distance is used to rank the points using the filtering method described in the next section.

4.2 Filtering Method

Algorithm 1 gives the detailed description of data filtering in PCAF. For each point $\mathbf{p}_i$, PCAF first calculates the distance between the corresponding projected point, $\mathbf{p}'_i$, and the projected query $\mathbf{q}'$. If this distance, δ' is smaller than the maximum distance in $heap'$, then δ, the distance between $\mathbf{p}_i$ and $\mathbf{q}$, is calculated. If δ is smaller than the maximum distance in $heap$, then δ is inserted into $heap$, and δ' is inserted into $heap'$. The final k-NN results are in $heap$ after the above process is repeated for each point in the dataset $\mathcal{R}$. Note that, in Algorithm 1, $m \geq 1$ is used to adjust the size of $heap'$ to accommodate slight errors in the ranking under the principal axes $\tilde{\mathbb{U}}^\top$. A value of 2 is enough for most cases, and incurs a negligible overhead. We will elaborate on this in Section 6.2.

4.3 Data parallelism

PCAF is particularly parallelisable since the there is no dependence in the search of k-NN within each query and the points can be searched in parallel for the same query. This is different from other AkNN algorithms like a kd-tree [9] where each query has to retrieve the index sequentially.

Algorithm 1 PCAF data filtering

Input: $\mathcal{R} = \{\mathbf{p}_1, \ldots, \mathbf{p}_N\}$: set of points
Input: $\mathcal{R}' = \{\mathbf{p}'_1, \ldots, \mathbf{p}'_N\}$: set of projected points
Input: $\mathbf{q}$: query point
Input: $\mathbf{q}'$: query point's projection
Input: k: number of nearest required
Input: m: heap scaling factor
Output: $heap$: contains the k-NN results

```
Initialise each element of heap of size k with ∞
Initialise each element of the temporary filter heap heap' of size km with ∞
for all i ∈ [1, N] do
    δ' ← ‖q' − p'_i‖²
    if δ' < heap'.max then
        δ ← ‖q − p_i‖²
        if δ < heap.max then
            heap'.insert(δ')
            heap.insert(δ)
        end if
    end if
end for
return heap with the k nearest points
```

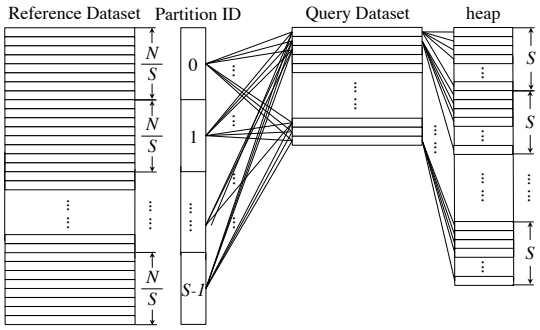


Fig. 2. Fine-grained parallel implementation based on data partitioning. Each task searches a set of $\frac{N}{S}$ points and maintains its own k-NN results in a heap.

Though the retrieval of the index within a query could be parallelisable, this complex parallel algorithm does not appear in available in k-NN implementations.

In PCAF, the projections in the PCA space are divided into S subsets, which are searched in parallel by threads using the same query as shown in Figure 2. In the figure, each task works on a subset of projections and maintains its own heaps. As the sizes of the heaps are very small (2 and 4 for most image processing applications), the extra space overhead has no noticeable impact on the performance but the support of fine-grained parallelism allows PCAF to run with high performance.

After the k-NN results are obtained from each subset, the final k-NN results are computed using a simple selection algorithm, such as the k-max heap used in our experiments.

4.4 Time and Space Complexity

The preprocessing of PCAF includes four steps. The first is centring the dataset $\mathcal{R}$, which also produces the mean vector $\bar{\mathbf{p}}$ for later projection of points. The time complexity of this step is $O(ND)$. Secondly, computing the covariance matrix has a time complexity of $O(ND^2)$. Then, SVD must

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2017.2748131, IEEE Transactions on Computers

be applied on the covariance matrix to obtain the singular values Σ and $\mathbb{U}$ matrix. Σ helps to compute the variance retained in the PCA space. The SVD process has $O(D^3)$ cost. Finally, generating $\tilde{\mathbb{U}}$, which is a submatrix consisting of the first d columns in $\mathbb{U}$, and projecting all the points in the dataset into the PCA space using $\tilde{\mathbb{U}}^\top$, costs $O(N)$. As the magnitude of N dominates the other parameters, the time complexity for preprocessing can be simplified to $O(N)$.

During the searching, PCAF takes little space to store the $d \times D$ projection matrix $\tilde{\mathbb{U}}^\top$, and D-vector $\bar{\mathbf{p}}$. During runtime, storing the projection of points takes $O(Nd)$ space, and the query $O(d)$ space. As the magnitude of N dominates the other parameters, the space complexity for PCAF can be simplified to $O(N)$.

PCAF can save time by reducing the number of distance calculations. Supposing the time for distance computation with all points for each query in the original (D-dimensional) space is T_D, and the filtering rate of excluded reference features is $F \in [0, 1)$. Theoretically PCAF takes $\left(1 - F + \frac{d}{D}\right) T_D$ time for distance computation. According to our experimental results, d is much smaller than D, and $F > 95\%$ in most cases. The extra cost for query projection during searching is only $O(dD)$. PCAF is very efficient and avoids many unnecessary distance computations.

We will illustrate detailed performance comparison with other k-NN algorithms in Section 5. Also note that by adjusting d, m (heap scaling) and S (the number of data subsets) in PCAF, the performance and precision changes. We will discuss the impact of each parameter on performance and precision in Section 6.

5 Evaluation

In this section, we evaluate the performance of our method against a brute-force k-NN algorithm, two state-of-the-art data selection AkNN algorithms and a data filtering AkNN algorithm on one synthetic dataset, five real-world datasets and a very large benchmark. The performance improvements attained on two different multicore platforms are analysed. All experiments are executed at least three times to make sure that stable results are obtained.

5.1 Experimental Setup

5.1.1 Multicore platforms

Two multicore platforms are used in our evaluations:

a) Intel16: Intel(R) Xeon(R) CPU E5-2665, 8 cores $\times$ 2 sockets @ 2.40 GHz, 20 MiB L3 shared cache, 128 GiB DDR3 (1600 MHz) memory, icc-14.0 compiler.
b) AMD64: AMD Opteron Processor 6276, 16 cores $\times$ 4 sockets @ 2.3 GHz, 16 MiB L3 shared cache, 512 GiB DDR3 (1600 MHz) memory, gcc-4.8 compiler.

5.1.2 Algorithms

We compare our algorithm with the four algorithms below:

a) Brute-force (BF): searches k-NN exhaustively in the whole database and gives the accurate k-NN results.
b) Randomized kd-trees (RKD): an efficient variant of the popular kd-tree algorithm [36]. Multiple trees are built as its index structure. During searching, it traverses these kd-trees and puts promising candidate nodes in a queue for distance calculations. The k-NN results contained within these nodes are the approximate k-NN results.
c) Random Ball Cover (RBC): a state-of-the-art scalable k-NN algorithm for multicore platforms [6]. First, it randomly chooses several representative points to represent a number of subsets of size s, each of which contains s points that are nearest to one representative. To produce the AkNN results, it finds the nearest representatives and then searches within those subsets using BF.
d) Subspace Clustering for Filtering (SCF): a recent algorithm that implements a data filtering strategy within the k-NN search problem [8], as discussed before.

We implement BF and RKD using the FLANN library [37], which provides fast AkNN search functionality for computer vision related tasks. The implementations of RBC and SCF are taken from the open source code provided in previous work. The parallelisation is carried out by using OpenMP. Note that all of our code uses Eigen [38] to perform matrix algebra. Eigen is a popular C++ library for linear algebra, matrix and vector operations, numerical solvers and related algorithms. In particular, the SVD decomposition in PCAF is adopted from Eigen and is a two-sided Jacobi R-SVD decomposition [34]. Although this Jacobi R-SVD is slower than the bidiagonalizing SVD algorithm [39], it ensures optimal reliability and accuracy. All of our code is available from our repository [35] on GitHub.

5.1.3 Datasets

The test datasets listed in Table 2 are used to evaluate the performance of the aforementioned algorithms. Among them, four are real-world datasets from the fields of computer vision and machine learning. "Digits" contains handwriting digits that have been size-normalised and centred into an 8×8 image. "SIFT" and "GIST" are generated from real images by extracting SIFT [24] and GIST [40] descriptors, which are two of the most widely used image feature detectors in computer vision [41]. "HAR" contains sensor readings related to 30 real-world subjects performing daily living activities, which are recorded by waist-mounted, embedded sensors. The others are artificial datasets: "Random" contains points that are chosen from a uniform distribution, and "Madelon" contains data points grouped in 32 clusters placed on the vertices of a five-dimensional hypercube.

TABLE 2
Overview of test datasets.

Name	Size (MB)	Dimensionality	Number of points	Number of queries
Digits [42]	0.93	64	3823	1797
Random	12.21	128	25000	7500
SIFT	27.38	128	56074	9929
Madelon [43]	3.81	500	2000	1800
GIST	5.86	512	3000	1000
HAR [44]	15.73	561	7352	2947

We separately analyse a dataset called "GIST1M" [12] in Section 5.6 to further evaluate the performance of the algorithms for a 'big data' use case on multicore platforms.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2017.2748131, IEEE Transactions on Computers

The dataset is 3.8 GB in size and contains one million 960-dimensional points and 1,000 queries extracted from a variety of images by using global colour GIST descriptors [40].

