Fast Approximate kNN Graph Construction for High Dimensional Data via Recursive Lanczos Bisection*

Jie Chen† Haw-ren Fang† Yousef Saad†

October 2, 2008

Abstract

Nearest neighbor graphs are widely used in data mining and machine learning. The brute-force method to compute the exact kNN graph takes $\Theta(dn^2)$ time for n data points in the d dimensional Euclidean space. We propose two divide and conquer methods for computing an approximate kNN graph in $\Theta(dn^t)$ time for high dimensional data (large d). The exponent t depends on an internal parameter and is larger than one. Experiments show that a high quality graph usually requires a small t which is close to one. A few of the practical details of the algorithms are as follows. First, the divide step uses an inexpensive Lanczos procedure to perform recursive spectral bisection. After each conquer step, an additional refinement step is performed to improve the accuracy of the graph. Finally, a hash table is used to avoid repeating distance calculations during the divide and conquer process. The combination of these techniques is shown to yield quite effective algorithms for building kNN graphs.

1 Introduction

Building nearest neighbor graphs is often a necessary step when dealing with problems arising from applications in such areas as data mining [Brito et al., 1997, Dasarathy, 2002], manifold learning [Belkin and Niyogi, 2003, Rowies and Saul, 2000, Saul and Roweis, 2003, Tenenbaum et al., 2000], robot motion planning [Choset et al., 2005], and computer graphics [Sankaranarayanan et al., 2007]. Given a set of n data points $X = \{x_1, \ldots, x_n\}$, a nearest neighbor graph consists of the vertex set X and an edge set which is a subset of $X \times X$. The edges are defined based on a *proximity* measure between two data points x_i and x_j. Here, we assume that this proximity is measured by a quantity $\rho(x_i, x_j)$ which gives the distance/dissimilarity between x_i and x_j, i.e., $\rho(x_i, x_j)$ is small when the two points are close and $\rho(x_i, x_j)$ is large when they are farther apart. Two types of nearest neighbor graphs [Belkin and Niyogi, 2003, He and Niyogi, 2004] are often used:

1. ϵ-graph: This is an undirected graph whose edge set consists of pairs (x_i, x_j) such that $\rho(x_i, x_j)$ is less than some pre-defined threshold $\epsilon \in \mathbb{R}^+$.

*This work was supported by the National Science Foundation (grants DMS-0528492 and DMS-0510131) and by the Minnesota Supercomputer Institute.

†Department of Computer Science and Engineering, University of Minnesota at Twin Cities, MN 55455. Email: {jchen, hrfang, saad}@cs.umn.edu.

2. kNN graph: This is a directed graph (in general). There is an edge from x_i to x_j if and only if $\rho(x_i, x_j)$ is among the k smallest elements of the set $\{\rho(x_i, x_\ell) \mid \ell = 1, \ldots, i-1, i+1, \ldots, n\}$.

The ϵ-graph is geometrically motivated, and many efficient algorithms have been proposed for computing this graph [see, e.g., Bentley et al., 1977, Chazelle, 1983]. However, the ϵ-graph can result in disconnected components [Belkin and Niyogi, 2003] and it is difficult to find a good value of ϵ which yields graphs with appropriate number of edges. Hence, they are not suitable in many situations. On the other hand, kNN graphs have been shown to be especially useful in practice. Therefore, this paper will focus on the construction of kNN graphs.

The data points x_i's are in general defined on a metric space [Clarkson, 2006, Paredes et al., 2006], with a distance metric $\rho(\cdot, \cdot)$ satisfying the properties of non-negativity, symmetry, and triangular inequality. Non-metric measures are sometimes of interest [Zhang and Srihari, 2002] as well, though not as common in practice. The Euclidean space with the Euclidean metric is a special yet extensively studied case of metric spaces. This space representation is a coordinate system, with each data point represented as a vector with numerical values. The proximity of two points x_i and x_j is simply defined as their Euclidean distance $\|x_i - x_j\|$. This paper focuses on the d dimensional Euclidean space for defining the k nearest neighbors. Indeed, the Euclidean metric is favored by the spectral methods for data analysis because of some nice properties (see Section 2.1).

When $k = 1$, only the nearest neighbor for each data point is considered. This particular case is called the *all nearest neighbors* problem, which has been extensively studied in the literature. To compute the 1NN graph, Bentley [1980] proposed a multidimensional divide-and-conquer method that takes $O(n \log^{d-1} n)$ time, Clarkson [1983] presented a randomized algorithm with expected $O(c^d n \log n)$ time (for some constant c), and Vaidya [1989] introduced a deterministic worst-case $O((c'd)^d n \log n)$ time algorithm (for some constant c'). These algorithms are generally adaptable to $k > 1$. Thus, Paredes et al. [2006] presented a method to build a kNN graph, which was empirically studied and reported to require $O(n^{1.27})$ distance calculations for low dimensional data and $O(n^{1.90})$ calculations for high dimensional data. Meanwhile, several parallel algorithms have also been developed [Callahan, 1993, Callahan and Kosaraju, 1995, Plaku and Kavraki, 2007]. Despite the rich literature, efficient algorithms for high dimensional data are still under-explored. In this paper we propose two methods that are especially effective in dealing with high dimensional data.

Note that the problem of constructing a kNN graph is different from the problem of *nearest neighbor(s) search* [see, e.g., Indyk, 2004, Liu et al., 2004, Shakhnarovich et al., 2006, and references therein], where given a set of data points, the task is to find the k nearest points for any query point. Usually, the nearest neighbors search problem is handled by first building a data structure (such as a search tree) for the given points in a preprocessing phase. Then, queries can be answered efficiently by exploiting the search structure. Of course, the construction of a kNN graph can be viewed as a nearest neighbors search problem where each data point itself is a query. Considerations related to an effective exploitation of this viewpoint in practice will not be explored in the present paper and will be left for future investigation.

The rest of the paper is organized as follows. Section 2 introduces the two proposed methods for computing the kNN graphs, and Section 3 analyzes their time complexities. Then, we show a few experiments to demonstrate the effectiveness of the methods in Section 4, and discuss two applications in Section 5. Finally, conclusions are given in Section 6.

2 Divide and Conquer kNN

Our general framework for computing an approximate kNN graph is as follows: We divide the set of data points into subsets (possibly with overlaps), recursively compute the (approximate) kNN graphs for the subsets, then conquer the results into a final graph. This divide and conquer framework can clearly be separated in two distinct sub-problems: how to divide and how to conquer. The conquer step is simple: If a data point belongs to more than one subsets, then its k nearest neighbors are selected from its neighbors in each subset. However, the divide step can be implemented in many different ways, resulting in different qualities of graphs. We propose two methods in the following. These methods are based on a spectral bisection technique [Boley, 1998, Juhász and Mályusz, 1980, Tritchler et al., 2005] which employs an inexpensive Lanczos procedure [Lanczos, 1950].

