Int J Comput Vis
DOI 10.1007/s11263-011-0496-1

Dense Neighborhoods on Affinity Graph

Hairong Liu · Xingwei Yang · Longin Jan Latecki · Shuicheng Yan

Received: 16 September 2010 / Accepted: 8 September 2011

Abstract In this paper, we study the problem of how to reliably compute neighborhoods on affinity graphs. The k-nearest neighbors (kNN) is one of the most fundamental and simple methods widely used in many tasks, such as classification and graph construction. Previous research focused on how to efficiently compute kNN on vectorial data. However, most real-world data have no vectorial representations, and only have affinity graphs which may contain unreliable affinities. Since the kNN of an object o is a set of k objects with the highest affinities to o, it is easily disturbed by errors in pairwise affinities between o and other objects, and also it cannot well preserve the structure underlying the data. To reliably analyze the neighborhood on affinity graphs, we define the *k-dense neighborhood* (kDN), which considers all pairwise affinities within the neighborhood, i.e., not only the affinities between o and its neighbors but also between the neighbors. For an object o, its kDN is a set $k\text{DN}(o)$ of k objects which maximizes the sum of all pairwise affinities of objects in the set $\{o\} \cup k\text{DN}(o)$. We analyze the properties of kDN, and propose an efficient algorithm to compute it. Both theoretic analysis and experimental results on shape retrieval, semi-supervised learning, point set matching and data clustering show that kDN significantly outperforms kNN on affinity graphs, especially when many pairwise affinities are unreliable.

Keywords Nearest neighbor · Affinity graph · Semi-supervised learning · Clustering

H. Liu (✉) · S. Yan
National University of Singapore, Singapore, Singapore
e-mail: lhrbss@gmail.com

S. Yan
e-mail: eleyans@nus.eud.sg

X. Yang · L.J. Latecki
Temple University, Philadelphia, USA

X. Yang
e-mail: happyyxw@gmail.com

L.J. Latecki
e-mail: latecki@temple.edu

1 Introduction

kNN is a fundamental method widely used in various tasks (Cover and Hart 1967; Horton and Nakai 1997; Korn et al. 1996; Connor 2010). For an object, its kNN is a set consisting of its k closest objects, which usually share the same properties with the given object. For example, in classification task (Cover and Hart 1967), the kNN of an object usually shares the same label as the given object with high probability.

Since vector is the most convenient and popular data form for machine processing, there exists extensive literature on efficient computation of kNN on vectorial data (Arya et al. 1998; Indyk and Motwani 1998; Connor 2010; Chan 1998). For data represented as vectors in certain dimension, there are two main problems to study: (1) how to measure pairwise distances, and (2) how to efficiently find the kNN of any query point based on a selected distance metric.

Although vectorial representation is very popular, most real-world data actually cannot be naturally represented in vectorial form, or may suffer severe information loss if transformed into vectorial form. For an object, its kNN only relies on the affinities (or distances) of this object to all other objects, thus kNN can be applied to different kinds of data, provided that we can compute the affinity between each pair

Published online: 27 September 2011

of objects. In real-world applications, we may naturally obtain the affinity graph of a dataset, or it is convenient to compute the pairwise affinities and construct the affinity graph. For a dataset O with n objects, that is, $O = \{o_1, o_2, \ldots, o_n\}$, $f : O \times O \to R$ is the affinity function over a pair of objects. We can construct an affinity graph G with every vertex representing an object and f is the weight function over edge set. O can represent a large range of objects, such as features in images, candidate correspondences in correspondence problem and nodes on the Internet. At the same time, f can take different forms in different applications, and in most of cases, it has no parametric form. For example, in image matching or shape matching fields (Ling and Jacobs 2007), f represents the matching score, which is computed by some matching method, using certain features. Our definition of f includes kernel functions, but we do not restrict f to have only non-negative values, in fact, f may also take negative values, which usually represent dissimilarity. In general setting, when $a \neq b$, $f(a, b)$ and $f(b, a)$ may have different meanings, thus, for every pair of objects, there may exist two directed edges between them, weighted by $f(a, b)$ and $f(b, a)$, respectively. At the same time, for every object a, we assume there exists a self link to itself, weighted by $f(a, a)$.

Based on a given affinity graph G, it is easy to obtain the kNN of every vertex (object). However, in real applications, the pairwise affinities are usually unreliable. First, $f(a, b)$, the affinity between object a and object b, may result from accidental relation between object a and object b, such as an accidental communication on the Internet. Second, the pairwise affinities are generally computed independently. For example, in image matching, $f(a, b)$ and $f(a, c)$, the similarity score between image a and image b and the similarity score between image a and image c, respectively, are usually computed independently. Because of the imperfectness of matching algorithms, the computed affinities are usually inaccurate, sometimes even wrong and incompatible with each other. Third, in some situations, $f(a, b)$ only conveys partial information, that is, large (small) value of $f(a, b)$ only indicates that object a and object b have very high probability of being similar (dissimilar), and does not mean that they are really similar (dissimilar). For example, if two images share some common properties, they probably, but not necessarily, contain similar objects. Obviously, on such affinity graphs with unreliable affinities, kNN is not reliable, and may classify dissimilar objects as similar ones.

To our best knowledge, no efforts have ever been devoted to the problem of how to reliably compute neighborhoods on affinity graphs. Since the kNN of an object only relies on the affinities of this object to all other objects, kNN has no mechanism to resist errors in pairwise affinities and to preserve structure underlying the data. Intuitively, two nearest neighbors of an object should also be similar, which is usually naturally satisfied for vectorial data if the distance measure satisfies the triangle inequality, but is frequently violated on affinity graphs with unreliable affinities. Inspired by this simple phenomenon, we propose to consider the *total affinity*, which is defined as $S(O) = \sum_{i=1}^{n} \sum_{j=1}^{n} f(o_i, o_j)$ for a set $O = \{o_1, o_2, \ldots, o_n\}$ and define the k-dense neighborhood as follows:

Definition 1 In the dataset O, the *k-dense neighborhood* (kDN) of an object o, is a set $k\text{DN}(o)$ with k objects, $k\text{DN}(o) = \{o_{p_1}, o_{p_2}, \ldots, o_{p_k}\} \subset O$ and $o \notin k\text{DN}(o)$, such that the total affinity of the set $U = \{o\} \cup k\text{DN}(o)$ reaches maximum. That is,

$$k\text{DN}(o) = \arg\max_{T} S(\{o\} \cup T),$$
$$T \subset O,\ o \notin T \text{ and } |T| = k, \tag{1}$$

where $|T|$ is the cardinality of the set T.

The name dense neighborhood is justified by the fact that $k\text{DN}(o)$, for a given node o, is locally the densest subgrpah containing o and its k neighbors. The key difference to kNN is the fact that the k neighbors in $k\text{DN}(o)$ are not necessarily the k nearest neighbors.

Obviously, for an object o, $k\text{DN}(o)$ considers not only the affinities between o and the objects in $k\text{DN}(o)$, but also all pairwise affinities between objects in $k\text{DN}(o)$, and it is the ensemble effect of all pairwise affinities within the set $U = \{o\} \cup k\text{DN}(o)$. Thus, kDN(o) represents a locally most coherent cluster or locally the densest subgraph. For affinity graphs with unreliable affinities, although some pairwise affinities are unreliable, the ensemble of all pairwise affinities within U is surprisingly reliable, which explains why kDN is significantly more reliable than kNN. Even for affinity graphs constructed from vectorial data, kDN is usually better than kNN, as illustrated later, since it is more coherent than kNN.

For certain applications, the affinity function f may be asymmetric, that is, $f(a, b) \neq f(b, a)$ for certain pair (a, b). In such situation, we can replace f by a new symmetric function $f' : O \times O \to R$, where $\hat{f}(a, b) = \hat{f}(b, a) = \frac{f(a,b)+f(b,a)}{2}$. Since replacing f by $\hat{f}$ does not change the total affinity of any cluster of objects, according to the Definition 1, it does not change the kDN of any object, either. Thus, in the following text, we only consider the symmetric affinity function f. At the same time, in graphical illustration, for clarity, we only draw one edge with weight $f(a, b)$ between two objects a and b, and we also ignore edges with small weights.