5.1.4 Parameter Settings

In order to find the performance bounds of each algorithm, we conduct as many scenarios as possible by adjusting parameters. We use $X \in [start \dots end, \Delta step]$ to denote that parameter X ranges from value $start$ to value end with an increase of $step$.

The RKD algorithm has two important parameters. The number of *trees* to build for indices greatly affects both indexing time and searching precision. The *checks* parameter represents the number of neighbouring buckets that should be checked during the search. As these parameters are increased, search time increases, but the results will be more precise [9]. We set *trees* as 4, 8, 16, 32, 64 or 128 and tested *checks* $\in [128 \dots 5120, \Delta 128]$.

The number of randomly chosen representatives in RBC positively affects the precision and negatively affects the search time [6]. Suppose $n = \lfloor \sqrt{N} \rfloor$, then the number of representatives is chosen from $[n \dots N, \Delta n]$.

For SCF, the number of *subspaces* and *clusters* are important parameters [8] and are set within $[4 \dots 64, \Delta 4]$ and $[8 \dots 32, \Delta 8]$ respectively.

The dimensionality of the PCA space, d, in PCAF is determined by the percentage of information/variance in the dataset that is retained in the PCA space. The lower and upper bounds of d listed in Table 3 are determined by the cases when 50% and 90% of the variance is retained, respectively. The heap scaling factor, m, is set from 1 to 5, while the datasets are partitioned into S parts, where S is set either 1, 2, 4, 8 or 16.

TABLE 3
The setting of d is chosen starting from the derived value when 50% of the variance in the dataset that is retained to the 90% case with an increase of $step$.

	Digits	Random	SIFT	Madelon	GIST	HAR
50%	5	61	10	53	8	1
90%	21	114	58	223	67	34
$step$	$\Delta 1$	$\Delta 1$	$\Delta 1$	$\Delta 5$	$\Delta 1$	$\Delta 1$

5.1.5 Evaluation Metrics

As a baseline for performance we use the brute-force (BF) method that checks the query, $\mathbf{q}$, against each of the points $\mathbf{p}_i$. Although computationally expensive, this method is guaranteed to return the exact k-NN results. We use three metrics to evaluate the parallel performance, and two more metrics to evaluate the filtering effectiveness of each data filtering algorithm:

a) Time: the total time used for searching.
b) Improvement: the search time of the exact BF solution divided by the search time of the AkNN algorithm.
c) Speedup: the sequential search time of an algorithm divided by its parallel search time. It measures the scalability.
d) Precision: the percentage of k-NN results that are correctly found.
e) Filtering rate, F: the percentage of points that are excluded. It is used to help measure the effectiveness of the filtering method.

5.2 Comparison with Brute-force

Brute-force (BF) is the traditional and straightforward implementation of a k-NN algorithm that provides accurate results. PCAF shows a great enhancement over BF in both speed improvement and scalability when exact k-NN results are retrieved.

5.2.1 Performance Improvement

In this section, we evaluate the parallel performance improvement when using all available cores. The improvements that PCAF can achieve over BF on various datasets are listed in Table 4.

TABLE 4
Performance improvement compared with BF search time (in seconds) after applying PCAF to each dataset on each platform using all available cores without losing precision.

Platform	Digits	Random	SIFT	Madelon	GIST	HAR
Intel16	2.47	1.07	3.66	1.99	5.85	2.25
AMD64	2.54	3.51	10.29	1.95	6.76	7.95

TABLE 5
Filtering rate, F, and the required PCA space dimension, d, for PCAF to produce exact k-NN results for each dataset.

	Digits	Random	SIFT	Madelon	GIST	HAR
$F(\%)$	95.27	94.70	98.60	87.53	96.81	62.93
d	5	90	15	53	8	24

The improvement mainly comes from the reduction in distance computations. The complexity of distance computation in BF is $O(ND)$, but in PCAF it is $O((1-F)ND) + O(Nd)$, where F is the filtering rate of excluded points for distance computation, and d is the dimensionality in PCA space. From Table 5, we can observe a high F and a small d in most of the datasets, which allows PCAF to significantly reduce the computation needed, and thus provides a considerable speed improvement.

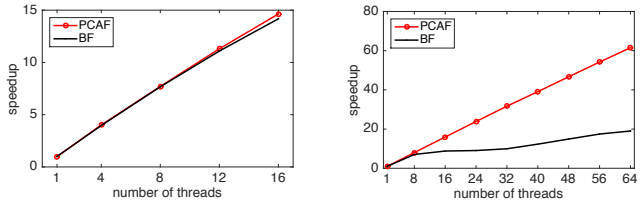
The improvement varies between different datasets because the amount of computation reduction varies. Take, for example, the datasets "SIFT" and "Random", which both consist of 128-dimensional data points. The dimensionality is greatly reduced in "SIFT": a 15-dimensional PCA space is sufficient to preserve most of the information from the original 128-dimensional space. However for "Random", 90 principal components have to be used for projections. Although the filtering rates between "SIFT" (98.60%) and "Random" (94.70%) are similar, the search time is significantly affected by the cost of the rank estimation, which is based on distance computations between projected points.

5.2.2 Scalability

PCAF has outstanding scalability for very high-precision k-NN search. Compared with BF, PCAF only frequently accesses $\frac{d}{D}$ of the dataset. Due to space restrictions we only use the dataset ("SIFT") as the example here to compare the scalability with BF for highly precise (in this case exact) k-NN search.

As shown in Figure 3a, both PCAF and BF have considerable scalability on the Intel16 platform. By examining hardware performance monitoring counters, we find that PCAF has a 20% reduction in L2 cache misses compared to BF, but

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2017.2748131, IEEE Transactions on Computers



(a) Intel16 (b) AMD64

Fig. 3. Scalability of PCAF and BF for "SIFT" dataset to find exact k-NN results on the Intel16 and the AMD64 platforms.

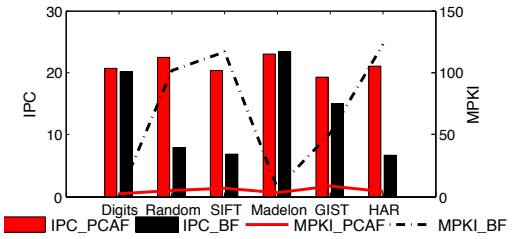


Fig. 4. Performance monitoring counter statistics for the AMD64.

the scalability improvement due to having fewer memory accesses is minor. This is because the Intel16 platform has a high memory bandwidth, so that the memory wall problem does not emerge. However, as the current trend is for CPUs to include more and more cores, the memory bandwidth will eventually be an inevitable issue.

Figure 3b shows the scalability on AMD64, where PCAF provides significant scalability while the curve of BF is quite flat. This is because unlike the Intel16 platform, the 64-core AMD64 platform suffers serious memory latency issues. The statistics from performance monitoring counters shown in Figure 4 help explain the results: the improvements to retired Instructions Per Cycle (IPC), which indicates the computation efficiency of the algorithm, is highly related to the reduction of last-level cache Misses Per (1000) Instructions (MPKI). When MPKI decreases, IPC increases accordingly, which leads to better scalability. Since the MPKI of PCAF is really small in all the cases, PCAF can be seen to be cache-friendly, and thus achieves substantial scalability.

5.3 Comparison with Data Selection Algorithms

Randomized kd-trees (RKD) and Random Ball Cover (RBC) are two typical *data selection* algorithms for AkNN search. RKD is highly efficient but its tree-based index structure becomes a barrier to achieving high parallel performance. RBC is developed as a state-of-the-art k-NN algorithm for multicore platforms, however it involves a large amount of unnecessary distance computations.

Compared with RKD and RBC where approximate k-NN results are required, PCAF produces higher precision results within shorter search times, and shows large improvements in scalability.

5.3.1 Performance Improvement

Figure 5 shows the search time of each algorithm on the Intel16 platform when using all available cores for different datasets reaching above 90% precision. The search time of BF is marked in the figure for reference. Due to space constraints, we omit the AMD64 results as they show similar trends to the Intel16 platform.

Figure 5 shows PCAF to be the only method that can provide exact results for every dataset. It is also the quickest to produce high precision k-NN results. As the majority of unnecessary distance computations between points is filtered, the advantage becomes much clearer as the dimensionality of dataset increases.

RKD produces k-NN results with good precision in a very short time only for lower-dimensional datasets such as "Digits". This is because RKD divides the reference space into bins along the axis of dimensionality, which makes it efficient for searching in low-dimensional space [9]. When it comes to high-dimensional space where the *curse of dimensionality* arises, RKD needs to spend much more time on visiting many more branches to achieve high precision [45]. Moreover, the precision of k-NN results found by RKD converges at a precision that is less than 100%.

RBC performs better than RKD in most cases, but it produces high precision results more slowly than PCAF. The only exceptions to this occur for lower precision (<97%) k-NN results from the "Digits" and "SIFT" datasets, where RBC is slightly faster than PCAF. However RBC cannot sustain results at high precision. Above 99.55% ("Digits") and 99.99% ("SIFT") respectively, RBC takes $5.18\times$ and $2.91\times$ the search time of PCAF.