2.1 Spectral Bisection

Let the data matrix

$$X = [x_1, \ldots, x_n] \in \mathbb{R}^{d\times n}$$

have each column as a data point. (By abuse of notation we also use X to denote the set of data points when the context is clear.) A typical spectral bisection technique is to split the centered data

$$\hat{X} = [\hat{x}_1, \ldots, \hat{x}_n] = X - ce^T,$$

where c is the centroid and e is a column of all ones, into halves using a hyperplane. Let (σ, u, v) denote the largest singular triplet of $\hat{X}$ with

$$u^T\hat{X} = \sigma v^T. \tag{1}$$

Then, the hyperplane is defined as $\langle u, x\rangle = 0$, i.e., it splits the set of data points into two subsets:

$$X_+ = \{x_i \mid u^T\hat{x}_i \geq 0\} \quad \text{and} \quad X_- = \{x_i \mid u^T\hat{x}_i < 0\}. \tag{2a}$$

This hyperplane is known to maximize the sum of squared distances between the centered points $\hat{x}_i$ to the hyperplane that passes through the origin. This is because for any hyperplane $\langle w, x\rangle = 0$ the squared sum is

$$\sum_{i=1}^{n}(w^T\hat{x}_i)^2 = \left\|w^T\hat{X}\right\|_2^2 \leq \left\|\hat{X}\right\|_2^2 = \sigma^2,$$

while $w = u$ achieves the equality.

By the property of the SVD (Equation (1)), this bisection technique is equivalent to splitting the set by the following criterion:

$$X_+ = \{x_i \mid v_i \geq 0\} \quad \text{and} \quad X_- = \{x_i \mid v_i < 0\}, \tag{2b}$$

where v_i is the i-th entry of v. If it is preferred that the sizes of the two subsets be balanced, an alternative is to replace the above criterion by

$$X_+ = \{x_i \mid v_i \geq \mathrm{m}(v)\} \quad \text{and} \quad X_- = \{x_i \mid v_i < \mathrm{m}(v)\}, \tag{3}$$

where $\mathrm{m}(v)$ represents the median of the entries of v.

The largest singular triplet (σ, u, v) of $\hat{X}$ can be computed using the Lanczos algorithm [Lanczos, 1950, Berry, 1992]. In short, we first compute an orthonormal basis of the Krylov subspace $\mathrm{span}\{q_1, (\hat{X}^T\hat{X})q_1, \ldots, (\hat{X}^T\hat{X})^{s-1}q_1\}$ for an arbitrary initial unit vector q_1 and a small integer s. Let the basis vectors form an orthogonal matrix

$$Q = [q_1, \ldots, q_s].$$

An equality resulted from this computation is

$$Q^T(\hat{X}^T\hat{X})Q = T,$$

where T is a symmetric tridiagonal matrix of size $s \times s$. Then we compute the largest eigenvalue θ and corresponding eigenvector y of T:

$$Ty = \theta y.$$

Therefore, θ (resp. Qy) is an approximation to the largest singular value (resp. right singular vector) of $\hat{X}$. Only a small value for s, say 5, is needed to yield a very accurate approximation. The computation of the orthonormal basis takes time $\Theta(sdn)$, where (recall that) d and n are the dimensions of $\hat{X}$. The time to compute θ and y is negligible since T is symmetric tridiagonal and s is very small. Hence the time for computing the largest singular triplet of $\hat{X}$ is bounded by $O(sdn)$.

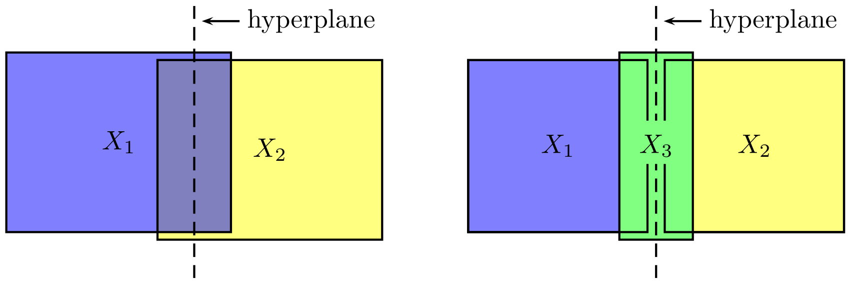
2.2 The Divide Step: Two Methods

Using the above bisection technique, the divide step for computing an approximate kNN graph can be performed in two ways. The first way, called the *overlap* method, divides the current set into two overlapping subsets. The second way, called the *glue* method, divides the current set into two disjoint subsets with a third gluing set. Details are as follows.

2.2.1 The Overlap Method

In this method, we divide the set X into two overlapping subsets X_1 and X_2:

$$\begin{cases} X_1 \cup X_2 = X, \\ X_1 \cap X_2 \neq \emptyset. \end{cases} \tag{4}$$



(a) The overlap method.

(b) The glue method.

Figure 1: Two methods to divide a set X into subsets.

Since σv_i is the signed distance from $\hat{x}_i$ to the hyperplane, let the set V be defined as

$$V = \{|v_i| \mid i = 1, 2, \ldots, n\}. \tag{5}$$

Then we use the following criterion to form X_1 and X_2:

$$X_1 = \{x_i \mid v_i \geq -h_\alpha(V)\} \quad \text{and} \quad X_2 = \{x_i \mid v_i < h_\alpha(V)\}, \tag{6}$$

where $h_\alpha(\cdot)$ is a function that returns the element which is only larger than $(100\alpha)\%$ of the elements in the input set. The purpose of criterion (6) is to ensure that the two subsets overlap $(100\alpha)\%$ of the data, i.e.,

$$|X_1 \cap X_2| = \lceil \alpha |X| \rceil .$$

See Figure 1(a) for an illustration.

2.2.2 The Glue Method

In this method, we divide the set X into two disjoint subsets X_1 and X_2 with a gluing subset X_3:

$$\begin{cases} X_1 \cup X_2 = X, \\ X_1 \cap X_2 = \emptyset, \\ X_1 \cap X_3 \neq \emptyset, \\ X_2 \cap X_3 \neq \emptyset. \end{cases} \tag{7}$$

The criterion is:

$$\begin{gathered} X_1 = \{x_i \mid v_i \geq 0\}, \quad X_2 = \{x_i \mid v_i < 0\}, \\ X_3 = \{x_i \mid -h_\alpha(V) \leq v_i < h_\alpha(V)\}. \end{gathered} \tag{8}$$

Note that the gluing subset X_3 in this method is exactly the intersection of the two subsets in the overlap method. Hence, X_3 also contains $(100\alpha)\%$ of the data. See Figure 1(b) for an illustration.