Figure 1 demonstrates the difference between kNN and kDN over an affinity graph with unreliable affinities. According to the definition of kNN, $3\text{NN}(o) = \{a, c, q\}$; however, since the object a has very low affinities with the ob-

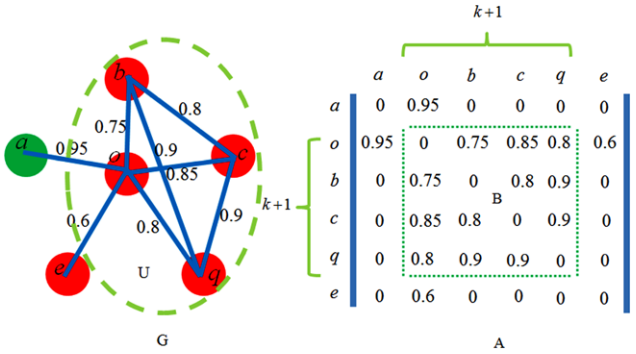

Fig. 1 kNN vs. kDN on a affinity graph with some unreliable affinities. $k = 3$, 3NN(o)= $\{a, c, q\}$, while 3DN(o)= $\{b, c, q\}$. U is the cluster containing b, c, q and o. $S(U)$ is the sum of all pairwise affinities within the cluster U, thus it is also the sum of all elements in the submatrix B, which is in fact the adjacency matrix of the subgraph U

jects c and q, the high affinity between a and o is probably wrong. According to our definition of kDN, 3DN(o) = $\{b, c, q\}$, which obviously form a more coherent cluster, together with o. Intuitively speaking, 3DN(o) is a local densest subgraph which contains o and other 3 vertices. From the viewpoint of the adjacency matrix A, it is identical to finding a $(3 + 1) \times (3 + 1)$ dense sub-matrix B with the same row and column index set.

Since kDN(o) represents a local coherent cluster around the object o, the kDNs of all objects then form an atlas of the whole dataset, which offers a useful tool to uncover patterns underlying the data. As will be shown afterwards, kDN is a very versatile tool in many tasks, such as cluster analysis and classification. While kNN is very useful and popular for vectorial data, kDN makes significant progress for neighborhood analysis on affinity graphs.

The main contributions of this paper are as follows. (1) We define the concept of kDN, analyze its properties and reveal the underlying mechanism on its robustness to unreliable affinities. (2) We propose an efficient approximation algorithm to compute the kDNs over affinity graphs. (3) We demonstrate the effectiveness of kDN on both vectorial data and affinity graphs with comparison to kNN and b-matching method (Jebara and Shchogolev 2006). (4) Based on kDN, we propose novel algorithms for point set matching and data clustering tasks, and demonstrate the state-of-the-art experimental results.

The rest of the paper is organized as follows. We introduce the related work in Sect. 2, and present our formulation of kDN in Sect. 3. In Sect. 4, we analyze the properties of our formulation and propose an efficient iterative algorithm to compute kDN. In Sect. 5, the experimental results on shape retrieval, semi-supervised learning, point set matching and data clustering are demonstrated. They clearly show the effectiveness of our proposed kDN. Finally, we draw some conclusive remarks in Sect. 6.

2 Related Work

There are three categories of algorithms related to our work. First, from the viewpoint of nearest neighbors, kNN, b-matching (Jebara and Shchogolev 2006) and our proposed kDN all compute neighbors, although the objective functions to optimize are different. Second, from the viewpoint of clustering, our algorithm is similar to the affinity based clustering algorithms, such as affinity propagation (Frey and Dueck 2007) and spectral clustering (Ng et al. 2001; Dhillon et al. 2004; Shi and Malik 2000). All these algorithms try to find coherent clusters based on affinity graphs. Finally, as will be revealed in Sect. 3, our algorithm is essentially a dense subgraph or dense submatrix detection method, thus closely related to methods for detecting dense subgraphs or dense submatrices.

Because of its importance and simplicity, kNN is widely used in many fields. For example, in classification task (Goldstein 1972; Han et al. 2001; Denoeux 2008; Cover and Hart 1967; Li et al. 2001), the k-nearest neighbor classification rule is still a general baseline. In computer graphics, kNN is a necessary building block to recover local geometry from point cloud (Fleishman et al. 2005). And in graph related fields, kNN is the most popular method to construct sparse graphs (Jebara et al. 2009).

There are many articles on fast computation of kNN (Arya et al. 1998; Indyk and Motwani 1998; Connor 2010; Chan 1998; Yu et al. 2001; Kolahdouzan and Shahabi 2004). Among them, two representative ones are Indyk's Approximate Nearest Neighbors (ANN) (Indyk and Motwani 1998) and Connor's work (Connor 2010). ANN (Indyk and Motwani 1998) assumes that the distances are measured by Minkowski metric, which includes the well known Euclidean distance, Manhattan distance, and max distance, and speeds up the computation of approximate nearest neighbors by kd-tree technique (Bentley 1975). In (Connor 2010), Connor utilizes Z-order of points, a space filling curve technique, to speed up the computation of nearest neighbors. All these methods are only suitable for vectorial data, and cannot deal with affinity graphs.

kNN is not symmetric, which is not desirable in many situations (Jebara and Shchogolev 2006). To overcome this shortcoming, Jebara and Shchogolev propose the b-matching (Jebara and Shchogolev 2006) method to compute symmetric nearest neighbors. The same as kNN, b-matching does not consider the interactions among neighbors.

Since k-dense neighborhood form a local coherent cluster, our algorithm is essentially an affinity based cluster detection algorithm. There are many clustering algorithms based on pairwise affinities, such as affinity propagation (Frey and Dueck 2007), kernel k-means (Ng et al. 2001; Dhillon et al. 2004) and normalized cut (Shi and Malik

2000). Affinity propagation (Frey and Dueck 2007) automatically selects exemplars and assigns other points to exemplars. Kernel k-means (Ng et al. 2001; Dhillon et al. 2004) generalizes the well-known k-means method into kernel space, while the normalized cut method (Shi and Malik 2000) utilizes the second smallest eigenvector of the Laplacian matrix to bisect the data. All these methods are partition-based, and insist on assigning each data point into a cluster. In many real applications, especially when the number of outliers is large compared to the size of the data set, insisting on partition every point into a cluster usually leads to bad results. In contrast, our k-dense neighborhood based clustering algorithm can automatically detect outliers and points near the boundaries, leaving these points un-grouped and clustering other "easy" points with very high precision.

Since each local coherent cluster corresponds to a dense subgraph on the affinity graph, or a dense submatrix in the adjacency matrix of the affinity graph, our proposed algorithm is in fact an efficient dense subgraph and dense submatrix detection algorithm. Dense subgraphs (submatrices) are widely used in many fields. For example, the community structure (Clauset et al. 2008), which appears in social networks, is in fact a dense subgraph of the social network. The maximal clique (Ouyang et al. 1997), which plays a fundamental role in many graph related problems, is also a kind of dense subgraph. In machine learning field, the one-class clustering/classification problem (Gupta and Ghosh 2005; Crammer et al. 2008), which finds a small and coherent subset of points within a given data set, arises naturally in a wide range of applications. Such coherent subsets of points also form dense subgraphs.

In essence, detecting dense subgraphs is a combinatorial optimization problem, thus NP-hard (Asahiro et al. 2002). Obviously, directly enumerating all dense subgraphs is prohibitive. Many approximation algorithms have been proposed for computing dense subgraphs (Pavan and Pelillo 2007; Ouyang et al. 1997; Zaki et al. 1997; Hu et al. 2005; Gibson et al. 2005). Among them the most famous one was proposed by Motzkin and Straus (1965), and they have proven that solving maximal clique problem is equivalent to finding the maxima of a quadratic function on the simplex. The weighted counterpart of maximal clique, dominant set, also corresponds to a dense subgraph, and has been used for pairwise clustering (Pavan and Pelillo 2007). These algorithms are computationally expensive, both in time and space. At the same time, they usually miss important dense subgraphs. Our algorithm optimizes the same objective function as the method in (Motzkin and Straus 1965), but imposes one more constraint which controls the size of dense subgraphs. Our algorithm can find all significant dense subgraphs efficiently with high probability.

3 Problem Formulation

As introduced in Sect. 1, for a dataset $O = \{o_1, o_2, \ldots, o_n\}$ with the affinity function $f : O \times O \to R$, we can construct an affinity graph G with every vertex representing an object and every edge representing the affinity between the corresponding objects of its two nodes. Denote the adjacency matrix of the affinity graph G as A, that is, $A(i, j) = f(o_i, o_j), 1 \le i \le n, 1 \le j \le n$. When $i \ne j$, $f(o_i, o_j)$ represents the affinity between object o_i and object o_j. When $i = j$, $f(o_i, o_j)$ may have different meanings in different applications. In some applications, $f(o_i, o_i)$ simply represents the similarity between an object and itself. Since an object is most similar with itself, in such cases, $f(o_i, o_i)$ is usually equal to a constant which represents the highest affinity value. In other applications, $f(o_i, o_i)$ may be assigned a value which represents certain quantity of the object o_i. For example, in discrete labeling, o_i represents an assignment, and $f(o_i, o_i)$ then represents the score of this assignment.