The improvement mainly comes from PCAF avoiding computation, as shown in Figure 6a. Based on performance monitoring counters, the ratio of total retired instructions compared to BF is the lowest for PCAF. A large proportion of computation is avoided compared to RKD and RBC, which brings PCAF the substantial improvements observed. Note that the number of retired instructions in RBC is more than that in BF since RBC performs redundant distance computations to achieve high precision.

5.3.2 Scalability

In this section, we evaluate the scalability of our algorithm compared with RKD and RBC. Given the space available, we only examine the "Madelon", "SIFT" and "HAR" datasets, which were found to be representative. The sizes of "Madelon", "Digits" and "GIST" are all quite small. "HAR" is much larger than "Madelon", and the dimensionality is the highest of all the datasets. "SIFT" is the biggest dataset among all the datasets, but the dimensionality is only a quarter of "HAR", and "Random" is similar. To consider the scalability of each algorithm under a reasonable search precision, we plot the average speedup of selected cases that produce results that are more than 95% accurate. We also excluded results that have longer search times than BF.

The difference in scalability between each algorithm on Intel16 is not substantial, as shown in Figure 7, and which has been explained in Section 5.2.2. While the AMD64 platform showed significant variance between repeated experimental runs. We thus add the maximum speedup that we observed in each case as a dotted line on the graphs in Figure 8. As "Madelon" is small enough to fit in the last-level cache, as on the Intel16 platform, the scalability improvement is not obvious. However for "SIFT" and "HAR", a disparity is seen. In all cases, PCAF reaches the best maximum scalability, and provides the best average speedup.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2017.2748131, IEEE Transactions on Computers

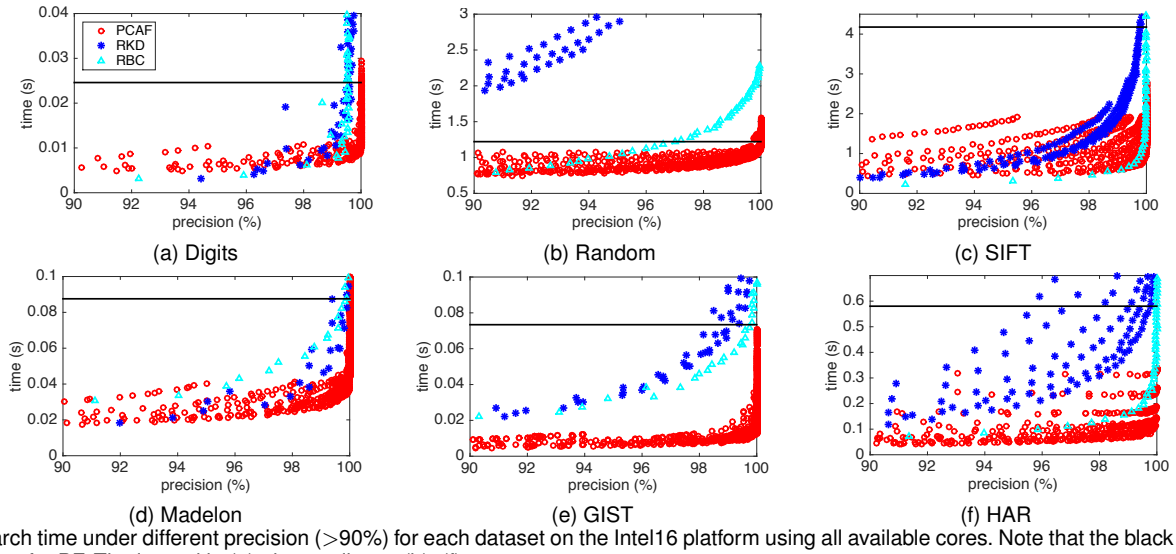


Fig. 5. Search time under different precision (>90%) for each dataset on the Intel16 platform using all available cores. Note that the black line shows the time cost for BF. The legend in (a) also applies to (b)–(f).

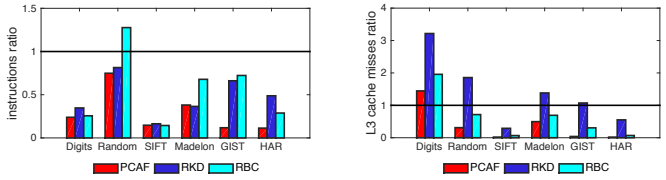


Fig. 6. Ratio of performance monitoring counters compared with BF for each algorithm on the AMD64 platform. Bars below 1.0 present desirable reductions, otherwise show the undesired overhead.

The last level cache misses recorded in the performance monitoring counters indicates the size of memory footprint. From the ratio of misses compared with BF shown in Figure 6b, PCAF can be seen to gain memory performance in almost all cases except "Digits", which is a small dataset. For very small datasets like "Digits", the benefit of reduced memory footprint is overshadowed by the non-contiguous, interleaved memory accesses to points and projected points, though the benefit of reduced computation still dominates the overall performance of PCAF.

5.4 Comparison with a Data Filtering Algorithm

As far as we know, SCF is the only other published approach to AkNN search that adopts a *data filtering* strategy. Compared to SCF, PCAF produces stable and higher precision results within less time and with similar scalability. In this section, we evaluate both the parallel performance and the effectiveness of the filtering.

The frequently accessed index in SCF has an $O(Ns)$ space complexity, where s is the number of subspaces (usually $s < 64$)—for detailed analysis refer to [8]. PCAF's projections set $\mathcal{R}'$ needs $O(Nd)$ space. Supposing each element in $\mathcal{R}'$ is a 32-bit floating point number, while each element of the index in SCF requires one byte of storage, then when $d < \frac{s}{4}$, PCAF will have a smaller memory footprint than SCF. As the dimensionality of projections in PCA space depends on the properties of the dataset, the d in our implementation for "Random" and "Madelon" is quite large (when 50% of the variance is retained, d is 61 and 53 respectively), while for other datasets, d is relative small—less than the dimensionality needed in the case of exact searching shown in Table 5. Still, the difference between d and $\frac{s}{4}$ is small, which results in comparative scalability. The speedup curves shown in Figure 9 support our analysis. As we can see, PCAF is only slightly better in scalability compared with SCF.

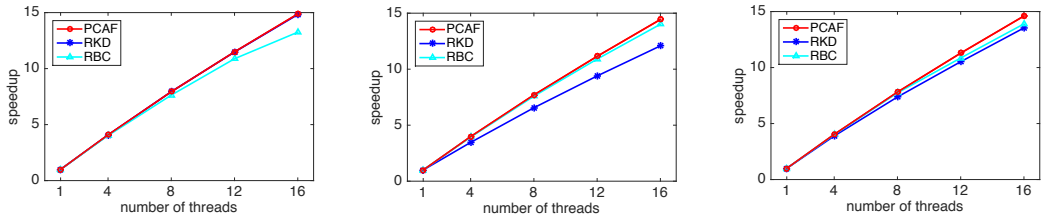
However, PCAF greatly improves the speed for high precision AkNN search. This is because PCAF's filtering strategy is more effective than SCF's, having a higher F in most cases, and thus avoids computation more effectively. Likewise, when achieving the same filtering rate, PCAF produces more precise results. In a data filtering algorithm, the AkNN results come from the non-filtered points, thus we compare the filtering effectiveness by plotting the precision achieved under the same filtering level in Figure 10. Due to limited space, we only present the high F levels between the range of $(95, 97]$ and $(97, 100]$, which are also key regions of concern for data filtering AkNN algorithms. Also note that the blank in "Madelon" for SCF is because the highest F SCF can reach for "Madelon" is only 82.74%. As we can see, PCAF produces nearly 100% precise results when $F >$ 95%, which is higher than SCF in all datasets. Thus there is a higher probability of the PCAF filtering method discovering the true k-NN points during searching, while false filtering occurs in SCF.

5.5 Preprocessing Comparison

In this section, we evaluate the preprocessing cost of PCAF with all the other algorithms' indexing cost using the six test datasets.

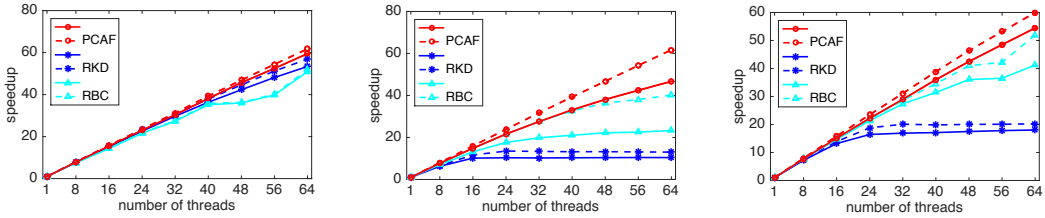
The preprocessing cost of PCAF and the indexing costs of all other algorithms on Intel16 are listed in Table 6. The left part of the table shows the time consumed in seconds when achieving a precision of >90% while the right part shows the preprocessing time for achieving a precision of >99%. The indexes are built by a single CPU core for all algorithms except RBC. RBC provides a parallel algorithm for index construction, so its index is built using 16 cores.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2017.2748131, IEEE Transactions on Computers



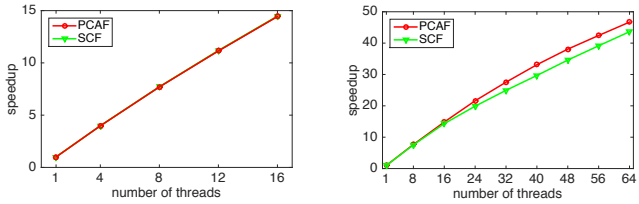
(a) Madelon (b) SIFT (c) HAR

Fig. 7. Scalability of each algorithm on the Intel16 platform, for results with a precision above 95%. The y-axis represents the speedup over the sequential algorithm.