2.3 Refinement

In order to improve the quality of the resulted kNN graph, during each recursion after the conquer step, the graph can be refined at a small cost. The idea is to update the k nearest neighbors for each point by selecting from a pool consisting of its neighbors and the neighbors of its neighbors. Formally, for each point x, we re-select its k nearest neighbors from

$$\big(A = \{x_i \mid x_i \text{ is a neighbor of } x\}\big) \cup \{x_j \mid x_j \text{ is a neighbor of } x_i \in A\}.$$

2.4 The Algorithms

We are ready to present the complete algorithms for both methods. See Algorithms 1 and 2. Both algorithms share many similarities: They both fall in the framework of divide and conquer; they both call the brute-force procedure kNN-BruteForce to compute the graph when the size of the set is smaller than a threshold (n_k); they both recursively call themselves on smaller subsets; and they both employ a Conquer procedure to merge the graphs computed for the subsets and a Refine procedure to refine the graph during each recursion. The difference is that Algorithm 1 calls Divide-Overlap to divide the set into two subsets (Section 2.2.1), while Algorithm 2 calls Divide-Glue to divide the set into three subsets (Section 2.2.2).

Algorithm 1 Approximate kNN Graph Construction: The Overlap Method

function $G =$ kNN-Overlap(X, k, α)
    if $|X| < n_k$ **then**
        $G \leftarrow$ kNN-BruteForce(X, k)
    else
        $(X_1, X_2) \leftarrow$ Divide-Overlap(X, α) ▷ Section 2.2.1
        $G_1 \leftarrow$ kNN-Overlap(X_1, k, α)
        $G_2 \leftarrow$ kNN-Overlap(X_2, k, α)
        $G \leftarrow$ Conquer(G_1, G_2) ▷ Section 2, beginning
        Refine(G) ▷ Section 2.3
    end if
end function

2.5 A Hash Table to Store the Computed Distances

The brute-force method computes $\Theta(n^2)$ pairs of distance, each of which takes $\Theta(d)$ time. One advantage of our methods over the brute-force method is that the distance calculation can be significantly reduced thanks to the nature of the divide and conquer approach. The distances are needed/computed in: (1) the kNN-BruteForce procedure which computes all the pairwise distances and selects the k smallest ones for each point, (2) the Conquer procedure which selects k smallest distances from at most $2k$ candidates for each point, and (3) the Refine procedure which selects the k smallest distances from at most $k + k^2$ candidates for each point. Many of the distances computed from kNN-BruteForce and Refine are reused in Conquer and Refine, with probably more than once for some pairs. A naive way is

Algorithm 2 Approximate kNN Graph Construction: The Glue Method

```
function G = kNN-GLUE(X, k, α)
    if |X| < n_k then
        G ← kNN-BRUTEFORCE(X, k)
    else
        (X_1, X_2, X_3) ← DIVIDE-GLUE(X, α)          ▷ Section 2.2.2
        G_1 ← kNN-GLUE(X_1, k, α)
        G_2 ← kNN-GLUE(X_2, k, α)
        G_3 ← kNN-GLUE(X_3, k, α)
        G ← CONQUER(G_1, G_2, G_3)                   ▷ Section 2, beginning
        REFINE(G)                                    ▷ Section 2.3
    end if
end function
```

to allocate memory for an $n \times n$ matrix that stores all the computed distances to avoid duplicate calculations. However this consumes too much memory and is not necessary. A better approach is to use a hash table to store the computed distances. This will save a significant amount of memory, as a later experiment shows that only a small portion of the n^2 pairs are actually computed. Furthermore the computational time will not be affected since hash tables are efficient for both retrieving wanted items from and inserting new items into the table.

3 Complexity Analysis

A thorough analysis shows that the time complexities for the overlap method and the glue method are sub-quadratic (on n), and the glue method is always asymptotically faster than the overlap method. To this end we assume that in each divide step the subsets X_1 and X_2 (in both methods) are balanced. In practice this may not be the case, but the complexities in general hold. Hence, the time complexity T_{o} for the overlap method and T_{g} for the glue method satisfy the following recurrence relations:

$$T_{\mathrm{o}}(n) = 2T_{\mathrm{o}}((1+\alpha)n/2) + f(n), \tag{9}$$
$$T_{\mathrm{g}}(n) = 2T_{\mathrm{g}}(n/2) + T_{\mathrm{g}}(\alpha n) + f(n), \tag{10}$$

where $f(n)$ is the combined time for the divide, conquer, and refine steps.

3.1 The Complexity of f

The function $f(n)$ has the following three components.

(a) The time for the divide step. This includes the time to compute the largest singular triplet (σ, u, v) of the centered matrix $\hat{X}$, and the time to divide points into subsets X_1 and X_2 (in the overlap method) or subsets X_1, X_2 and X_3 (in the glue method). The former has been shown to be $O(sdn)$ in Section 2.1, while for the latter we can use a linear time selection method to find the value $h_\alpha(V)$. Therefore the overall time for this step is $O(sdn)$, where s is a very small constant.

(b) The time for the conquer step. This step only involves the points in $X_1 \cap X_2$ in the overlap method or X_3 in the glue method. For each of the αn points in these subsets, k nearest neighbors are chosen from at most $2k$ candidates. Therefore the time is $O(k\alpha n)$.

(c) The time for the refine step. For each point, k nearest neighbors are chosen from at most $k + k^2$ candidates. If all these distances need to be computed, the overall time is $O(k^2 dn)$. To the other extreme, if none of them are computed, the factor d can be omitted, which results in $O(k^2 n)$. In practice, by using a hash table, only a very small fraction of the $k + k^2$ distances are actually computed in this step. Hence, we can safely assume that the time is bounded by $O(k^2 n)$.

Since d is assumed to be large compared with k, and s is a small constant, we conclude that $f(n)$ is bounded by $O(dn)$.

3.2 The Complexities of $T_\mathbf{o}$ and $T_\mathbf{g}$

By substituting $f(n) = O(dn)$ into (9), we immediately have the following theorem.

Theorem 1. *The time complexity for the overlap method is*

$$T_o(n) = \Theta(dn^{t_o}), \tag{11}$$

where

$$t_o = \log_{2/(1+\alpha)} 2 = \frac{1}{1 - \log_2(1+\alpha)}. \tag{12}$$

Proof. This follows from the Master Theorem [Cormen et al., 2001, Chapter 4.3]. □

In order to establish the time complexity for the glue method, we need the following lemma.

Lemma 2. *The recurrence relation*

$$T(n) = 2T(n/2) + T(\alpha n) + n$$

with $T(1) = 1$ has a solution

$$T(n) = \left(1 + \frac{1}{\alpha}\right) n^t - \frac{n}{\alpha}$$

where t is the solution to the equation

$$\frac{2}{2^t} + \alpha^t = 1. \tag{13}$$

Proof. A direct verification proves this lemma. □

Now we are ready to have the following result.

Theorem 3. *The time complexity for the glue method is*

$$T_g(n) = \Theta(dn^{t_g}/\alpha), \tag{14}$$

where t_g is the solution to (13).

Proof. This immediately follows from Lemma 2 by noting that the solution to (13) is large than 1. □

To have an idea of the sizes of t_{o} and t_{g}, for example when $\alpha = 0.1$, $t_{\mathrm{o}} = 1.16$ while $t_{\mathrm{g}} = 1.12$. Other examples and an extensive study of these two exponents are proposed next.