According to the definition of kDN, to compute the kDN of an object o, we need to find the solution to the following combinatorial optimization problem:

$$k\mathrm{DN}(o) = \arg\max_{T} S(\{o\} \cup T),$$
$$T \subset O,\ o \notin T \text{ and } |T| = k. \tag{2}$$

For a set U, suppose $y(U)$ is its indicator vector, i.e., a column vector such that $y_i(U) = \begin{cases} 1, & o_i \in U \\ 0, & o_i \notin U \end{cases}$, $i = 1, \ldots, n$, then the total affinity of the set U can be represented in the following form:

$$S(U) = \sum_{o_i \in U} \sum_{o_j \in U} f(o_i, o_j) = y(U)^T A y(U), \tag{3}$$

and (2) can be represented as:

$$\begin{cases} \max\limits_{y} \{S(y) = y^T A y\}, \\ \text{s.t.} \quad \sum\limits_{i} y_i = k + 1, \quad y_i \in \{0, 1\} \text{ and } y_o = 1. \end{cases} \tag{4}$$

The constraint $\sum_i y_i = k + 1, y_i \in \{0, 1\}$ means the solution y indicates $k + 1$ objects, while the constraint $y_o = 1$ means that the solution must contain the object o.

Note that we can decompose y into two parts, that is, $y = y_o + y_u$, where y_o is the indicator vector of the object o and y_u is the indicator vector of the other objects, then we have:

$$\begin{aligned} S(y) &= (y_o + y_u)^T A (y_o + y_u) \\ &= y_o^T A y_o + 2 y_o^T A y_u + y_u^T A y_u. \end{aligned} \tag{5}$$

The first term $y_o^T A y_o$ is a constant, the second term $2y_o^T A y_u$ is the objective function that standard kNN maximizes, while the third term $y_u^T A y_u$ represents interactions among the k selected objects. Consequently, (5) nicely illustrates the key difference between kNN and the proposed kDN.

We first divide $S(y)$ by $(k+1)^2$, which is a constant when k is fixed:

$$\frac{S(y)}{(k+1)^2} = \frac{y^T A y}{(k+1)^2} = \left(\frac{y}{k+1}\right)^T A \left(\frac{y}{k+1}\right) = x^T A x, \quad (6)$$

where $x = \frac{y}{k+1}$, then we transform (4) into:

$$\begin{cases} \max\limits_x \{g(x) = x^T A x\}, \\ \text{s.t.} \quad \sum\limits_i x_i = 1, \quad x_i \in \left\{0, \frac{1}{k+1}\right\} \text{ and } x_o = \frac{1}{k+1}. \end{cases} \quad (7)$$

Since there are $(k+1)^2$ components in $S(y)$, $g(x)$ represents the average affinity within the vertex set corresponding to the positive elements in x.

The complexity of (7) mainly comes from the constraint $x_i \in \{0, \frac{1}{k+1}\}$, which requires that each component of x can only have two possible values, either 0 or $\frac{1}{k+1}$. In this work, we relax this constraint to $x_i \in [0, \frac{1}{k+1}]$. To simply the notion, we let $\delta = \frac{1}{k+1}$ and denote the solution space $\{x | \sum_{i=1}^n x_i = 1, x_i \in [0, \delta], x_o = \delta\}$ as V_o^δ, then we get a constrained quadratic programming problem:

$$\begin{cases} \max\limits_x \{g(x) = x^T A x\}, \\ \text{s.t.} \quad x \in V_o^\delta. \end{cases} \quad (8)$$

Note that V_o^δ is part of the simplex in R^n.

Although (8) is an approximation of (7), its solution is usually the correct kDN(o), and we will demonstrate this point in the experimental part. At the same time, formulation (8) has many good properties, and we highlight some important ones in the following:

- Adding a constant to all elements of A will not affect the solution. Suppose C is an $n \times n$ matrix with all elements equal to c, then $\hat{g}(x) = x^T(A + C)x = g(x) + x^T C x = g(x) + c$, thus, $\hat{g}(x)$ and $g(x)$ have the same maximizers. In the same way, multiplying A by a positive value c will not affect the solution, since $\hat{g}(x) = x^T(cA)x = cx^T Ax = cg(x)$. Owing to these properties, we simply assume $A(i, j) \in [0, 1], \forall i, j \in \{1, 2, \ldots, n\}$ in this work.
- The adoption of ℓ_1-norm in (8), that is, $\sum_{i=1}^n x_i = 1$, makes the solution sparse, which means that the neighbors are automatically selected, and other objects are discarded. x_i has an intuitive probabilistic meaning, namely, it represents the probability of $o_i \in \{o\} \cup k\text{DN}(o)$. For $o_i \neq o$, the larger x_i is, the more probably $o_i \in k\text{DN}(o)$.
- Since $\delta = \frac{1}{k+1}$ and $x_i \leq \delta$, the number of non-zero components of the solution x^* is at least $k+1$. In fact, in most cases, the number is exactly $k+1$. Even when the number of non-zeros components is larger than $k+1$, we can easily select the k neighbors according to their probabilities.

The formulation (8) is similar to the standard quadratic programming problem (SQP) (Bomze and De Klerk 2002), and both of them maximize a quadratic function on the simplex. The difference lies in that (8) has an extra constraint $x_i \leq \delta$, which can prevent x from being dominated by a small number of components. Although there are many numeric methods to solve the constrained quadratic programming problem, by analyzing the properties of (8), a more efficient algorithm can be found, which can also scale up well.

4 Solution

The objective function $g(x) = x^T A x, x \in V_o^\delta$ may have many local maxima, and the solution of (8) is its global maximizer. All local maximizers of the function $g(x) = x^T A x, x \in V_o^\delta$ constitute a set, denoted by Υ_o^δ. In this section, we first analyze the common properties of the elements in Υ_o^δ, and then introduce our algorithm to compute the solution of (8).

4.1 Properties of the Solution

Since $x_o = \delta$, there is $n-1$ variables in (8), that is, $\{x_i | o_i \neq o\}$. For each variable $x_i, o_i \neq o$, we add two Lagrangian multipliers, μ_i and β_i, and for the equation $\sum_i x_i = 1$, we add a Lagrangian multiplier λ, then we can obtain the Lagrangian function of (8):

$$L(x, \lambda, \mu, \beta) = \frac{1}{2} g(x) - \lambda \left(\sum_{i=1}^n x_i - 1 \right) + \sum_{i, o_i \neq o} \mu_i x_i + \sum_{i, o_i \neq o} \beta_i (\delta - x_i). \quad (9)$$

Any local maximizer x^* must satisfy the Karush-Kuhn-Tucker (KKT) conditions (Kuhn and Tucker 1951), i.e., the first-order necessary conditions for local optimality. That is,

$$\begin{cases} (Ax^*)_i - \lambda + \mu_i - \beta_i = 0, \quad o_i \neq o; \\ \sum\limits_{i, o_i \neq o} x_i^* \mu_i = 0; \\ \sum\limits_{i, o_i \neq o} (\delta - x_i^*) \beta_i = 0. \end{cases} \quad (10)$$

Since x_i^*, μ_i and β_i are all nonnegative, $\sum_{i, o_i \neq o} x_i^* \mu_i = 0$ is equivalent to saying that when $o_i \neq o$, if $x_i^* > 0$, then

$\mu_i = 0$, and $\sum_{i,o_i \neq o}(\delta - x_i^*)\beta_i = 0$ is equivalent to saying that when $o_i \neq o$, if $x_i^* < \delta$, then $\beta_i = 0$. Hence, the KKT conditions can be rewritten as:

$$(Ax^*)_i \begin{cases} \leq \lambda, & x_i^* = 0 \text{ and } o_i \neq o; \\ = \lambda, & 0 < x_i^* < \delta \text{ and } o_i \neq o; \\ \geq \lambda, & x_i^* = \delta \text{ and } o_i \neq o. \end{cases} \tag{11}$$

Note that $(Ax)_i = \sum_j A(i,j)x_j = e_i^T Ax$, where e_i is a column vector with only the ith component equal to 1 and zero otherwise. Since it reflects the total affinity of o_i to other objects represented by x, we call it *reward* at object o_i, and denoted it by $r_i(x)$. Since $g'(x) = 2Ax$, the reward at node i is in fact half of the partial derivative $\frac{\partial g}{\partial x_i}(x)$.

According to x and δ, the dataset O can be divided into four disjoint subsets, $\{o\}$, $S_1(x,\delta) = \{o_i | x_i = 0\}$, $S_2(x,\delta) = \{o_i | x_i \in (0,\delta)\}$, and $S_3(x,\delta) = \{o_i | x_i = \delta, o_i \neq o\}$. The index set of all non-zero components of x constitutes its *support*, which is denoted by $\zeta(x)$, and $|\zeta(x)|$ denotes the size of $\zeta(x)$.

A point $x \in V_o^\delta$ satisfying (11) is called a *KKT point*, and all KKT points form a set, denoted by Γ_o^δ. Since KKT condition is a necessary condition, then $\Upsilon_o^\delta \subseteq \Gamma_o^\delta$, and (11) in fact characterizes the common property of the local maximizers of the function $g(x) = x^T Ax, x \in V_o^\delta$, which is summarized in the following theorem.