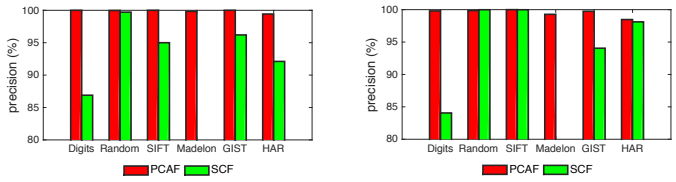
(a) Madelon (b) SIFT (c) HAR

Fig. 8. Scalability of each algorithm on the AMD64 platform, for results with a precision above 95%. The y-axis represents the speedup over the sequential algorithm. The continuous lines represent the average and dotted lines the maximum speedups that were observed.



(a) Intel16 (b) AMD64

Fig. 9. Scalability of SCF and PCAF for the "SIFT" dataset on the Intel16 and the AMD64 platforms.



(a) Filtering rate, F, between (95,97] (b) Filtering Rate, F, between (97,100]

Fig. 10. Precision of PCAF and SCF in the same filtering level

TABLE 6
Preprocessing cost of AkNN algorithms on Intel16

Dataset	Precision > 90%				Precision > 99%			
	PCAF	RKD	RBC	SCF	PCAF	RKD	RBC	SCF
Digits	0.01	0.03	0.02	0.06	×1.17	×7.68	×2.40	×1.76
Random	0.11	5.47	2.61	2.70	×1.05	×8.13	×2.96	×1.00
SIFT	0.23	6.43	0.57	4.51	×1.01	×1.00	×2.73	×1.40
Madelon	2.81	0.09	0.03	0.62	×1.00	×1.99	×1.48	×1.00
GIST	3.43	1.18	0.04	0.62	×1.00	×1.00	×2.23	×1.18
HAR	4.31	0.39	0.08	1.94	×1.00	×1.00	×1.82	×1.26

As suggested by the time complexity analysis in Section 4.4, PCAF's preprocessing is efficient when the dimensionality (D) of the dataset is relatively small compared with the dataset scale (N). Thus, in "Digits", "Random" and "SIFT" datasets, PCAF is the fastest. For the other higher-dimensional datasets "Madelon", "GIST" and "HAR", PCAF is slower than the others. This is because in these datasets, the number of points included is small (several thousands) compared with the dimensionality (of over 500). However, in reality, N is usually very large and dominates D. In the further tests on the real-world "GIST1M" dataset in Table 8, PCAF shows significant advantages over the others. We expect that datasets will increase faster in N than in D, and thus that PCAF will be increasingly efficient compared to other algorithms in dealing with such future datasets.

Besides, PCAF has relatively fixed preprocessing time. To produce more accurate results, PCAF usually projects points into a higher-dimensional PCA space, which means only redoing the last step of the preprocessing with a new $\tilde{\mathbb{U}}$ while the results of all the other steps are exactly the same. Thus, the time consumed for PCAF does not vary much in all the datasets, which is shown in the right part of Table 6. However, for the other algorithms, the indexing time increases since RKD has to build more indexing trees, RBC has to choose more representatives with more duplicated points, and SCF has to divide the space into more subspaces and form more clusters. Moreover, since all the other algorithms have to rebuild entirely the indexes but PCAF only needs to redo the last step, the precision tuning cost of PCAF is much cheaper compared to the others.

5.6 Case Study of a Very Large Dataset

PCAF is very efficient in processing very large, high-dimensional datasets and offers high search precision. We use a very large and high-dimensional dataset—"GIST1M"—to evaluate the performance of PCAF and other algorithms in a real 'big data' application. We set the precision to be 95% for all algorithms. In our study, we show the best parallel execution results for each algorithm that satisfies the required precision.

Table 7 shows the performance of each algorithm on the Intel16 and the AMD64 platforms. Compared with BF, the performance improvements of PCAF on the Intel16 and the AMD64 are 28.9× and 70.46× respectively, which are the best of all the algorithms. In these cases, PCAF estimates the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2017.2748131, IEEE Transactions on Computers

TABLE 7
Performance of the parallel k-NN algorithms with precision over 95%.

Algorithm	Precision	Intel16		AMD64	
		◇	●	◇	●
PCAF	95.20%	2.22	28.90	2.72	70.46
BF	100.00%	64.22	1	191.64	1
RKD	95.15%	13.35	4.81	32.67	5.87
RBC	95.00%	10.35	6.21	31.80	6.03
SCF	95.10%	12.75	5.04	16.61	11.54

◇ denotes the search time in seconds when using multiple cores.
● denotes the improvement against parallel BF when using multiple cores.

TABLE 8
Parameters and preprocessing time of each algorithm.

Algorithm	Parameters	Preprocessing (s)	
		Intel16	AMD64
PCAF	[d m S]: [29 2 16]	49.87	111.61
RKD	[*trees checks*]: [128 55000]	2930.63	3884.36
RBC	[*representatives*]: [60000]	1294.14	2700.19
SCF	[*subspaces clusters*]: [176 120]	2925.34	4884.64

rank of points based on a projected 29-dimensional (instead of 960-dimensional) space, which results in a high filtering rate of 99.79%.

Other algorithms do not perform as well as PCAF for the dataset due to the following observations of our experimental statistics in Table 8. RKD needs to build hundreds of index trees and check thousands of branches for each query to achieve a high precision of over 95% in such a high-dimensional space, which is very time consuming. RBC needs to duplicate $> 59\times$ the number of points in total to ensure the high precision, which increases the search time. Similar to PCAF, SCF can reduce computation by filtering 99.97% of the points from the distance calculations. However, the cost of estimation in the 960-dimensional space using hundreds of subspaces and clusters is quite high.

Figure 11 shows PCAF has better scalability than other algorithms on both Intel16 and AMD64 platforms, validating our previous conclusion that PCAF has much better memory performance when the dataset is larger. Nonetheless, the scalability curve of PCAF flattens after 40-threads on AMD64, when it hits a memory bottleneck. This is later than the others, which flatten at or before 24-threads.

Table 8 also lists the preprocessing time for each algorithm on the two platforms. The preprocessing is executed by a single core for all algorithms except RBC, which is run with all available cores. Although preprocessing is a one-off cost, PCAF only takes 49.87s on the Intel16 and 111.61s on the AMD64 to preprocess the "GIST1M" dataset—the shortest among all the algorithms.

6 Parameter Selection

Balancing performance and search precision is a relevant issue for AkNN algorithms. Typically, higher performance can be obtained by sacrificing precision. However, very low precision is definitely not acceptable in many application domains. Balancing this trade-off requires the careful choice of parameter values. In this section, we discuss the impact of each parameter upon performance and precision in PCAF to help with parameter selection.

There are three important parameters within PCAF: (i) the dimensionality of PCA space, d, (ii) the scaling factor, m, for adjusting the size of the heap, and (iii) the number of partitions, S, for parallel processing of the dataset.

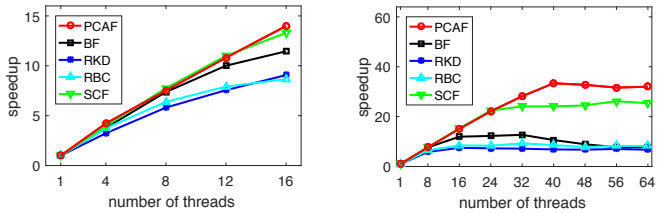


Fig. 11. Scalability of the parallel k-NN algorithms for "GIST1M" on the Intel16 and the AMD64 platforms.

Including the metrics that have been introduced in Section 5.1.5, we add an extra metric named *effectiveness* (ξ) to evaluate PCAF. It is the product of filtering rate and precision—$\xi = F \times precision$. In this case, neither low filtering rate nor poor precision would lead to good effectiveness as high filtering rates are useless when they result in poor precision, and low filtering rates with good precision involve a high computational load.

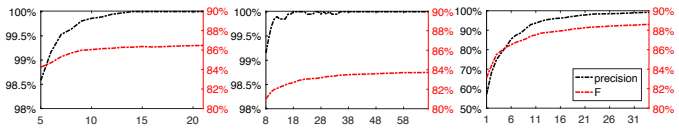
6.1 Selection of Dimensionality in PCA space

Selection of the dimensionality, d, of the PCA space is very important to PCAF as it impacts on every evaluation metric.

Theoretically, d can be chosen within the range of [1, D]. If d is chosen to be identical to D, then the complete variation information is contained in the projected space, which can provide accurate ranks. But in this case, the rank estimation requires distance computation between the D-dimensional projections, which has the same high cost compared with the real distance computation between the original points. At the other extreme, if only one principal component is used, the rank estimation has the least overhead but may cause a large loss of precision. Therefore, an optimal choice of d is vital to balance between the precision and performance of PCAF. However, the choice is difficult to make because different datasets have different data variation patterns. For some datasets, like "HAR", a small proportion of principal components is sufficient to contain sufficient information about variance, but for others, like "Random", about one half of the components are needed to guarantee the required precision. In our experiments, we require a fixed percentage of variation to be retained in the PCA space. Based on the percentage, we can derive the required number of components. This method is more adaptive and evidence-based than setting a fixed number of components. More specifically, the lower and upper bound of d in the experiments are set respectively to 50% and 90% of the variance that is retained in the PCA space. The range of d according to this method has been listed previously in Table 3. In order to have a fair comparison, the settings of other parameters are kept the same, *e.g.*, the heap scaling is set to 2 and the datasets are partitioned into 16 parts.