Assuming the same value α is used, which of the two methods is asymptotically faster? This is equivalently to asking the question: Which of the two exponents, t_{o} or t_{g}, is smaller? For this, we need the following two lemmas.

Lemma 4. *When* $0 < x < 1$,

$$\log_2(1-x^2) > \Big(\log_2(1-x)\Big)\Big(\log_2(1+x)\Big).$$

Proof. By Taylor expansion,

$$\begin{aligned}
&\Big(\ln(1-x)\Big)\Big(\ln(1+x)\Big)\\
&= \left(-\sum_{n=1}^{\infty}\frac{x^n}{n}\right)\left(\sum_{n=1}^{\infty}\frac{(-1)^{n+1}x^n}{n}\right)\\
&= -\sum_{n=1}^{\infty}\left(\sum_{m=1}^{2n-1}\frac{(-1)^{m-1}}{m(2n-m)}\right)x^{2n}\\
&= -\sum_{n=1}^{\infty}\left(\frac{1}{2n}\left(\frac{1}{1}+\frac{1}{2n-1}\right)-\frac{1}{2n}\left(\frac{1}{2}+\frac{1}{2n-2}\right)+\cdots+\frac{(-1)^{n-1}}{n^2}\right.\\
&\qquad\qquad \left.+\cdots-\frac{1}{2n}\left(\frac{1}{2n-2}+\frac{1}{2}\right)+\frac{1}{2n}\left(\frac{1}{2n-1}+\frac{1}{1}\right)\right)x^{2n}\\
&= -\sum_{n=1}^{\infty}\left(\frac{1}{n}\left(1-\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{2n-1}\right)\right)x^{2n}\\
&< -\sum_{n=1}^{\infty}\left(\frac{\ln 2}{n}\right)x^{2n}\\
&= (\ln 2)\cdot\ln(1-x^2).
\end{aligned}$$

By changing the bases of the logarithms, the inequality in the lemma holds. □

Lemma 5. *The following inequality*

$$\log_2(ab) > (\log_2 a)(\log_2 b)$$

holds whenever $0 < a < 1 < b < 2$ *and* $a + b \geq 2$.

Proof. By using $b \geq 2 - a$, we have the following two inequalities

$$\log_2(ab) \geq \log_2(a(2-a)) = \log_2(1-(1-a))(1+(1-a)) = \log_2(1-(1-a)^2)$$

and

$$(\log_2 a)(\log_2 b) \leq (\log_2 a)(\log_2(2-a)) = \log_2(1-(1-a)) \times \log_2(1+(1-a)).$$

Then by applying Lemma 4 on $1 - a = x$, we have

$$\log_2(1 - (1 - a)^2) > \log_2(1 - (1 - a)) \times \log_2(1 + (1 - a)).$$

Thus, the inequality in the lemma holds. □

Remark 1. Note that Lemma 4 is now a special case of Lemma 5 by letting $a = 1-x$ and $b = 1 + x$.

Now we are ready to prove that the glue method is always asymptotically faster than the overlap method for any choice of α.

Theorem 6. *When $0 < \alpha < 1$, the exponents t_o in Theorem 1 and t_g in Theorem 3 obey the following relation:*

$$1 < t_g < t_o. \tag{15}$$

Proof. From Theorem 3 we have

$$t_\mathrm{g} = 1 - \log_2(1 - \alpha^{t_\mathrm{g}}) > 1.$$

Then

$$\begin{aligned} t_\mathrm{o} - t_\mathrm{g} &= \frac{1}{1 - \log_2(1 + \alpha)} - 1 + \log_2(1 - \alpha^{t_\mathrm{g}}) \\ &= \frac{\log_2(1 + \alpha) + \log_2(1 - \alpha^{t_\mathrm{g}}) - \log_2(1 + \alpha) \times \log_2(1 - \alpha^{t_\mathrm{g}})}{1 - \log_2(1 + \alpha)}. \end{aligned}$$

Since $0 < \alpha < 1$, the denominator $1 - \log_2(1 + \alpha)$ is positive. By Lemma 5 the numerator is also positive. Hence $t_\mathrm{o} > t_\mathrm{g}$. □

We remark that when $\alpha > \sqrt{2} - 1 \approx 0.41$, $t_\mathrm{o} > 2$, which means that the overlap method is asymptotically slower than the brute-force method. Hence, a large α (> 0.41) may not be useful in practice. Figure 2 plots the two curves for t_o and t_g, and Table 1 lists some of their values at different α's.

α	0.05	0.10	0.15	0.20	0.25	0.30
t_o	1.08	1.16	1.25	1.36	1.47	1.61
t_g	1.06	1.12	1.17	1.22	1.27	1.33

Table 1: The values of t_o and t_g at different α's.

4 Experiments

In this section we show a few experimental results to illustrate the running times of the two proposed methods compared with the brute-force method, and the quality of the resulted graphs.

Figure 3 plots the running times versus the dimension d and the number of data points n on randomly generated points. Since the distribution of the data should have little impact on the running times of the methods, we use randomly generated data for this experiment. From Figure 3(a), it is clear that the running time is linear

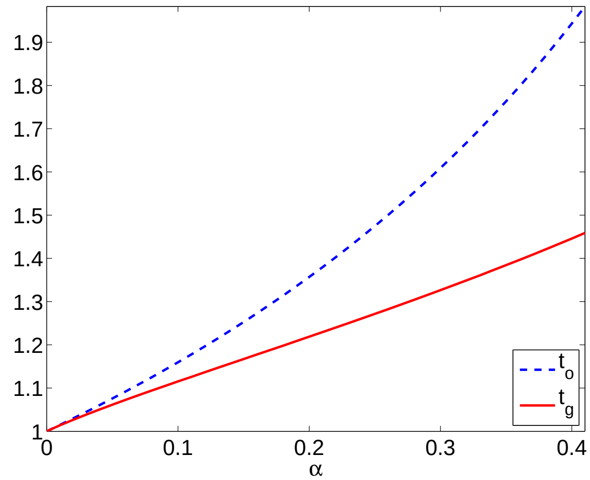
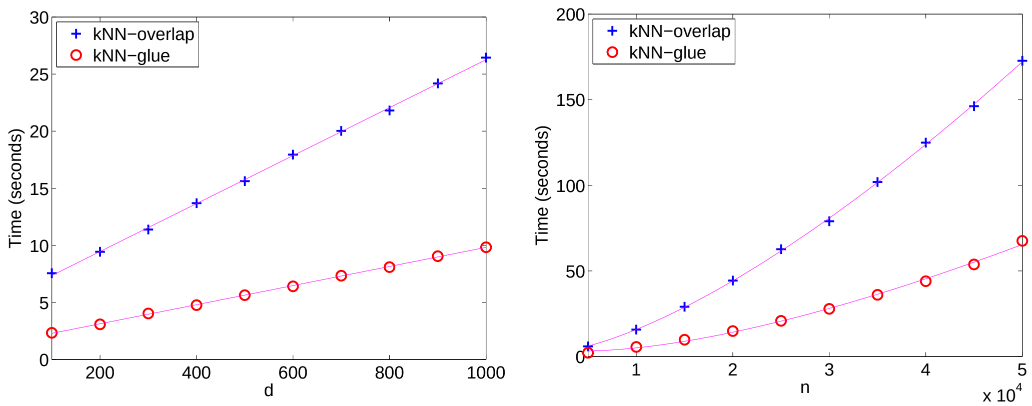


Figure 2: The curves of $t_{\rm o}$ and $t_{\rm g}$.