Theorem 1 *If x^* is the local maximizer of the function $g(x) = x^T Ax, x \in V_o^\delta$, then there exists a constant λ such that* (1) *the rewards at all objects belonging to $S_1(x^*,\delta)$ are not larger than λ*; (2) *the reward at all objects belonging to $S_2(x^*,\delta)$ are equal to λ*; *and* (3) *the rewards at all objects belonging to $S_3(x^*,\delta)$ are not smaller than λ.*

4.2 Compute Maximizers by Pairwise Updating

Our aim is to obtain the global maximizer of the function $g(x) = x^T Ax, x \in V_o^\delta$. Although Theorem 1 states the common properties of all local maximizers (including the global maximizer), it does not point out how to compute these local maximizers.

For any $x \in V_o^\delta$, we propose to update it in the following pairwise way:

$$x_k^{\text{new}} = \begin{cases} x_k, & k \neq i, k \neq j; \\ x_k + \alpha, & k = i; \\ x_k - \alpha, & k = j. \end{cases} \tag{12}$$

That is, each time we only update a pair of components $(x_i, x_j), i \neq j$, as the change in value of function $g(x)$ we obtain:

$$\begin{aligned} \Delta g(x) &= g(x^{\text{new}}) - g(x) \\ &= (A_{ii} + A_{jj} - 2A_{ij})\alpha^2 \\ &\quad + 2\left(\sum_k A_{ik}x_k - \sum_k A_{jk}x_k\right)\alpha \\ &= (A_{ii} + A_{jj} - 2A_{ij})\alpha^2 + 2(e_i Ax - e_j Ax)\alpha \\ &= (A_{ii} + A_{jj} - 2A_{ij})\alpha^2 + 2(r_i(x) - r_j(x))\alpha. \end{aligned} \tag{13}$$

To maximize $\Delta g(x)$, according to (13) and the constraints over x, α can be computed in the following way:

$$\alpha = \begin{cases} \min(x_j, \delta - x_i), \\ \quad r_i(x) > r_j(x), A_{ii} + A_{jj} - 2A_{ij} \geq 0; \\ \min(x_j, \delta - x_i, \frac{r_j(x) - r_i(x)}{A_{ii} + A_{jj} - 2A_{ij}}), \\ \quad r_i(x) > r_j(x), A_{ii} + A_{jj} - 2A_{ij} < 0; \\ \min(x_j, \delta - x_i), \\ \quad r_i(x) = r_j(x), A_{ii} + A_{jj} - 2A_{ij} > 0. \end{cases} \tag{14}$$

Note that we just consider the situation where $r_i(x) \geq r_j(x)$, when $r_i(x) < r_j(x)$, we can exchange i and j.

From (13) and (14), we obtain that: (1) if $r_i(x) > r_j(x)$, then there exists $\alpha > 0$ such that $g(x)$ can be increased by updating x according to (12); (2) when $r_i(x) = r_j(x)$, if $A_{ii} + A_{jj} - 2A_{ij} > 0$, $g(x)$ can also be increased by increasing either x_i or x_j, and decreasing the other one; (3) when $r_i(x) = r_j(x)$ and $A_{ii} + A_{jj} - 2A_{ij} = 0$, according to (13), we can change x without affecting the value of $g(x)$, which means that x may be on a ridge or a plateau.

As stated before, since $x_i \leq \delta = \frac{1}{k+1}$, the number of non-zero components of the elements in Υ_o^δ is at least $k+1$. However, in some cases, we can get very firm result, which is stated in the following theorem:

Theorem 2 *If $A_{ii} + A_{jj} - 2A_{ij} > 0$ for all $i, j \in \{1, \ldots, n\}$ and $\delta = \frac{1}{k+1}$, then for any $x \in \Upsilon_o^\delta$ we have $|\zeta(x)| = k+1$.*

Proof According to Theorem 1 and (13), we know that if $x \in \Upsilon_o^\delta$ and $A_{ii} + A_{jj} - 2A_{ij} > 0$, then o_i and o_j cannot coexist in $S_2(x,\delta)$. For any $x \in \Upsilon_o^\delta$, if $|\zeta(x)| > k+1$, then $S_2(x,\delta)$ contains at least 2 objects, namely o_i and o_j; however, since $A_{ii} + A_{jj} - 2A_{ij} > 0$, this is impossible. Thus, our assumption is not correct and the theorem has been proved. □

In many applications, self-links have no meanings and we can freely assign values to them. According to the tasks, we usually adopt two strategies: (1) if the aim is to get accurate k neighbors, according to Theorem 2, we can assign the largest possible value of affinity (1) to all self-links; (2) if the aim is to automatically detect local clusters, we usually assign the smallest possible value of affinity (0) to all self-links to encourage automatic cluster detection. In such cases, the parameter δ is utilized to control the minimal size of the local clusters.

Note that the formulation (8) is closely related to the dominant set method proposed by Pavan and Pelillo (2007).

Algorithm 1 Compute the local maximizer x^* from a starting point $x \in V_Q^\delta$

Input: The weighted adjacency matrix A of graph G, the starting point x and δ.
repeat
  Select $o_i \in S_1(x,\delta) \cup S_2(x,\delta)$ with the largest reward $r_i(x)$;
  Select $o_j \in S_2(x,\delta) \cup S_3(x,\delta)$ with the smallest reward $r_j(x)$;
  if $r_i(x) > r_j(x)$ **then**
    Compute α according to (14), update x and rewards;
  else if $r_i(x) = r_j(x)$ **then**
    Find a pair (o_t,o_s) satisfying $A_{tt} + A_{ss} - 2A_{ts} > 0$ and $r_t(x) = r_s(x)$, where $o_t \in S_1(x,\delta) \cup S_2(x,\delta)$ and $o_s \in S_2(x,\delta) \cup S_3(x,\delta)$, respectively. If such a pair exists, then compute α according to (14), update x and rewards; Otherwise, x is already a local maximizer.
  end if
until x is the local maximizer
Output: The local maximizer x^*.

Both of these two methods detect dense subgraphs by quadratic programming. However, the difference in constraint results in different algorithms. In our method, kDN(o) always contains the object o; while in the dominant set method, the detected clusters cannot guarantee to contain certain objects. In our method, the constraint over x_i is $x_i \in [0,\delta]$; while in the dominant set method, the constraint over x_i is $x_i \in [0,1]$. Since $\delta \leq 1$, our method in fact limits the upper-bound of x_i, which not only prevents x from being dominated by a small number of components, but also results in the accurate number of neighbors according to Theorem 2. In other words, the method in Pavan and Pelillo (2007) does not guarantee that the obtained dense subgraph contains k elements.

By iteratively utilizing the update equation (12) and computing α according to (14), we can efficiently compute a local maximizer x^* of $g(x) = x^T Ax, x \in V_o^\delta$ from any starting point $x \in V_o^\delta$, and the algorithm is summarized in Algorithm 1.

In each iteration, to maximize the increase of $g(x)$, Algorithm 1 adopts a heuristic strategy to select a pair (o_i, o_j): selecting the pair (o_i, o_j) such that o_i is the object with the largest reward among the objects whose probabilities can increase (objects in the set $S_1(x,\delta) \cup S_2(x,\delta)$), and o_j is the object with the smallest reward among the objects whose probabilities can decrease (objects in the set $S_2(x,\delta) \cup S_3(x,\delta)$). According to (13), this pair has the potential to maximize the increase of $g(x)$. Intuitively speaking, we select the "best" vertex and the "worst" vertex and then update their corresponding components of x. When the rewards of o_i and o_j is equal, we get a KKT point, and then we check whether a local maximizer has been reached or not according to (14). If $g(x)$ cannot be increased, then x is already a local maximizer, otherwise it is not.

In each iteration, Algorithm 1 only deals with two components of x, and only the rewards of the objects with large affinities to these two objects need to be updated, thus it is very efficient, both in time and memory. Intuitively speaking, Algorithm 1 prefers objects with large rewards and tries to increase the probabilities of objects with large rewards, and at the same time, decreases the probabilities of objects with small rewards.

4.3 Construct Initializations Using Neighborhood Information

Since Algorithm 1 can find a local maximum of the function $g(x) = x^T Ax, x \in V_o^\delta$ from any initialization $x \in V_o^\delta$, to find the global maximum, we need an initialization lying in the attractive basin of the global maximum. In theory, it is hard to generate initializations with such guarantee. However, in many real applications, the global maximum is usually significantly larger than other maxima and its attractive basin usually covers a very large region. In such case, we have a very high probability to generate initializations lying in the attractive basin of the global maximum, especially when we explore the local data distribution around the object o.

Although no principled criterion, there is a general heuristic rule to construct the initialization lying in the attractive basin of the global maximum: the initialization should contain as many real nearest neighbors as possible. This is because the affinities between real nearest neighbors are usually large. If many real nearest neighbors are contained in the initialization, they will contribute to each other on the rewards and their rewards are usually large. At the same time, other objects in the initialization usually cannot form coherent clusters, thus, their rewards are usually small. As Algorithm 1 iterates, the probabilities of real nearest neighbors increase and the probabilities of other objects decrease, and finally coverage to the true kDN of o.