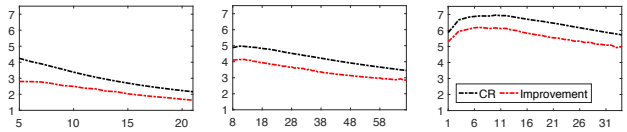
Figure 12 shows that both F and $precision$ increase with d, thus so will ξ, which indicates that with more components used for the PCA space, PCAF becomes more effective—a larger number of points are filtered without calculating their real distances but the k-NN results are more precise. This is because projection into a higher-dimensional PCA

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2017.2748131, IEEE Transactions on Computers



(a) Digits (b) GIST (c) HAR

Fig. 12. The $precision$ and filtering rate (F) under different dimensionality of the PCA space (d). The axes of x, left-y and right-y are the number of d, $precision$ and filtering rate in percentage. The legend in (c) applies to all subfigures.



(a) Digits (b) GIST (c) HAR

Fig. 13. Computation reduction (CR) and sequential improvement of PCAF over the brute force k-NN with an increasing d values Intel16. The x-axis is the number of d. The left y-axis represents the value of CR or improvement. The legend in (c) applies to all subfigures.

space contains more variance information about the original reference dataset. Also, in a higher-dimensional PCA space, the rank of the projected points in the PCA space gets closer to the rank of the original data points. Thus more data points can be correctly filtered.

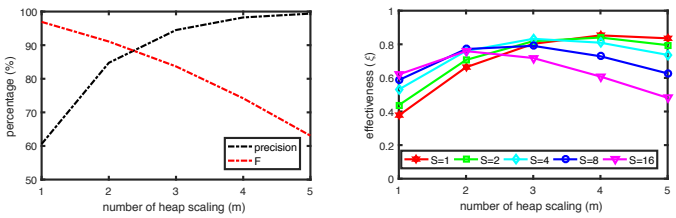
Although a larger d raises the effectiveness of PCAF, it also results in higher cost on distance computation in rank estimation. According to our algorithm, the total cost of distance computation, denoted as DC, consists of two parts: the distance calculation for the d-dimensional projected points during the rank estimation and the real distance calculation for the unfiltered original points. Assuming the distance computation cost of the brute force k-NN is DC_{BF}, then DC can be derived in Equation 5. The computation reduction of PCAF over the brute force k-NN algorithm, CR, is expressed in Equation 6.

$$DC \approx \frac{d}{D} \times DC_{BF} + (1 - F) \times DC_{BF} \tag{5}$$

$$CR = \frac{DC_{BF}}{DC} = \frac{1}{1 + \frac{d}{D} - F} \tag{6}$$

The search speed is highly affected by distance computation cost. This is supported by Figure 13, which shows a high relevance between the improvement of PCAF in sequential execution and CR. Equation 6 also indicates that when d is much smaller than D, CR should be dominated by F, as validated by Figure 13. For example, in the case of "HAR", the curve rises at the start due to the increasing filtering rate when the ratio of d to D is between $\frac{1}{561}$ and $\frac{7}{561}$. Then the curve becomes flat until d reaches 12. After that the performance gain from the increasing filtering rate is overshadowed by the distance computation in the PCA space with a larger d, which explains the drop of the curve. For other datasets, the ratio of d to D dominates CR either from the beginning or at an early stage, so the curve of CR starts to drop at the beginning.

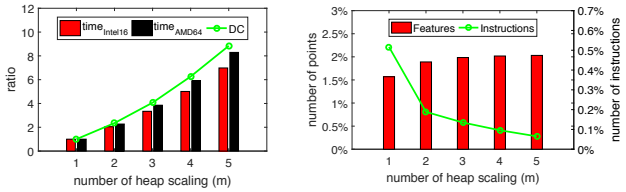
Note that only "Digits", "GIST" and "HAR" datasets running on the Intel16 platform are included in Figure 12 and Figure 13; the other datasets running on either Intel16 or AMD64 were found to show similar patterns.



(a) The $precision$ and F with different m when $d = 7, S = 8$.

(b) Effectiveness with different values of m and S when $d = 7$.

Fig. 14. The $precision$, filtering rate (F) and effectiveness (ξ) of PCAF with increasing heap scaling (m) applied to the "HAR" dataset.



(a) Ratio of distance computation (DC) and sequential search time on the Intel16 and the AMD64 platforms.

(b) Percentage of number of points that cause heap operations and of heap operational instructions on the AMD64 platform.

Fig. 15. The analytical statistics with an increasing value for m applied to the "HAR" dataset when $d = 7, S = 8$.

In summary, with increasing d, the precision at first rises up quickly, along with the performance improvement of PCAF. Later, the performance improvement is overshadowed by the increasing cost of distance computations. Therefore, we suggest choosing the value of d based on a critical point where the performance improvement stalls and yet the required precision is met.

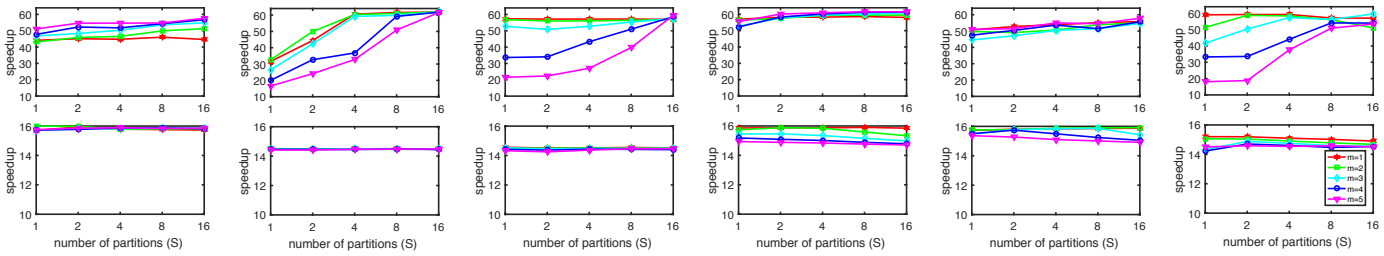
6.2 Impact of Heap Scaling

The heap scaling factor, m, adjusts the size of the filter heap that stores distances in the PCA space. Usually, for an application searching for k-NN results, the size the filter heap should be $m \times k$, where k is set to 2 in most of the real-world applications we investigated. m is set within the range $[1, 5]$ in our experiments.

In PCAF, the purpose of adjusting the size of the filter heap is to mitigate the negative effect brought about by false filtering. False filtering may occur when both the query and a true k-NN candidate lie closely along a non-principal component vector in the PCA space. In this case, the projection distance between the query and a true k-NN candidate point is incorrectly larger than the filtering threshold—the maximum value in the filter heap. When the heap size is enlarged, the filtering threshold becomes higher accordingly. More points are kept for further verification. This results in a falling filtering rate, but as more promising candidates are included in these points, it helps increase the precision. Figure 14a shows a typical case when m increases, F decreases and the precision becomes higher. Note that, since all the results are very similar, we only present the results of "HAR" dataset in this subsection.

Since the variation patterns of precision and the filtering rate under an increasing m act in an opposite manner, the effectiveness ξ plotted in Figure 14b can better demonstrate the impact of m on the effectiveness of PCAF. From the curves in the figure, each of which represents the variation

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2017.2748131, IEEE Transactions on Computers



(a) Digits (b) Random (c) SIFT (d) Madelon (e) GIST (f) HAR

Fig. 16. The maximum speedup with a different number of partitions (S) and heap scaling (m) for results with precision above 99% when using all cores on the AMD64 (upper plots) and the Intel16 (lower plots) platforms. The legend in (f) applies to all subfigures.

of ξ under the same S with different m, we can observe that with an increasing m, ξ always ascends to a peak, and then starts to drop. That means that in the beginning as m increases, the precision grows at a faster speed than the dropping of F, which conforms to our expectations, but the situation eventually reverses. This indicates that it is not worth trading the filtering rate for higher precision beyond that point.

Patterns in ξ are also affected by the number of partitions of the reference data, S. Based on Figure 14b, the effectiveness ξ reaches the peak sooner with a smaller m and a larger number of partitions. Although the highest ξ usually happens at a large m with a small S (in the example case in Figure 14b that $m = 4$ and $S = 1$), the variation of the peaks' value under different S is less than 0.1 among all six datasets. Thus, the setting of (m, S) pair at peaks is a guarantee of high filtering effectiveness.

The search time also increases with m due to the higher distance computation cost, which is brought about by F dropping, according to Equation 5. The relationship between the DC and search time can be used to verify our analysis. We illustrate this in Figure 15a, where we choose the case where m is 1 as a baseline and plot the ratio of DC (curve) and the ratio of search time (bars) for different m against this baseline. The consistent value and growth between the two ratios well demonstrates our point.