(a) $n = 10000$, $\alpha = 0.2$, d varying

(b) $d = 500$, $\alpha = 0.2$, n varying

Figure 3: The running times for randomly generated data.

with respect to the dimension d. This is expected from the complexity analysis. In Figure 3(b), we use $\alpha = 0.2$, which corresponds to the theoretical values $t_{\rm o} = 1.36$ and $t_{\rm g} = 1.22$. We use curves in the form $c_1 n^{1.36} + c_2 n + c_3$ and $c_4 n^{1.22} + c_5 n + c_6$ to fit the running times of the overlap method and the glue method, respectively. The fitted curves are also shown in Figure 3(b). It can be seen that the experimental results match the theoretical analysis quite well.

In another experiment, we use three real-life data sets to test the quality of the resulted kNN graphs: The FREY face video frames [Rowies and Saul, 2000], the MNIST digit images [LeCun et al., 1998], and the PIE face database [Sim et al., 2003]. These data sets are all image sets that are widely used in the literature in the areas of face recognition, dimensionality reduction, etc. The data are downloaded from: FREY: `http://www.cs.toronto.edu/~roweis/data.html`; MNIST: `http://yann.lecun.com/exdb/mnist`; PIE: `http://www.cs.uiuc.edu/homes/dengcai2/Data/FaceData.html`. For MNIST, we use only the test set. Ta-

	FREY	MNIST	PIE
# imgs	1,965	10,000	11,554
img size	20×28	28×28	32×32

Table 2: Image data sets.

ble 2 gives some characteristics of the data. The number of images is equal to n and the size of each image, i.e., the total number of pixels for each image, is the dimension d. The dimensions vary from about 500 to 1000.

The accuracies of the resulted graphs versus the running times are plotted in Figure 4. Each plotted point in the figure corresponds to a choice of α ($= 0.05, 0.10, 0.15, 0.20, 0.25, 0.30$). The running times of the brute-force method are also indicated. For different data sets we use different k's for the tests. The accuracy of an approximate kNN graph G' (with regard to the exact graph G) is defined as

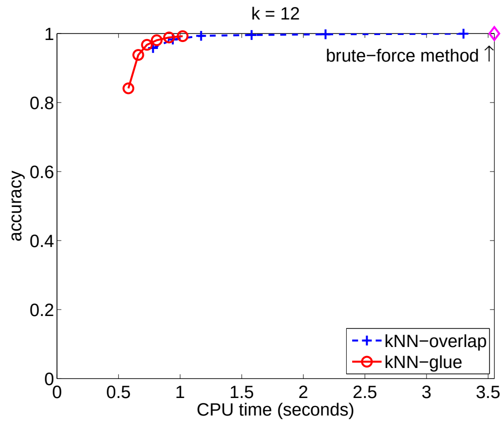
$$\frac{|E(G') \cap E(G)|}{|E(G)|},$$

where $E(\cdot)$ means the set of directed edges in the graph. It can be seen from Figure 4 that the larger α the more accurate the resulting graph. However, larger values of α lead to more time-consuming runs. In addition, the glue method is much faster than the overlap method for the same α, while the latter yields more accurate graphs than the former. The two methods are both significantly faster than the brute-force method when an appropriate α is chosen, and they can yield high quality graphs even when α is small.

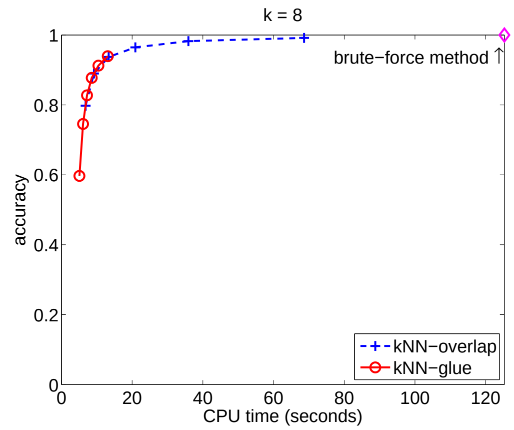
We also report the percentage of the distance calculations in our methods. For n data points, the brute-force method needs to compute distances between $n(n-1)/2$ pairs of points. As can be seen in Table 3, our methods compute only a small percentage of this number. The savings are even more significant when n is larger and k is smaller, e.g., $k = 5$ for the data set PIE. This is one of the key factors in making the proposed methods run significantly faster than the brute-force method.

	FREY ($k = 12$)		MNIST ($k = 8$)		PIE ($k = 5$)	
α	overlap	glue	overlap	glue	overlap	glue
0.05	6.10%	5.02%	1.20%	0.94%	0.45%	0.34%
0.10	6.59%	5.80%	1.42%	1.20%	0.59%	0.47%
0.15	7.37%	6.19%	1.73%	1.37%	0.78%	0.57%
0.20	8.34%	6.50%	2.21%	1.52%	1.05%	0.66%
0.25	9.67%	6.85%	2.92%	1.71%	1.48%	0.77%
0.30	11.54%	7.25%	4.02%	1.90%	2.06%	0.90%

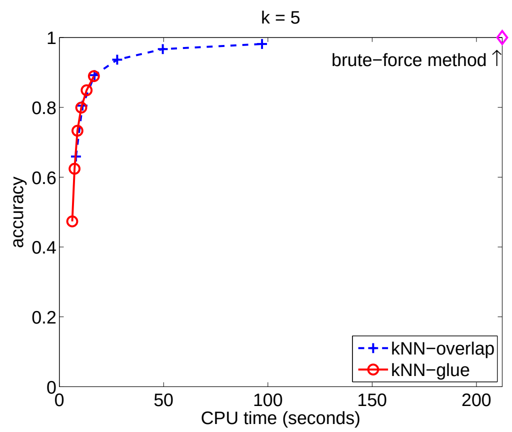
Table 3: Percentages of distances calculations, for different data sets, different methods, and different α's.



(a) FREY



(b) MNIST



(c) PIE

Figure 4: Graph accuracy versus running time for different data sets. Each plotted point corresponds to a choice of α.