According to this heuristic rule, there are two natural initialization schemes. One is based on kNN(o) and the other is based on ϵNN(o). The initialization based on kNN(o) is a vector with $x_j = \delta$ if $o_j \in \{o\} \cup k\text{NN}(o)$ and $x_j = 0$ otherwise, where $\delta = \frac{1}{k+1}$. The initialization based on ϵNN(o) is a vector with $x_o = \delta$, $x_j = \frac{1-\delta}{|\epsilon\text{NN}(o)|}$ if $o_j \in \epsilon\text{NN}(o)$ and $x_j = 0$ otherwise. Note that ϵ must be selected to ensure that $|\epsilon\text{NN}(o)| \geq k$. The initialization constructed in this way usually lies in the attractive basin of the global maximum of $g(x) = x^T Ax, x \in V_o^\delta$, since usually only a small portion of pairwise affinities are not reliable and most of objects in kNN(o) and ϵNN(o) are true nearest neighbors of o. In the

Algorithm 2 Refine the candidate kDNs of all objects in the dataset O

1: **Input:** O and the candidate kDNs of all objects in O.
2: Sort the objects in dataset O in descending order according to the total affinities of their candidate kDNs.
3: Set $M = \emptyset$;
4: **repeat**
5: Select an object $o \notin M$ whose candidate kDN has the largest total affinity, and add it into the set M;
6: **for** every object o_i belonging to the candidate kDN of o **do**
7: If $o_i \notin M$, then set the candidate kDN of o_i to the candidate kDN of o and add o_i into the set M;
8: **end for**
9: **until** $|M| = |O|$
10: **Output:** The refined candidate kDNs of all objects in O.

experiments, we select the initialization scheme according to the properties of the applications.

For each object o, we can construct an initialization and then obtain a local maximizer x^* by Algorithm 1. As stated before, x^* in fact represents a local coherent cluster, which contains at least $k+1$ objects, together with o. Imagining that there exist real clusters in the dataset O, then for each object in a real cluster, the obtained local maximizer x^* is very likely to represent part of a real cluster. At the same time, each obtained cluster has at least $k+1$ objects, which means that we can control the lower bound of the size of obtained cluster. This property is very useful, since in many applications, clusters of very small sizes are usually not desirable (Shi and Malik 2000). Thus, Algorithm 1 in fact offers a flexible tool to detect clusters underlying the data.

Based on the local maximizer x^*, we can construct a candidate kDN of o. Since $x_i^*, o_i \neq o$ represents the probability of $o_i \in k\text{DN}(o)$, we select the k most probable objects to construct the candidate kDN. In most of cases, the candidate kDN is $k\text{DN}(o)$; even if it is not $k\text{DN}(o)$, it usually has large intersection with $k\text{DN}(o)$. At the same time, the kDNs of different objects are dependent. For two objects o_i and o_j, if o_i is an element of $k\text{DN}(o_j)$, then the total affinity of $k\text{DN}(o_i)$ should be not smaller than the total affinity of $k\text{DN}(o_j)$, otherwise the candidate kDN of o_i is not the $k\text{DN}(o_i)$ and $k\text{DN}(o_j)$ is a better candidate kDN of o_i. Since we usually need to find the kDNs of many objects and for some objects, their candidate kDNs are not their true kDNs, we can use this dependency information to get the true kDNs of these objects.

Suppose we need to find the kDNs of all objects in the dataset $O = \{o_1, \ldots, o_n\}$. Based on the candidate kDNs of all objects and utilizing the dependency of kDNs of different objects, we can refine the candidate kDNs by Algorithm 2.

For some object o_i, maybe the initialization does not lie in the attractive basin of the global maximum of the function $g(x) = x^T Ax, x \in V_{o_i}^{\delta}$, thus, the obtained local maximizer is not its global maximizer and the candidate kDN is not its kDN. However, for another object $o_j \in k\text{DN}(o_i)$, $k\text{DN}(o_j)$ may be the same as $k\text{DN}(o_i)$. If we find the global maximizer of the function $g(x) = x^T Ax, x \in V_{o_j}^{\delta}$, according to Algorithm 2, we may also get the kDN of o_i. Thus, the dependency between kDNs of different objects may increase the chance of finding the true kDN, and Algorithm 2 not only increases the accuracy of kDNs, but also makes the kDNs of different objects consistent.

4.4 Implemental Details and Time Complexity

In the implementation of Algorithm 1, to save the memory, the adjacency matrix A is usually stored in sparse form, and the entries of very small values are discarded. Suppose the number of stored entries is m, then at the initialization, the time complexity of computing rewards for all variables is $O(m)$. We use the heap data structure to store both sets $S_1(x,\delta) \cup S_2(x,\delta)$ and $S_2(x,\delta) \cup S_3(x,\delta)$. Since there are at most n objects in each heap, the time complexity of constructing two heaps is $O(n)$. In each iteration, the time complexity of finding a pair of objects is $O(\log(n))$. At the same time, we need to update the rewards of affected objects, which are neighbors of the selected two objects, and also adjust the heap structure. The time complexity of adjusting the position of an object in a heap is $O(\log(n))$. Suppose the number of neighbors of each object is h and the number of iterations is t, then the total time complexity of Algorithm 1 is $O(t(h+\log(n)+h\log(n))+m+n)$ and the time complexity of computing kDNs of all objects is $O(nt(h+\log(n)+h\log(n))+nm+n^2)$. The time complexity of Algorithm 2 is $O(n\log(n))$ and the main computation cost is the sort operation.

5 Experimental Results

Both kNN and kDN are neighborhood analysis techniques. Thus, kDN is a natural alternative to kNN in many applications, such as object retrieval (Ling and Jacobs 2007) and classification (Cover and Hart 1967). Because kDN better preserves the neighborhood structure, especially on affinity graphs with unreliable affinities, kDN can get much better results than kNN, and hereafter we demonstrate this point with experiments on shape retrieval and semi-supervised learning. We also compare with the b-matching method (Jebara et al. 2009), which is another neighborhood analysis technique that provides symmetric neighborhoods.

Since $k\text{DN}(o)$ is a locally coherent cluster around object o, as stated in Sect. 4, kDN is also a good cluster analysis tool. Based on Algorithm 1, we propose a novel point set

matching algorithm, and show that it is very robust to noises and outliers. Based on kDN, we also propose a new clustering method, and illustrate its performance on four popular datasets, Iris, Wine, Breast-cancer (Frank and Asuncion 2010) and Australian (Chang and Lin 2001).

5.1 Precision of Candidate kDNs

For an object, the candidate kDN obtained by Algorithms 1 and 2 may be not its kDN. First, kDN is the solution of (7), but we compute it by solving the relaxed problem (8), which is an approximation of (7). Second, the obtained local maximizer of (8) may be not its global maximizer.

To evaluate the precision of the candidate kDNs, we conduct the following experiment: randomly generate a 20×20 symmetric matrix, with each element of the matrix being a uniformly distributed random number in the range $[0, 1]$ to represent similarity between a pair of points, then we compute the candidate kDN of every object and compare it with the true kDN obtained by exhaustive enumeration. Note that such randomly generated matrices do not contain obvious clusters; while in the real applications, there often exists obvious clusters. Thus, computing kDNs on such randomly generated matrices is much harder than computing kDNs on real data, since the average affinities of real clusters are usually significantly larger than the average affinities of other groups of objects.

For an object o, the computed candidate kDN by Algorithms 1 and 2 is assumed to be R. Because of the relaxation in (8) and the obtained local maximizer may be not the global maximizer of (8), R may be not the real kDN of o. How to measure the precision of R? From the viewpoint of nearest neighbors, a natural measure is *correct ratio*, which is defined to be $\frac{|R \cap k\mathrm{DN}(o)|}{k}$, namely, the percentage of correct k-nearest neighbors in R. From the viewpoint of maximum, another natural measure is *difference ratio*, which measures how well the local maxima approximates the global maximum, and is defined to be $\frac{S(k\mathrm{DN}(o)) - S(R)}{S(k\mathrm{DN}(o))}$.

Figure 2 illustrates the average correct ratio and average difference ratio as k varies. In this experiment, we utilize kNN to construct the initializations. Of course, we can also use εNN to construct the initializations. For each value of k, we repeat the experiments 30 times to obtain the mean performance. Figure 2 reveals an important fact: as k increases, the performance generally improves. This is because the total affinity is the sum of $(k+1)^2$ elements, and such ensemble effect is generally enhanced as the number of elements increases. At the same time, at small k, the correct ratio seems to be not high, however, the difference ratio is very small, which means that the errors resulting from the small differences between global maximum and local maxima. In real applications, the differences between global maximum and local maxima are usually very large, in such situations, even at small k, the correct ratios are usually very high.