Note that besides the distance computation cost, the enlarged heap size theoretically adds extra system overhead thus slowing down the speed. However, this overhead is small enough to be neglected. This is because (i) each heap operation is cheap, (ii) the number of heap operations barely increases. In PCAF, the heap only involves one kind of operation—insertion—which has a theoretical time complexity of $O(\log km)$. Thus, the overhead of every operation remains small when k or m varies within a range of small values, which is the practical case in most of the real-world applications in many domains. From the bars in Figure 15b showing the average percentage of data points that are pushed into the filter heap, we can observe that increasing m does cause more frequent heap operations as the percentage increases, but that the growth rate is quite small ($< 0.5\%$). Thus, the total cost of heap operations remains low. In addition, the percentage of total heap operational instructions on the AMD64 platform, as plotted in Figure 15b further verifies our point. The percentage is less than 0.52% and keeps dropping when m increases because of the rapid increase in the computation cost. We confidentially conclude that the heap overhead is negligible to PCAF.

Based on the above, with increasing m, the precision keeps rising but the performance drops. Thus, the effectiveness (ξ) is helpful to find the balance between the precision and performance. We suggest choosing m at the peak in ξ when the required precision is first met in order to achieve the best overall performance.

6.3 Evaluation of Data Partitioning

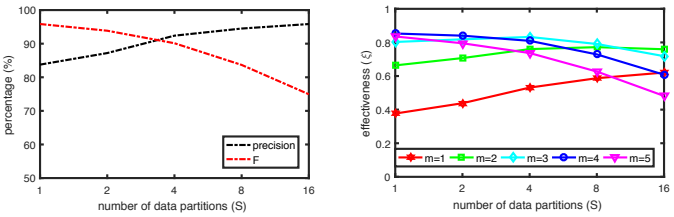
Data partitioning is a fine-grained parallel implementation for PCAF. The intention behind it is to take advantage of the independence in the search of k-NN within each query to increase the parallelism of the execution. In this section, we will discuss the impact of data partitioning on performance and the precision of k-NN results. In our experiments, the dataset is partitioned into $S \in \{1, 2, 4, 8, 16\}$ parts. In practice, S can be set larger, but our experimental setting is already sufficient for discussion.

In parallel execution of OpenMP, smaller granularity of parallelism brings more parallel tasks for the parallel threads so the speedup is larger due to better load balance among threads. However, if the granularity is too small, the benefit of balanced load may be overshadowed by task scheduling overhead. On the other hand, if the number of threads is small, the granularity of parallelism has little impact on performance as each thread has sufficient tasks to execute. Our experimental results conform to these general principles of parallel execution.

Figure 16 plots the maximum speedup that can be achieved under different S and m on both AMD64 and Intel16 platforms for all the datasets. Each speedup is computed against the sequential PCAF algorithm with the same S and m. The general impression of Figure 16 is that, when S is larger, the speedup is higher due to better load balance. However, there are some exceptions.

The first exception is that the results on the Intel16 platform stay almost the same when S becomes larger. This is because the Intel16 platform only has 16 cores so the parallelism is sufficient for it to have very high speedups even when S is 1. The second exception is that, when the dataset's size is small, *e.g.* for "Digits" or "Madelon", or when the heap scaling m is small in large datasets like "SIFT" and "HAR", the speedups stay the same on AMD64 for increased S. The reason is that the computation load is low in these cases, so the granularity of parallelism is already small enough even when S is 1. Therefore, a larger S has little impact on the load balance. In contrast, according to the analysis in Section 6.2, when the heap scaling m is large, the filtering rate is quite low, which leads to higher

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2017.2748131, IEEE Transactions on Computers



(a) The $precision$ and F with different S when $d = 7, m = 3$.

(b) Effectiveness with different S and m when $d = 7$.

Fig. 17. The $precision$, filtering rate (F) and effectiveness (ξ) of PCAF under increasing values of partitions (S) applied to the "HAR" dataset.

computational load. Therefore, a larger S has more impact on the speedup of the algorithm as it greatly reduces the granularity of parallelism and achieves better load balance in OpenMP.

One interesting observation is that the increasing S also causes higher computation load in total, and it produces more precise k-NN results. Figure 17a gives the typical relationship between precision and filtering rate with S—that the filtering rate drops with increasing S. Although the precision improves, it is not wise to keep increasing S since the computational cost will be high due to a very low filtering rate. To find the best trade-off, we suggest using the effectiveness (ξ) as an indicator. Taking the "HAR" dataset as an example, Figure 17b shows the ξ under different values of S and m. Observed from the figure, the filtering strategy is more effective with an increasing S when m is small. However, when m is larger than 2, the effectiveness curve drops from the very beginning, which indicates that it is not cost-effective anymore to increase S for high precision.

To summarise, as S increases, both the precision and the computational cost increase. For large S, the scalability of PCAF improves due to more parallelism and better load balance, especially for larger m values, that result in both higher speedup and higher precision. On the other hand, when m is small, S can only be increased to the point where the desired precision is met in order to prevent encountering problematic computational costs.

6.4 Discussion

Based on the evaluation described above, in Table 9 we show the values of the parameters of the fastest parallel searching case with a precision loss within 5% for each dataset.

TABLE 9
Parameter selection of the fastest parallel searching case with precision over 95%.

Name	d	m	S	F	$Pr\%$	Intel16		AMD64	
						◇	●	◇	●
Digits	8	2	2	96.86	95.21	4.63	15.97	4.92	45.87
Random	65	3	8	90.34	95.16	1.54	14.51	5.16	60.09
SIFT	10	2	16	97.72	96.79	8.30	14.50	31.34	56.95
Madelon	58	2	4	91.07	95.08	4.27	15.91	4.37	58.06
GIST	20	2	1	97.82	95.30	12.20	15.74	12.68	49.05
HAR	28	2	2	98.09	95.22	11.53	15.04	44.74	58.71

$Pr\%$ denotes precision: the percentage of correctly found k-NN.
◇ denotes improvement: the search time ratio of PCAF to BF when using all available cores.
● denotes speedup: the search time ratio of parallel PCAF when using all cores to sequential PCAF execution.

In our experiments, there are lots of suitable sets of (d, m, S) that meet the requirement of a precision loss within 5%, thus we have to compare the performance of each case to get the best set among them. In practice, the evaluation of parameters can only help with parameter selection to some degree. Although it is relatively easy to find suitable sets of (d, m, S) that achieve desirable precision—since the precision increases with any of d, m and S—it is not an easy task to find the set with the best performance. First, even if the variation pattern of each parameter is observed, there are still quite a few possible sets to search, which is time consuming. Second, the three metrics to some degree depend upon each other which makes the selection more complex. Thus, it is worth devoting effort in future to seek intelligent parameter learning approaches or to select appropriate training samples to help find targeted parameter sets at a reasonable cost.

7 Related Work

The focus of our work is speeding up high-dimensional, high precision k-NN search on multicore architectures. We are not aware of many similar efforts to optimise AkNN algorithms for performance and precision in this way. Still, the idea behind our method stems from related research.

7.1 Data Filtering for AkNN on multicore systems

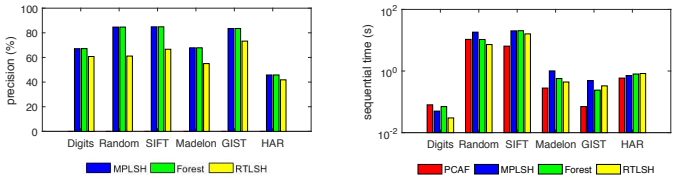
The algorithm previously proposed by Tang [8], [16] is the key work defining the notion of data filtering for AkNN. The rationale is to use a low-cost method to filter away many expensive distance computations, and maintain a small memory footprint at the same time achieving high parallel performance. We adopt this notion of data filtering. However, the precision of Tang's algorithm is not stable, and is highly dependent on the nature of the dataset used.

7.2 Data Selection for AkNN on multicore systems

Apart from the two popular algorithms RKD and RBC evaluated previously, there are other algorithms such as SONNET [7] using data selection strategy. SONNET seeks the kNN candidates based on the value rank along each dimension. However, the point near the query in one dimension is not highly likely the true candidate in the whole D-dimensional space, so the precision of SONNET is of concern. We have conducted extra experiments for SONNET [35]. According to our experiments, PCAF is much faster and produces results of higher precision than SONNET.[1] PCAF is also better in scalability since every distance computation in SONNET requires one extra random access to fetch the point from the dataset, and the index size of SONNET is $\frac{1.5D}{d}\times$ the size of projections in PCAF. The precision of SONNET was too low in our preliminary tests for us to include it in our full experimental evaluation.

1. In our experiments, the highest 2-NN search precision of SONNET under $frac = 0.05$ is 17.71% (for "HAR" dataset). Under equal filtering rate that $FR = (1 - frac) \times 100\% = 95\%$, PCAF produces results with precision over 90% and has a 2.17× speedup on the synthetic "Random" dataset and at least 7.74× for other datasets over SONNET.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2017.2748131, IEEE Transactions on Computers



(a) The converged $precision$ of various LSH algorithms.

(b) Time cost of PCAF with precision >90% and of LSH algorithms near their converged precision ($< 85\%$)

Fig. 18. Performance of PCAF and various LSH algorithms when using one core for searching on Intel16.

7.3 Dimension Reduction

Many real-world high-dimensional datasets are actually governed by a small number of dominant dimensions, as discussed in [25] which inspires our data filtering method using lower-dimensional projections. The idea of using low-dimensional intrinsic structure in high-dimensional space is popular in computer vision [18], [46], [47]. Many solutions proposed for k-NN search in computer vision use dimensionality reduction techniques to either form lower-dimensional feature descriptors [48] or sort points in advance [49]. However, these techniques either reduce the dimensionality of the space directly which results in loss of information, or incur extra overhead *e.g.* using sorting. In PCAF we use dimensionality reduction for filtering process without changing the original space for searching, which retains high precision of k-NN search and reduces computation.