5 Applications

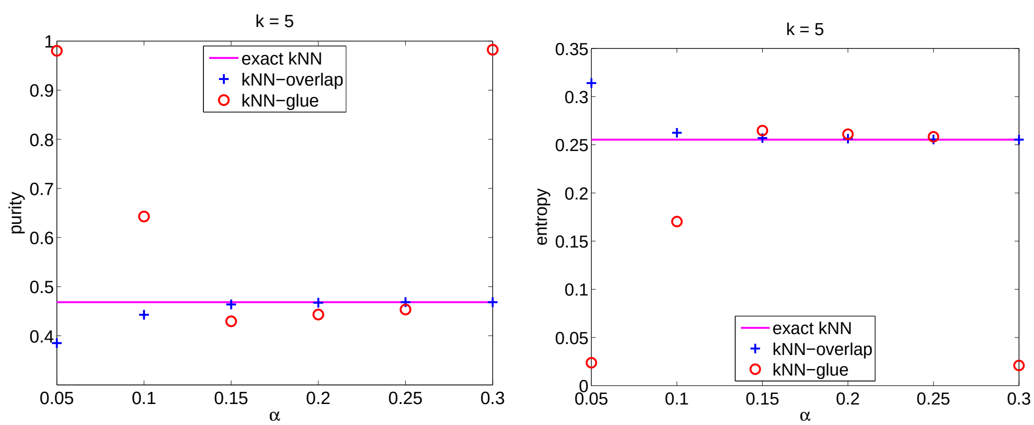
kNN graphs have been widely used in various data mining and machine learning applications. In this section, we discuss two scenarios where the approximate kNN graphs resulted from the proposed methods can be an effective replacement for the exact kNN graph.

5.1 Agglomerative Clustering

Agglomerative clustering [Ward, 1963] is a hierarchical clustering method that iteratively merges a pair of clusters with the lowest merge cost at present to a new cluster. Initially each data point is a cluster, and the algorithm terminates when the desired number of clusters are found (or when all the points belong to the single final cluster). A straightforward implementation of the method takes $O(n^3)$ time, since there are $O(n)$ iterations, each of which takes $O(n^2)$ time to find the pair that has the lowest merge cost [Shanbehzadeh and Ogunbona, 1997]. Fränti et al. [2006]

proposed to search for the pair from only the set of edges in the kNN graph of the clusters at each iteration, instead of from all the $O(n^2)$ pairs. With a delicate implementation using a double linked list, they showed that the overall running time of the clustering process reduces to $O(\tau n \log n)$, where τ is the number of nearest neighbor updates at each iteration. Their method greatly speedups the clustering process, while the clustering quality is not much degraded.

However, the quadratic time to create the initial kNN graph eclipses the improvement on the clustering time. One solution is to use an approximate kNN graph that can be efficiently created. Virmajoki and Fränti [2004] proposed a divide-and-conquer algorithm to create an approximate kNN graph, but their time complexity was overestimated. Our methods also follow the common framework of divide and conquer. However, they bring three substantial improvements over the previous work: (1) Two methods to perform the divide step are proposed; (2) an efficient way to compute the separating hyperplane, and (3) a detailed and rigorous analysis on the time complexity is provided. This analysis in particular makes the proposed methods practical, especially for high dimensional data (e.g., when d is at the magnitude of hundreds or thousands).



(a) Purity. (b) Entropy.

Figure 5: Agglomerative clustering using kNN graphs on the image data set PIE (68 human subjects).

We perform an experiment on the data set PIE with 68 classes (see Figure 5). Since class labels are known, we use the purity and the entropy [Zhao and Karypis, 2004] as the quality criteria. They are defined as

$$\text{Purity} = \sum_{i=1}^{q} \frac{n_i}{n} \text{Purity}(i), \qquad \text{where} \qquad \text{Purity}(i) = \frac{1}{n_i} \max_j \left(n_i^j \right),$$

and

$$\text{Entropy} = \sum_{i=1}^{q} \frac{n_i}{n} \text{Entropy}(i), \qquad \text{where} \qquad \text{Entropy}(i) = -\sum_{j=1}^{q} \frac{n_i^j}{n_i} \log_q \frac{n_i^j}{n_i}.$$

Here, q is the number of classes/clusters, n_i is the size of class i, and n_i^j is the number of class i data that are assigned to the j-th cluster. The purity and the entropy both range from 0 to 1. In general, a higher purity and/or a low entropy means a better clustering quality. It can be seen from Figure 5 that the qualities of the clusterings using the approximate kNN graphs are very close to that resulted from the exact graph, with some being even much better. It is interesting to note that the clustering results seem to have little correlation with the qualities of the graphs governed by the value α.

5.2 Dimensionality Reduction

Many dimensionality reduction methods, e.g., locally linear embedding (LLE) [Rowies and Saul, 2000], Laplacian eigenmaps [Belkin and Niyogi, 2003], locality preserving projections (LPP) [He and Niyogi, 2004], and orthogonal neighborhood preserving projections (ONPP) [Kokiopoulou and Saad, 2007], compute a low dimensional embedding of the data by preserving the local neighborhoods for each point. For example, in LLE, a weighted adjacency matrix W is first computed to minimize the following objective:

$$\mathcal{E}(W) = \sum_i \left\| x_i - \sum_{x_j \in N(x_i)} W_{ij} x_j \right\|^2$$

where $N(\cdot)$ means the neighborhood of a point. Then, given the computed W, a low dimensional embedding (y_i's) is computed such that it minimizes a similar objective:

$$\Phi(Y) = \sum_i \left\| y_i - \sum_{y_j \in N(x_i)} W_{ij} y_j \right\|^2 .$$

As another example, in Laplacian eigenmaps, the low dimensional embedding (y_i's) is computed so as to minimize the cost function:

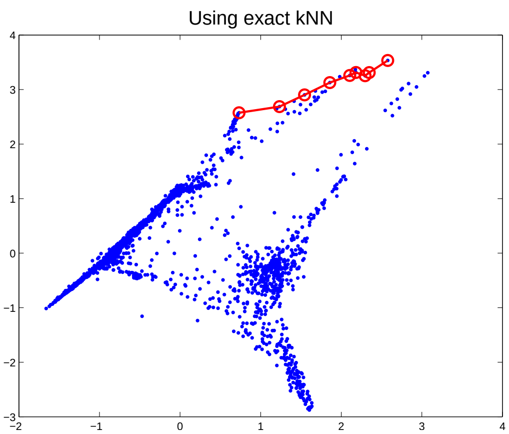
$$\Psi(Y) = \sum_{i,j} W_{ij} \left\| y_i - y_j \right\|^2 .$$