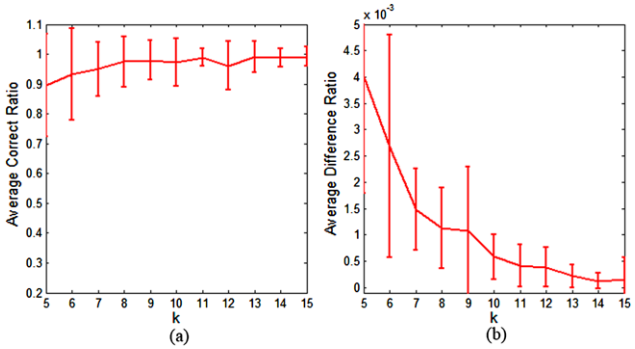

Fig. 2 Average performance of our algorithm, evaluated based on ground truths obtained by exhaustive enumeration

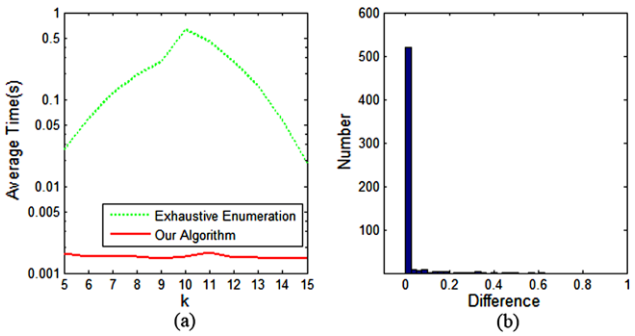

Fig. 3 (**a**) Time complexity of both our method and exhaustively enumeration method on randomly generated 20×20 symmetric matrix. (**b**) Histogram of differences in total affinity between candidate kDNs and true kDNs

In Fig. 3(a), the red solid curve shows the time complexity of our algorithm, while the green dotted curve shows the time complexity of the exhaustive enumeration method. The exhaustive enumeration is very time-consuming, and when n is large, exhaustive enumeration is time-prohibitive. However, our method is very efficient and has high probability to find the correct kDNs.

In another experiment, we set $n = 20$ and $k = 10$, and repeat the experiment 30 times. Thus, the total number of candidate kDNs is $20 \times 30 = 600$. For each candidate kDN obtained by our algorithm, we calculate its difference in total affinity with the true kDNs obtained by exhaustive enumeration. The histogram of the difference is shown in Fig. 3(b). As expected, most differences are zeros, which means that the candidate kDNs are true kDNs. Even for those candidate kDNs which are not true kDNs, the differences are very small.

In Fig. 4(a), we do the experiments on randomly generated 30×30 symmetric matrix. Our proposed method is still very efficient; however, the exhaustively enumeration method becomes very time-consuming. For each value of k, we only do the experiments one times, because of the high

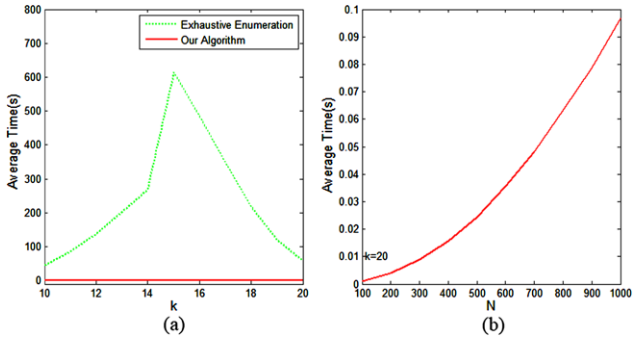

Fig. 4 (**a**) Time complexity of both our method and exhaustively enumeration method on randomly generated 30×30 symmetric matrix. (**b**) Time complexity of our method on randomly generated symmetric matrix of size $N \times N$, with $k = 20$

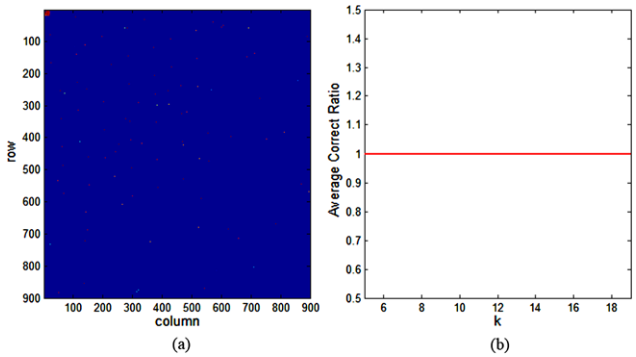

Fig. 5 (Color online) (**a**) Graph illustration of the affinity matrix, with *red* represents large affinities and *blue* represents small affinities. (**b**) The curve of average correct ratio as k varies from 5 to 19

time complexity of exhaustively enumeration method. Note that as the size of matrix increases (from 20 in Fig. 3(a) to 30 in Fig. 4(a)), the time complexity of the exhaustively enumeration method increases very quickly. In Fig. 4(b), we fix $k = 20$, increase the size of matrix (N) from 100 to 1000, and repeat the experiments 30 times for each value of N. Obviously, our method is still very efficient for very large matrix.

We also do the experiments on a 900×900 matrix obtained from the point set matching experiment described in the Sect. 5.3.1. The matrix is illustrated in Fig. 5(a). The first 20 objects form a real cluster, thus, there is a very dense 20×20 submatrix in the upper left corner of Fig. 5(a). In this experiment, we only compute the kDNs of the first 20 objects, since the kDNs of other objects are meaningless. We vary k from 5 to 19 and demonstrate the average correct ratio in Fig. 5(b). The obtained candidate kDNs are always true kDNs, since there exists obvious cluster structure in the data, and the average affinity of the first 20 objects is significantly larger than the average affinities of any other groups of 20 objects.

5.2 kDN vs. kNN and b-matching

To clearly illustrate the differences between kNN, b-matching method (Jebara et al. 2009) and kDN, we first perform experiments on two toy datasets, namely, affinity graphs with unreliable affinities and the two-moons dataset. On the affinity graphs, we will demonstrate that kDN can suppress the influence of some unreliable affinities and obtain more reliable neighborhoods than kNN and b-matching method; while on the two-moons dataset, we will demonstrate that kDN can preserve the structure underlying the dataset. Then we compare the three methods on two tasks, namely, shape retrieval and semi-supervised learning. In both tasks, all other factors are fixed and remain the same, and we only replace kNN by kDN and b-matching method to compare the performance. The initializations of Algorithm 1 is constructed based on kNN. Since we need to find exactly k neighbors, in these two tasks, we set the weights of self-links as 1 and keep all the pairwise affinities smaller than 1. According to Theorem 2, in such situation, the number of obtained neighbors is exactly k. For the b-matching method, we use the c code provided by the author, which can be downloaded from the following website: http://www.cs.columbia.edu/~jebara/code/loopy/.

5.2.1 *Results on Toy Datasets*

To clearly illustrate that kDN can obtain more reliable neighborhoods on affinity graphs, we do the experiment on a toy problem. We first generate an affinity graph with 3 clusters, with each cluster contains 30 vertices, and the affinities between vertices in the same cluster are 1s, otherwise 0s. Then we randomly choose edges and modify their weights from 0 to 1 or from 1 to 0. In this way, we obtain an affinity graph with unreliable affinities. We randomly choose edges for $\frac{90(90-1)}{2}\alpha$ times. Thus, the number of unreliable affinities is roughly controlled by α (an edge may be chosen multiple times). On the generated affinity graph, for each vertex, we calculate its 29 neighbors by kNN, b-matching method and kDN, respectively, and the average precision of the neighbors is used for comparison.

In our experiment, $\alpha = \{0.01, 0.02, 0.04, 0.08, 0.16, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$, and for each value of α, we repeat the experiment 30 times to obtain the average performance. The result is shown in Fig. 6. Clearly, kDN significantly outperforms kNN and the b-matching method, this is because kDN utilizes the ensemble effect of all pairwise affinities within a cluster. The performance curves of kNN and b-matching method drop very quickly as α increases; however, the performance curve of the proposed kDN drops very slowly when α increases from 0.01 to 0.4. Note that only when α gets close to 0.9, the ensemble effect makes our method perform worse than kNN and b-matching. In such situation,

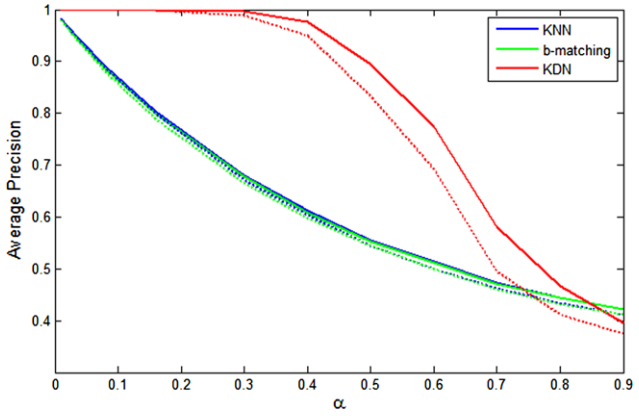

Fig. 6 (Color online) Precisions of neighbors on affinity graphs with unreliable affinities. The *solid lines* show the mean performance, and the *dotted lines* show one *std* below the mean. Our method is shown *in red*, while kNN and b-matching method are illustrated in *blue* and *green*, respectively