7.4 Feature Hashing

Hashing based similarity searching represents a category of algorithms [10], [50], [51], [52] dealing with high-dimensional data points. The most popular algorithm in this category is Locality-Sensitive Hashing (LSH), which has much in common with nearest neighbour search. LSH applies a hash function to each point and aims for similar points to have the same hash value with high probability. Search can be efficient [53], but its precision is not stable, as it is highly dependent on the hash function.

Figure 18a shows that the highest precision achieved by three LSH based AkNN algorithms are all less than 85%. The algorithms, Multi-probe LSH (MPLSH) [10], [54], Forest [52] and Random Thresholding LSH (RTLSH), are taken from a popular LSH library named LSHKIT [55]. An appropriate hash function for LSH may improve the precision, but designing such functions is always time consuming [6], [10]. In contrast, PCAF can easily achieve high precision by adjusting the parameter d due to the principle of the principal component analysis. Figure 18b plots the fastest cases found by LSH variants achieving near-converged precision over a large number of trials using various parameter settings in comparison of the case of PCAF over 90% precision. PCAF produces results of higher precision within less time for most of the datasets. Since LSHKIT only provides sequential implementations, the results in Figure 18b are obtained from sequential executions. LSH requires a memory space of $O(LN)$, where L is the number of hash tables from tens to hundreds. In contrast, the memory footprint of PCAF, $O(\frac{dN}{D}), d \ll D$, is much smaller. Considering the memory wall in multicore systems and the better sequential performance of PCAF, the parallel performance of PCAF is presumed to be better than parallel LSH based algorithms, of which no implementations appear to be available yet.

7.5 Subspace Learning

Linear subspace learning algorithms are dimensionality reduction techniques that represent data as vectors and then search for an optimal linear mapping to a lower-dimensional space. The PCA algorithm adopted in PCAF is a typical example of such algorithms. Other popular methods include Independent Component Analysis (ICA) [56] and Linear Discriminant Analysis (LDA) [57].

ICA focuses on independent and non-Gaussian components and performs a non-orthogonal transformation to separate a set of source signals from a set of mixed signals with little or no information about the signals. ICA is often applied [58] to Blind Signal Separation (BBS) applications, such as speech isolation [59], brain imaging [60], and stock prediction [61]. LDA and its variants [62], [63] are usually used in statistics and pattern recognition to find a linear combination of features that characterises or separates two or more classes of objects or events. LDA is closely related to PCA, but PCA does not consider classification. Linear dimensionality reduction algorithms are restrictive in the sense that the spaces are linear.

Recently, there have been many manifold learning algorithms [64], [65], [66] proposed for nonlinear dimensionality reduction. These algorithms work towards extracting the low-dimensional manifold that can be used to describe the high-dimensional data distributed in the space with complex (*e.g.* twisted or rolling) shapes.

8 Conclusions

Data selection AkNN algorithms encounter serious memory bottlenecks on multicore systems. Using a data filtering strategy that reduces computation and memory footprint can improve scalability. However it can be challenging to maintain the accuracy of the result set with reduced computation. In this paper, a novel filtering method, PCAF, is proposed to exclude unlikely k-NN points with accurate estimation of similarity in high-dimensional space. Experimental results show that PCAF achieves substantial speedups and good scalability on multicore platforms compared with many data selection AkNN algorithms, and provides higher precision than an existing data filtering algorithm. This paper extends [67] with more details and many more experimental results, including new parameter selection analyses for finding appropriate parameter settings for precision and performance.

9 Acknowledgements

We thank the anonymous reviewers for their valuable comments and suggestions. Huan Feng would like to thank the University of Otago and China Scholarship Council for offering and sponsoring her internship during the course of this research. We are grateful to Xiaoxin Tang and Mohammad Al Hasan for their generous provision of source

code. This work is partially supported by National Key Research & Development Program of China (2016YFB1000504), Natural Science Foundation of China (61433008, 61373145, 61572280, U1435216), National Basic Research (973) Program of China (2014CB340402), Shenzhen City Branch Committee under contract No. 2016-092.

REFERENCES

[1] J. Buhler, "Provably sensitive indexing strategies for biosequence similarity search," *Journal of Computational Biology*, vol. 10, no. 3-4, pp. 399–417, 2003.

[2] D. L. Donoho *et al.*, "High-dimensional data analysis: The curses and blessings of dimensionality," *AMS Math Challenges Lecture*, pp. 1–32, 2000.

[3] A. Torralba, R. Fergus, and W. T. Freeman, "80 million tiny images: A large data set for nonparametric object and scene recognition," *Pattern Analysis and Machine Intelligence*, vol. 30, no. 11, pp. 1958–1970, 2008.

[4] Z. Wu, Q. Ke, M. Isard, and J. Sun, "Bundling features for large scale partial-duplicate web image search," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009*. IEEE, 2009, pp. 25–32.

[5] C. Zanchettin, B. L. D. Bezerra, and W. W. Azevedo, "A KNN-SVM hybrid model for cursive handwriting recognition," in *Neural Networks (IJCNN), The 2012 International Joint Conference*. IEEE, 2012, pp. 1–8.

[6] L. Cayton, "Accelerating nearest neighbor search on manycore systems," in *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*. IEEE, 2012, pp. 402–413.

[7] M. Al Hasan, H. Yildirim, and A. Chakraborty, "SONNET: Efficient approximate nearest neighbor using multi-core," in *Data Mining (ICDM), 2010 IEEE 10th International Conference on*. IEEE, 2010, pp. 719–724.

[8] X. Tang, Z. Huang, D. Eyers, S. Mills, and M. Guo, "Scalable multicore k-NN search via Subspace Clustering for Filtering," *Parallel and Distributed Systems, TPDS, IEEE Transactions on*, vol. 26, no. 12, pp. 3449–3460, 2015.

[9] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," *VISAPP*, vol. 2, pp. 331–340, 2009.

[10] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe LSH: efficient indexing for high-dimensional similarity search," in *Proceedings of the 33rd international conference on Very large data bases*. VLDB Endowment, 2007, pp. 950–961.

[11] D. Nister and H. Stewenius, "Scalable recognition with a vocabulary tree," in *Computer vision and pattern recognition, 2006 IEEE computer society conference on*, vol. 2. IEEE, 2006, pp. 2161–2168.

[12] H. Jégou, M. Douze, and C. Schmid, "Product Quantization for Nearest Neighbor Search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 1, pp. 117–128, Jan. 2011.

[13] T. Liu, A. W. Moore, K. Yang, and A. G. Gray, "An investigation of practical approximate nearest neighbor algorithms," in *Advances in Neural Information Processing Systems*, 2004, pp. 825–832.

[14] Y. Tao, K. Yi, C. Sheng, and P. Kalnis, "Quality and efficiency in high dimensional nearest neighbor search," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. ACM, 2009, pp. 563–576.

[15] G. Cong and K. Makarychev, "Optimizing large-scale graph analysis on multi-threaded, multicore platforms," in *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*. IEEE, 2012, pp. 414–425.

[16] X. Tang, S. Mills, D. Eyers, K.-C. Leung, Z. Huang, and M. Guo, "Data filtering for scalable high-dimensional k-NN search on multicore systems," in *Proceedings of the 23rd international symposium on High-performance Parallel and Distributed Computing, HPDC 2014*. ACM, 2014, pp. 305–310.

[17] I. T. Jolliffe, *Principal Component Analysis, 2nd*. Springer, 2002.

[18] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When is "nearest neighbor" meaningful?" in *International conference on database theory*. Springer, 1999, pp. 217–235.

[19] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on Mathematical Statistics and Probability*. Oakland, CA, USA., 1967, pp. 281–297.

[20] A. Abdullah, A. Andoni, R. Kannan, and R. Krauthgamer, "Spectral approaches to nearest neighbor search," in *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*. IEEE, 2014, pp. 581–590.

[21] B. Pandya, U. Singh, and K. Dixit, "An analysis of euclidean distance preserving perturbation for privacy preserving data mining," *International Journal for Research in Applied Science and Engineering Technology*, vol. 2, 2014.

[22] J. McNames, "A fast nearest-neighbor algorithm based on a principal axis search tree," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 9, pp. 964–976, 2001.

[23] K. Mikolajczyk and C. Schmid, "A performance evaluation of local descriptors," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 27, no. 10, pp. 1615–1630, 2005.

[24] D. G. Lowe, "Object recognition from local scale-invariant features," in *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, vol. 2. Ieee, 1999, pp. 1150–1157.

[25] H. Feng, S. Mills, D. Eyers, X. Shen, and Z. Huang, "Optimal space subdivision for parallel approximate nearest neighbour determination," in *Proceedings of 30th International Conference on Image and Vision New Zealand*. IEEE, 2015.

[26] G. Treen and A. Whitehead, "Efficient SIFT matching from keypoint descriptor properties," in *Applications of Computer Vision (WACV), 2009 Workshop on*. IEEE, 2009, pp. 1–7.

[27] R. E. G. Valenzuela, W. R. Schwartz, and H. Pedrini, "Dimensionality reduction through PCA over SIFT and SURF descriptors," in *Cybernetic Intelligent Systems (CIS)*, 2012, pp. 58–53.