Here the weighted adjacency matrix W is explicitly defined as the heat kernel

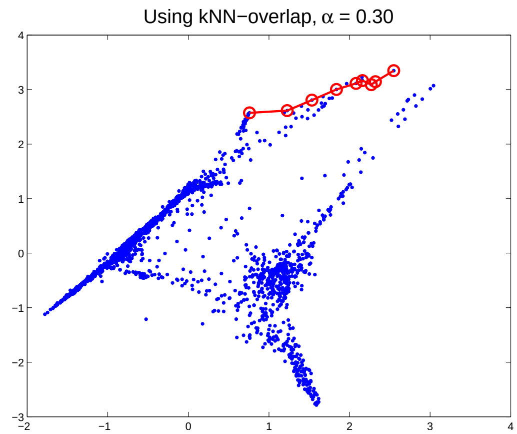
$$W_{ij} = \begin{cases} \exp(-\left\| x_i - x_j \right\|^2 / \sigma^2) & \text{if } x_i \in N(x_j) \text{ or } x_j \in N(x_i), \\ 0 & \text{otherwise.} \end{cases}$$

Despite the nicely motivated formulations for the above approaches and the elegant numerical linear algebraic solutions to the methods, these methods all begin with a rather expensive computation to obtain the neighborhood graph of the data. The methods discussed in this paper are suitable alternatives to the expensive brute-force approach to obtain the exact kNN graph, since the approximate graphs are accurate enough for the purpose of dimensionality reduction, while the time costs are significantly smaller. Figures 6 and 7 illustrate two examples.

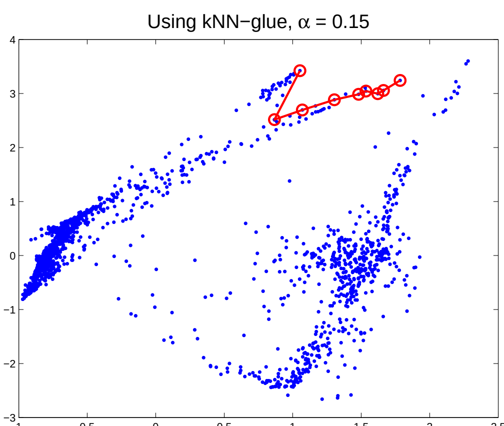
In Figure 6 are the plots of the dimensionality reduction results of LLE applied to the data set FREY, where we used $k = 12$ as that in Rowies and Saul [2000, Figure 3]. Figure 6(a) shows the result when using the exact kNN graph, while Figure 6(b) shows the result when using the approximate kNN graph by the overlap method with $\alpha = 0.30$. It is clear that the two results are almost identical. Figures 6(c) and 6(d) give two plots when the glue method is used. Although the embeddings are different from Figure 6(a), they also well represent the original data manifold. This can be seen by tracking the relative locations of the sequence of images (as shown in 6(e)) in the low dimensional space.



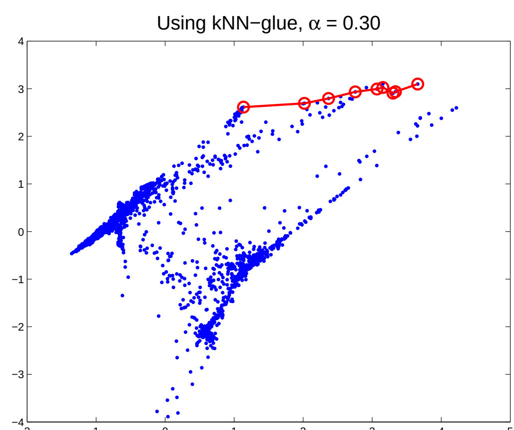
(a)



(b)



(c)



(d)

(e) Images on the red path. Facial expressions gradually change.

Figure 6: Dimensionality reduction on the data set FREY by LLE.

In Figure 7 are the plots of the dimensionality reduction results of Laplacian eigenmaps applied to the data set MNIST, where we used $k = 5$. Figure 7(a) shows the original result by using the exact kNN graph, while 7(b), 7(c) and 7(d) show the results by using the overlap method with $\alpha = 0.30$, the glue method with $\alpha = 0.15$, and the glue method with $\alpha = 0.30$, respectively. The four plots look similar, and they all show a clear clustering of the ten classes (digits from 0 to 9).

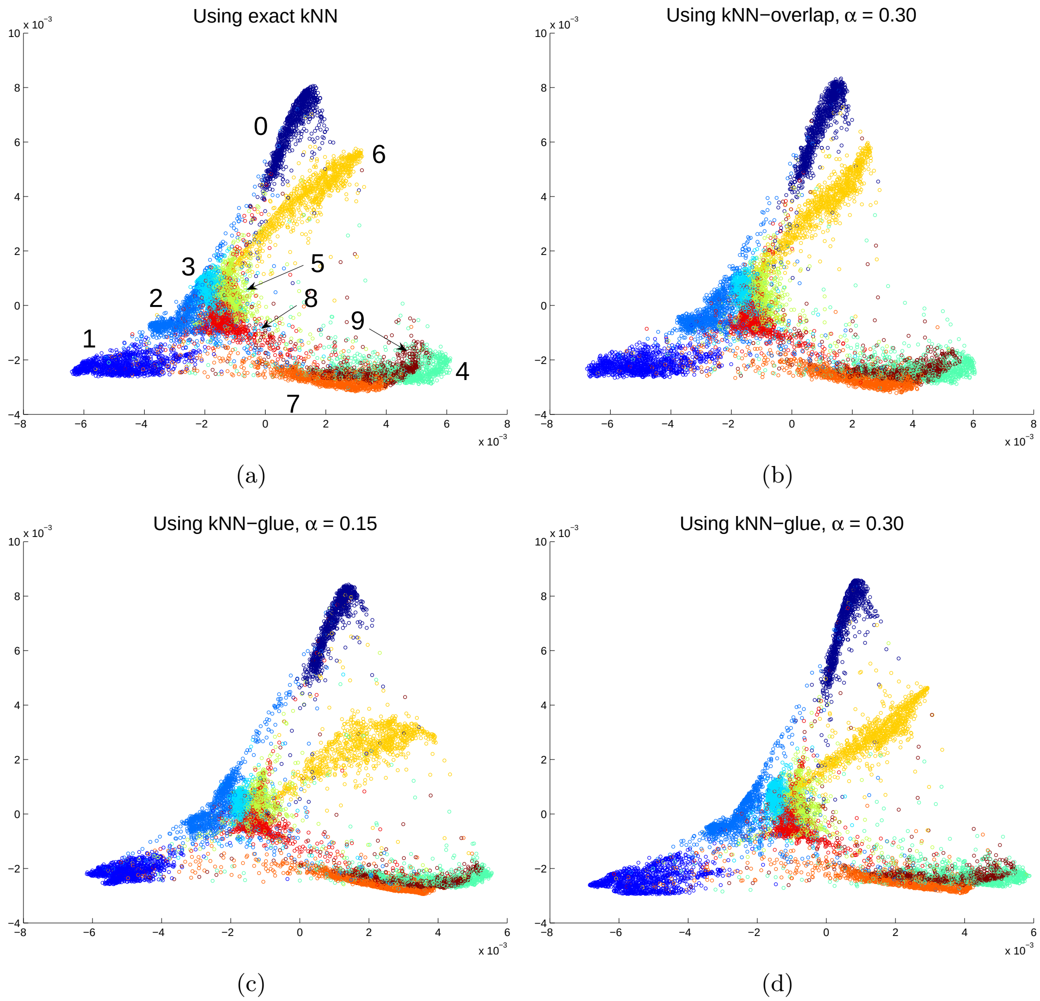


Figure 7: Dimensionality reduction on the data set MNIST by Laplacian eigenmaps.