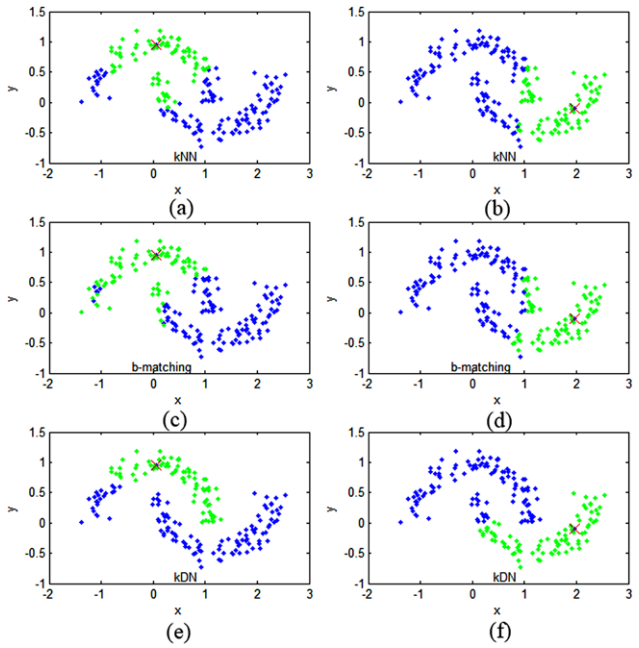

Fig. 7 (Color online) Illustration of the differences between kNN, b-matching and kDN on the two-moons dataset, $k = 80$. *Red cross points* are the reference points, and *green dot points* are their 80 nearest neighbors

most of affinities are unreliable, which wipes out the ground truth cluster information.

To intuitively illustrate the differences between kNN, b-matching method and kDN on vectorial data, in Fig. 7, we plot the results of kNN, b-matching method and kDN of two points in the two-moons dataset. Red cross points are the reference points, and green dot points are 80 nearest neighbors of the reference points, obtained by kNN in (a) and (b), by b-matching method in (c) and (d), and by kDN in (e) and (f), respectively. Obviously, kDN is more

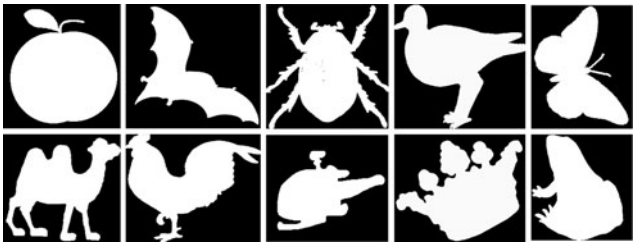

Fig. 8 Ten examplar shapes from MPEG-7 shape dataset

coherent and thus does not contain outliers, while kNN and b-matching contain a lot of outliers.

5.2.2 *Shape Retrieval*

The shape retrieval experiment is based on the affinity data from shape matching. The database is the well known MPEG-7 shape database (Ling and Jacobs 2007). There are 70 categories and each category contains 20 shapes. Figure 8 demonstrates 10 examplar shapes in this dataset. Obviously, it is hard to naturally describe these shapes by vectorial data, thus existing state-of-the-art shape matching methods (Ling and Jacobs 2007) are based on direct distance computations. For each pair of shapes, we calculate their matching score (affinity value) using IDSC method (Ling and Jacobs 2007), and thus obtain a 1400×1400 affinity matrix. Such affinity data has no corresponding vectorial representation. At the same time, many affinities (matching scores) are not reliable, since for different pairs of shapes, their matching scores are computed independently, and the matching method may produce inaccurate results on some pairs of shapes.

For each shape o, we can find k most similar shapes based on kNN(o), kDN(o) and b-matching method (Jebara et al. 2009), respectively. The retrieval precision is then measured by the percentage of shapes in the same category as shape o in these k nearest neighbors. In Fig. 9(a), we vary k from 1 to 19, and plot the average retrieval precision of all 1400 shapes as a function of k. For small k, the performance of kDN is nearly the same as kNN; however, when k becomes larger, kDN performs much better, which implies that the ensemble positively influences the performance. In Fig. 9(b), we also vary k from 1 to 19, and plot the ROC curves of all three methods. Note that in each category, there is only 20 shapes, thus, we restrict $k \leq 19$. As Fig. 9(b) demonstrates, the proposed method works much better than the other two methods. For small k, kNN outperforms b-matching; while for large k, b-matching outperforms kNN. The average time of computing nearest neighbors for all 1400 objects is 0.1881 seconds for the kNN method, 1.9753 seconds for our proposed kDN method, and 912.2045 seconds for the b-matching method.

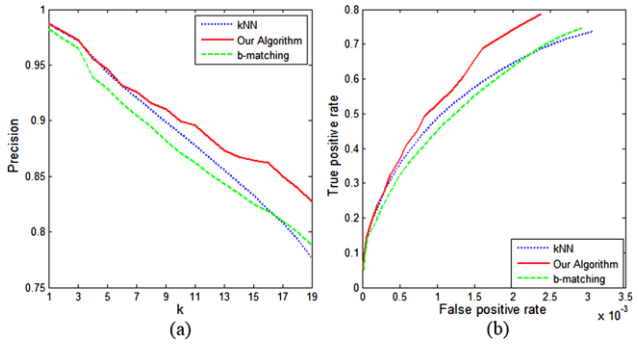

Fig. 9 (Color online) Performance curves of kNN, kDN and b-matching method for shape retrieval over MPEG-7 shape dataset. *Red solid curve* shows the performance of kDN, *blue dotted curve* illustrates the performance of kNN, and *green dashed curve* illustrates the performance of b-matching method

5.2.3 Semi-supervised Learning

Recently, semi-supervised learning (SSL) has proliferated in practical machine learning settings (Chapelle et al. 2006a), since labeled data are often easily complemented with large amount of unlabeled data. Among all SSL methods, graph based ones (Zhu et al. 2003; Zhou et al. 2004) have been widely preferred owing to the advantages of accuracy and efficiency. The graph based SSL methods operate on affinity graphs, and the sparsity of affinity graphs is important to ensure that the SSL methods remain efficient. The most common approach to construct such a sparse graph is k-nearest neighbors. In this case, each vertex merely recovers its k nearest neighbors using the affinity (or distance) function and instantiates k undirected edges between itself and the neighbors. Obviously, the robustness of k nearest neighbors directly influences the performance of SSL methods. Since kDN is usually more robust than kNN, it is expected to lead to better results on the sparse graphs constructed by kDN.

In our experiment, we adopt four methods proposed in (Jebara et al. 2009), GRF-KNN-GK, LGC-KNN-GK, GRF-BM-GK, LGC-BM-GK. GRF-KNN-GK and LGC-KNN-GK construct the sparse graph by kNN, while GRF-BM-GK and LGC-BM-GK construct the sparse graph by b-matching method. GRF-KNN-GK and GRF-BM-GK diffuse the known labels to unlabeled vertices by Gaussian Random Fields (GRF) (Zhu et al. 2003), while LGC-KNN-GK and LGC-BM-GK diffuse the known labels by Local and Global Consistency (LGC) method (Zhou et al. 2004).

We construct the sparse graph by the proposed kDN, thus obtain two new methods, GRF-KDN-GK and LGC-KDN-GK. We conducted the experiments on two benchmark data sets, USPS (Chapelle et al. 2006b) and TEXT (Chapelle et al. 2006b). Moreover, we used the originally suggested data splits for fair comparison, where each data set is associated with 12 different partitions of labeled and unlabeled subsets. Each dataset has two versions, one of which has 10 points with labels and the other has 100 points with labels. The parameters are the same as in Jebara et al. (2009), and $k = 12$. Thus, the only difference between GRF-KNN-GK (or GRF-BM-GK) and GRF-KDN-GK, and between LGC-KNN-GK (or LGC-BM-GK) and LGC-KDN-GK, is the method to find the k nearest neighbors.

Table 1 Experimental results on the benchmark data sets (in terms of % error rate) for semi-supervised learning

Dataset	USPS		TEXT	
# of labels	10	100	10	100
GRF-KNN-GK	19.98	12.36	49.93	48.78
LGC-KNN-GK	15.17	13.23	49.69	48.72
GRF-BM-GK	19.89	9.79	45.45	**26.32**
LGC-BM-GK	13.97	10.76	40.72	27.48
GRF-KDN-GK	17.25	**5.13**	46.25	28.54
LGC-KDN-GK	**11.31**	5.35	**39.96**	29.08

The experimental results are shown in Table 1. For each dataset, the best result is shown in bold. Most of the best results are obtained by kDN, and the improvements are significant, especially compared to kNN. The reason why kDN performs better is probably that it yields a more coherent neighborhood than kNN. In other words, kNN may contain outliers, while kDN utilizes the average affinity to eliminate the outliers. Since b-matching method emphasizes the consistency of nearest neighbors, thus it also performs much better than kNN. For the time complexity, on the task of USPS data set with 10 labels, the time needed is 10.2289 seconds for the kNN method, 63.1439 seconds for our proposed kDN method, and 14133.6 seconds for the b-matching method, respectively.