[28] S. Lahabar and P. Narayanan, "Singular value decomposition on GPU using CUDA," in *Parallel and Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*. IEEE, 2009, pp. 1–10.

[29] W. Liu, H. Zhang, D. Tao, Y. Wang, and K. Lu, "Large-scale paralleled sparse principal component analysis," *Multimedia Tools and Applications*, pp. 1–13, 2014.

[30] J. Zhang and K. H. Lim, "Implmentation of a covariance-based principal component analysis algorithm for hyperspectral imaging applications with multi-threading in both CPU and GPU," in *2012 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*. IEEE, 2012, pp. 4264–4266.

[31] G. H. Golub and C. Reinsch, "Singular value decomposition and least squares solutions," *Numerische mathematik*, vol. 14, no. 5, pp. 403–420, 1970.

[32] J. A. Gunnels, G. M. Henry, and R. A. Van De Geijn, "A family of high-performance matrix multiplication algorithms," in *Computational Science, ICCS 2001*. Springer, 2001, pp. 51–60.

[33] S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, and J. Demmel, "Optimization of sparse matrix–vector multiplication on emerging multicore platforms," *Parallel Computing*, vol. 35, no. 3, pp. 178–194, 2009.

[34] G. Guennebaud, B. Jacob *et al.*, "Eigen v3.2.8 JacobiSVD class template reference," http://eigen.tuxfamily.org/dox/classEigen_1_1JacobiSVD.html/, 2016.

[35] H. Feng, "PCAF: Pricipal conponent analysis based filtering," 2016. [Online]. Available: https://github.com/c30268056/PCAF

[36] C. Silpa-Anan and R. Hartley, "Optimised kd-trees for fast image descriptor matching," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*. IEEE, 2008, pp. 1–8.

[37] M. Muja and D. G. Lowe, "Scalable nearest neighbor algorithms for high dimensional data," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 36, no. 11, pp. 2227–2240, 2014.

[38] G. Guennebaud, B. Jacob *et al.*, "Eigen: a C++ linear algebra library," http://eigen.tuxfamily.org, 2014.

[39] V. Hernández, J. E. Román, and A. Tomás, "A robust and efficient parallel SVD solver based on restarted Lanczos bidiagonalization," *Electronic Transactions on Numerical Analysis*, vol. 31, pp. 68–85, 2008.

[40] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," *International Journal of Computer Vision*, vol. 42, no. 3, pp. 145–175, 2001.

[41] N. Khan, B. McCane, and S. Mills, "Better than SIFT?" *Machine Vision and Applications*, vol. 26, no. 6, pp. 819–836, 2015.

[42] K. Bache and M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[43] I. Guyon, S. Gunn, A. Ben-Hur, and G. Dror, "Result analysis of the NIPS 2003 feature selection challenge," in *Advances in Neural Information Processing Systems*, 2004, pp. 545–552.

[44] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine," in *Ambient assisted living and home care*. Springer, 2012, pp. 216–223.

[45] S. A. Nene and S. K. Nayar, "A simple algorithm for nearest neighbor search in high dimensions," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 19, no. 9, pp. 989–1003, 1997.

[46] G. Hua, M. Brown, and S. Winder, "Discriminant embedding for local image descriptors," in *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*. IEEE, 2007, pp. 1–8.

[47] K. Mikolajczyk and J. Matas, "Improving descriptors for fast tree matching by optimal linear projection," in *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*. IEEE, 2007, pp. 1–8.

[48] Y. Ke and R. Sukthankar, "PCA-SIFT: A more distinctive representation for local image descriptors," in *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, vol. 2. IEEE, 2004, pp. II–506.

[49] G. Treen and A. Whitehead, "A PCA-based binning approach for matching to large SIFT database," in *Computer and Robot Vision (CRV), 2010 Canadian Conference on*. IEEE, 2010, pp. 9–16.

[50] A. Gionis, P. Indyk, R. Motwani *et al.*, "Similarity search in high dimensions via hashing," in *VLDB*, vol. 99, no. 6, 1999, pp. 518–529.

[51] L. Paulevé, H. Jégou, and L. Amsaleg, "Locality sensitive hashing: A comparison of hash function types and querying mechanisms," *Pattern Recognition Letters*, vol. 31, no. 11, pp. 1348–1358, 2010.

[52] M. Bawa, T. Condie, and P. Ganesan, "LSH forest: selftuning indexes for similarity search," in *Proceedings of the 14th international conference on World Wide Web*. ACM, 2005, pp. 651–660.

[53] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proceedings of the 12th annual symposium on Computational geometry*. ACM, 2004, pp. 253–262.

[54] W. Dong, Z. Wang, W. Josephson, M. Charikar, and K. Li, "Modeling LSH for performance tuning," in *Proceedings of the 17th ACM conference on Information and Knowledge Management*. ACM, 2008, pp. 669–678.

[55] W. Dong, "LSHKIT," 2008. [Online]. Available: http://lshkit.sourceforge.net/

[56] J. V. Stone, "Independent component analysis: A tutorial introduction," *The Knowledge Engineering Review*, vol. 20, no. 2, p. 198, 2005.

[57] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of human genetics*, vol. 7, no. 2, pp. 179–188, 1936.

[58] A. Hyvärinen and E. Oja, "Independent component analysis: algorithms and applications," *Neural networks*, vol. 13, no. 4, pp. 411–430, 2000.

[59] G. D. Brown, S. Yamada, and T. J. Sejnowski, "Independent component analysis at the neural cocktail party," *Trends in neurosciences*, vol. 24, no. 1, pp. 54–63, 2001.

[60] A. Delorme, T. Sejnowski, and S. Makeig, "Enhanced detection of artifacts in EEG data using higher-order statistics and independent component analysis," *Neuroimage*, vol. 34, no. 4, pp. 1443–1449, 2007.

[61] A. D. Back and A. S. Weigend, "A first application of independent component analysis to extracting structure from stock returns," *International journal of neural systems*, vol. 8, no. 04, pp. 473–484, 1997.

[62] T. Zhang, D. Tao, and J. Yang, "Discriminative locality alignment," *Computer Vision–ECCV 2008*, pp. 725–738, 2008.

[63] D. Tao, X. Li, X. Wu, and S. J. Maybank, "Geometric mean for subspace selection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 2, pp. 260–274, 2009.

[64] T. Zhou, D. Tao, and X. Wu, "Manifold elastic net: a unified framework for sparse dimension reduction," *Data Mining and Knowledge Discovery*, vol. 22, no. 3, pp. 340–371, 2011.

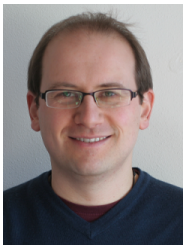
[65] W. Liu, Z.-J. Zha, Y. Wang, K. Lu, and D. Tao, "p-Laplacian regularized sparse coding for human activity recognition," *IEEE Transactions on Industrial Electronics*, vol. 63, no. 8, pp. 5120–5129, 2016.

[66] Q. Gao, Y. Huang, H. Zhang, X. Hong, K. Li, and Y. Wang, "Discriminative sparsity preserving projections for image recognition," *Pattern Recognition*, vol. 48, no. 8, pp. 2543–2553, 2015.
based filtering," in *Parallel Processing (ICPP), 2016 45th International Conference on*. IEEE, 2016, pp. 638–647.

[67] H. Feng, D. Eyers, S. Mills, Y. Wu, and Z. Huang, "PCAF: Scalable, high precision k-nn search using principal component analysis

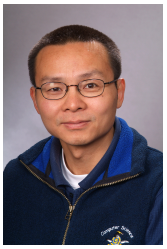
Huan Feng is a Ph.D candidate in Department of Computer Science and Technology, Tsinghua University. She received the bachelor degree from Beijing Jiaotong University in 2011. Her major research interests include high performance computing, parallel computing, and distributed systems.

David Eyers is a Senior Lecturer in the Department of Computer Science at the University of Otago. He received his PhD in Computer Science from the University of Cambridge in 2006. His research research interests include high efficiency distributed systems and cloud security.



Steven Mills is a Senior Lecturer in the Department of Computer Science at the University of Otago. He received his BSc (Hons) in 1997 and PhD in 2000 from Otago, and has previously worked at the University of Nottingham and in commercial research and development roles. His research interests are in computer vision, particularly the reconstruction of 3D scenes from multiple images and he has published over 80 peer-reviewed research articles.

Yongwei Wu received the PhD degree in applied mathematics from the Chinese Academy of Sciences in 2002. He is currently a professor in computer science and technology at Tsinghua University. His research interests include parallel and distributed processing, and cloud storage. Dr. Wu has published over 80 research publications and has received several best paper awards. He is currently on the editorial board of the IEEE Transaction on Cloud Computing, Journal of Parallel and Distributed Computing. He is an IEEE senior member.



Zhiyi Huang is an Associate Professor at the Department of Computer Science, University of Otago. He received the BSc degree in 1986 and the PhD degree in 1992 from the National University of Defense Technology (NUDT) in China. He was a visiting professor at EPFL (Swiss Federal Institute of Technology Lausanne) and Tsinghua University in 2005, Cambridge University in 2013, and a visiting scientist at MIT CSAIL in 2009. His research fields include parallel/distributed computing, multicore architectures, operating systems, signal processing, green computing, cluster/grid/cloud computing, high-performance computing, bio-engineering, and computer networks. He has more than 120 publications.