6 Conclusions

We have proposed two sub-quadratic time methods under the framework of divide and conquer for computing approximate kNN graphs for high dimensional data sets. The running times of the methods, as well as the qualities of the resulted graphs, depend on an internal parameter that controls the overlap size of the subsets in the divide step. Experiments show that in order to obtain a high quality graph, a small overlap size is usually required which leads to a small exponent in the time complexity. The resulted approximate graphs have a wide range of applications as they can be safely used as alternatives to the exact kNN graph. We have shown two such sample examples: in agglomerative clustering and in dimensionality reduction. Thus, replacing the exact kNN graph construction with one produced by the methods proposed here, can significantly alleviate what currently constitutes one of the major bottlenecks in these applications.

References

M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computatioin*, 16(6):1373–1396, 2003.

J. Bentley. Multidimensional divide-and-conquer. *Comm. ACM*, 23:214–229, 1980.

J. Bentley, D. Stanat, and E. Williams. The complexity of finding fixed-radius near neighbors. *Information Processing Letters*, 6:209–213, 1977.

Michael W. Berry. Large scale sparse singular value computations. *International Journal of Supercomputer Applications*, 6(1):13–49, 1992.

D. L. Boley. Principal direction divisive partitioning. *Data Mining and Knowledge Discovery*, 2(4):324–344, 1998.

M. Brito, E. Chávez, A. Quiroz, and J. Yukich. Connectivity of the mutual k-nearest neighbor graph in clustering and outlier detection. *Statistics & Probability Letters*, 35:33–42, 1997.

Paul B. Callahan. Optimal parallel all-nearest-neighbors using the well-separated pair decomposition. In *Proc. 34th IEEE Symp. Foundations of Computer Science*, 1993.

Paul B. Callahan and S. Rao Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *J. ACM*, 42(1):67–90, 1995.

Bernard Chazelle. An improved algorithm for the fixed-radius neighbor problem. *Information Processing Letters*, 16:193–198, 1983.

Howie Choset, Kevin M. Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. The MIT Press, 2005.

Kenneth L. Clarkson. Fast algorithms for the all nearest neighbors problem. In *Proc. 24th Ann. IEEE Sympos. on the Found. Comput. Sci.*, pages 226–232, 1983.

Kenneth L. Clarkson. Nearest-neighbor searching and metric space dimensions. In Gregory Shakhnarovich, Trevor Darrell, and Piotr Indyk, editors, *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice*. The MIT Press, 2006.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2nd edition, 2001.

Belur V. Dasarathy. Nearest-neighbor approaches. In Willi Klosgen, Jan M. Zytkow, and Jan Zyt, editors, *Handbook of Data Mining and Knowledge Discovery*, pages 88–298. Oxford University Press, 2002.

Pasi Fränti, Olli Virmajoki, and Ville Hautamäki. Fast agglomerative clustering using a k-nearest neighbor graph. *IEEE Trans. Pattern Anal. Machine Intell.*, 28 (11):1875–1881, 2006.

X. He and P. Niyogi. Locality preserving projections. In *Advances in Neural Information Processing Systems 16 (NIPS)*, 2004.

Piotr Indyk. Nearest neighbors in high-dimensional spaces. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*. CRC Press LLC, 2nd edition, 2004.

F. Juhász and K. Mályusz. *Problems of cluster analysis from the viewpoint of numerical analysis*, volume 22 of *Colloquia Mathematica Societatis Janos Bolyai*. North-Holland, Amsterdam, 1980.

E. Kokiopoulou and Y. Saad. Orthogonal neighborhood preserving projections: A projection-based dimensionality reduction technique. *IEEE Trans. Pattern Anal. Machine Intell.*, 29(12):2143–2156, 2007.

C. Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *J. Res. Nat. Bur. Stand.*, 45:255–282, 1950.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

T. Liu, A. W. Moore, A. Gray, and Ke Yang. An investigation of practical approximate nearest neighbor algorithms. In *Proc. of Neural Information Processing Systems (NIPS 2004)*, 2004.

Rodrigo Paredes, Edgar Chávez, Karina Figueroa, and Gonzalo Navarro. Practical construction of k-nearest neighbor graphs in metric spaces. In *Proc. 5th Workshop on Efficient and Experimental Algorithms (WEA'06)*, 2006.

Erion Plaku and Lydia E. Kavraki. Distributed computation of the $k$nn graph for large high-dimensional point sets. *J. Parallel Distrib. Comput.*, 67(3):346–359, 2007.

S. T. Rowies and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290:2323–2326, 2000.

Jagan Sankaranarayanan, Hanan Samet, and Amitabh Varshney. A fast all nearest neighbor algorithm for applications involving large point-clouds. *Computers and Graphics*, 31(2):157–174, 2007.

Lawrence K. Saul and Sam T. Roweis. Think globally, fit locally: unsupervised learning of low dimensional manifolds. *J. Machine Learning Research*, 4:119–155, 2003.

Gregory Shakhnarovich, Trevor Darrell, and Piotr Indyk, editors. *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice*. The MIT Press, 2006.

J. Shanbehzadeh and P. O. Ogunbona. On the computational complexity of the LBG and PNN algorithms. *IEEE Trans. Image Process.*, 6(4):614–616, 1997.

T. Sim, S. Baker, and M. Bsat. The CMU pose, illumination, and expression database. *IEEE Trans. Pattern Anal. Machine Intell.*, 25(12):1615–1618, 2003.

J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

D. Tritchler, S. Fallah, and J. Beyene. A spectral clustering method for microarray data. *Comput. Statist. Data Anal.*, 49:63–76, 2005.

P. M. Vaidya. An $O(n \log n)$ algorithm for the all-nearest-neighbors problem. *Discrete Comput. Geom.*, 4:101–115, 1989.

Olli Virmajoki and Pasi Fränti. Divide-and-conquer algorithm for creating neighborhood graph for clustering. In *Proc. of the 17th International Conference on Pattern Recognition (ICPR'04)*, 2004.

Joe H Ward. Hierarchical grouping to optimize an objective function. *J. Am. Statistical Assoc.*, 58(301):236–244, 1963.

Bin Zhang and Sargur N. Srihari. A fast algorithm for finding k-nearest neighbors with non-metric dissimilarity. In *Proc. of the Eighth International Workshop on Frontiers in Handwriting Recognition (IWFHR'02)*, 2002.

Ying Zhao and George Karypis. Empirical and theoretical comparisons of selected criterion functions for document clustering. *Machine Learning*, 55(3):311–331, 2004.