To verify our judgement that kDN is more reliable on these two datasets, we compute the retrieval precisions of kNN, kDN and b-matching method under different k on these two datasets, and the results are illustrated in Fig. 10. The curves clearly validate our hypothesis. On the USPS dataset, the precision of the proposed method is the best, which explains the lowest error rate in Table 1. On the TEXT dataset, the precision of b-matching is a bit higher than our method, which coincides with the fact that b-matching has the lowest error rate when the number of labeled points is 100 in Table 1. On the USPS dataset, although the precision of kNN is a bit higher than the b-matching method, the error rates of kNN are always larger than the b-matching method. The reason may be that the nearest neighbors of b-matching are symmetric (if a is one of the nearest neighbors of b, then b is also one of the nearest neighbors of a), while the nearest neighbors of kNN are not guaranteed to be symmetric. The precision curves also reveal the reason why kDN and b-matching method make more significant improvements on the TEXT dataset, this is because the precision of nearest neighbors improves more significantly on the TEXT dataset.

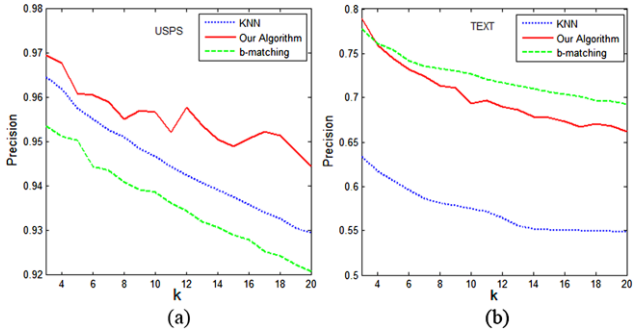

Fig. 10 (Color online) Precision curves of kNN, b-matching method and our algorithm on USPS and TEXT datasets. *Red solid curve* shows the precision of our algorithm, *blue dotted curve* illustrates the precision of kNN, while *green dashed curve* illustrates the precision of b-matching method

Especially when $k = 12$, on the USPS dataset, the precisions of kNN and kDN are 94.26% and 95.78%, respectively, and the improvement is only 1.52%. While on the TEXT dataset, the precisions of kNN and kDN are 56.46% and 68.95%, respectively, and the improvement reaches up to 12.49%.

5.3 Cluster Analysis

The solution of (8) can be viewed as a local coherent cluster around object o. For a dataset $O = \{o_1, \ldots, o_n\}$ with n objects, we can get n such local clusters and these clusters constitute a set, denoted by $C = \{C_1, \ldots, C_n\}$, with C_i is the local cluster around the object o_i. As stated before, (8) has a good property: the parameter δ offers us a tool to control the minimum size of a cluster. We can estimate the minimum size of a cluster in advance and set δ accordingly. If it is hard to estimate, we can simply set $\delta = 1$, which means that no constraint is put on the size of a cluster.

Since the value of $g(x)$ is a good approximation of the average affinity of the cluster x and the average affinity measures the goodness of a cluster, $g(x)$ is then a good indicator of whether x represents a real cluster or not. The larger $g(x)$ is, the more probable x is a real cluster. Thus, we can check all clusters with large values of $g(x)$, to verify whether they are real clusters or not. In this way, we can detect real clusters underlying the data. Note that some points, such as outliers, may not belong to any clusters with large value of $g(x)$. This is because these points cannot form coherent clusters with other points. This property is very important and it implies that our method is very robust to outliers, since it can automatically exclude outliers. In fact, when there is many outliers, insisting on partitioning all points into coherent clusters usually leads to bad results.

5.3.1 Point Set Matching

Establishing correspondences between two point sets is a long standing fundamental problem in computer vision (Caetano et al. 2006; Georgescu and Meer 2004). In general, the point set matching problem can be described as follows: given two point sets, P and Q, with n_P and n_Q points, respectively, the task is to obtain the correct correspondences between them. This is a hard problem and there are three main difficulties associated with this problem. First, the mapping between points in two point sets may be one-to-one, one-to-many, and even many-to-many. Second, it is inherently a combinatorial optimization problem, thus suffers from high complexity due to the huge dimensionality of the combinatorial search space. The product space $H = P \times Q$ contains $n_P n_Q$ candidate correspondences, which is usually very large. Finally, the existence of outliers must be considered. Since in many applications, the number of outliers is very large, sometimes even much larger than the number of inliers.

Each candidate correspondence $h_i \in H$ is a pair $(P_i, Q_{i'})$, where $P_i \in P$ and $Q_{i'} \in Q$. Since the points themselves are not distinctive, we rely fully on the geometric consistency information to find the correct correspondences, and the compatibility function of a pair of candidate correspondences, (h_i, h_j), is defined as follows:

$$f_1(h_i, h_j)(s) = \begin{cases} 1 - \frac{(l_{ij} - s l_{i'j'})^2}{9\sigma_d^2} \\ \quad \text{if } i \neq j \text{ and } |l_{ij} - s l_{i'j'}| < 3\sigma_d; \\ 0 \quad \text{otherwise,} \end{cases} \tag{15}$$

where l_{ij} is the distance between P_i and P_j, $l_{i'j'}$ is the distance between $Q_{i'}$ and $Q_{j'}$, and s is a scale factor. The parameter σ_d controls the sensitivity of the function value to deformations. The larger the σ_d is, the more deformations we can accommodate, but also the more pairwise relationships between incorrect correspondences will get positive values.

Based on the candidate correspondence set H, the correspondence graph G is constructed as follows: each vertex of G represents a candidate correspondence in H, the weight of the edge between vertex i and vertex j is set to $f_1(i, j)$. Note that the weights of all self-links are 0. The correspondence graph G usually has a large number of vertices (the number of candidate correspondences); however, the number of vertices corresponding to correct correspondences are usually very small, and most of vertices are outliers (incorrect correspondences).

From each vertex (candidate correspondence) of the correspondence graph G, we can find a local cluster. The clusters obtained from all vertices then form the cluster set C. Since the weights of all self-links are zero, thus $A_{ii} + A_{jj} \leq 2A_{ij}$ for all i and j. In such situation, the size of each cluster will be automatically determined by Algorithm 1 and probably larger than the minimum size we set. For each cluster in C with large average affinity, we can verify whether it is a real correspondence configuration or not by checking the

Algorithm 3 Dense Neighborhood based Point Set Matching Algorithm

Input: Two point sets P and Q, the compatibility function f_1 and ϵ.
According to f_1, construct the correspondence graph G.
Set $C = \emptyset$ and $M = \emptyset$;
for $i = 1, \ldots, n$ **do**
    For the vertex i, construct the initialization by ϵNN, and then obtain a local cluster by Algorithm 1. Add this cluster into the set C.
end for
for each cluster $C_i \in C$ with large average affinity **do**
    Verify whether it is a correct correspondence configuration, if yes, add C_i into the set M.
end for
Output: All the correct correspondence configurations in M.

coherence of all correspondences. If each pair of correspondences in C are coherent, that is, with large weights, then C is a real correspondence configuration, otherwise not. In this way, we can find all correct correspondence configurations, which is summarized in Algorithm 3.

Note that $f_1(h_i, h_j)$ only conveys partial information: small value of $f_1(h_i, h_j)$ means that h_i and h_j should not be in the same correspondence configuration, but large value of $f_1(h_i, h_j)$ only indicates that h_i and h_j may be, not necessarily, in the same correspondence configuration. In fact, the number of edges with high affinities in the graph G is huge, and only a very small portion of them are edges between correspondences in the same correspondence configuration. This is because strong pairwise relations can be easily produced by noises and outliers. In such situation, for every candidate correspondence in H, there are many candidate correspondences with large affinities to it, and according to our heuristic rule, ϵNN is a much better choice than kNN to construct initializations. Thus, for point set matching, we adopt ϵNN to construct initializations and set $\epsilon = 0.1$ in all experiments.

We compare our proposed method with the spectral method in Leordeanu and Hebert (2005), which utilizes the largest eigenvector of the adjacency matrix A of the correspondence graph G to find correct correspondence. The experimental setting is as follows: first generate data set Q of 2D model points by randomly selecting n_Q^i inliers in a given region of the plane, then obtain the corresponding inliers in P by disturbing independently the n_Q^i points from Q with white Gaussian noise $N(0, \sigma)$, and then rotate and translate the whole data set Q with a random rotation and translation. Next we add n_Q^o and n_P^o outliers in Q and P, respectively, by randomly selecting points in the same region as the inliers from Q and P, respectively, from the same random uniform distribution over the x–y coordinates. The range of the x–y point coordinates in Q is $256\sqrt{n_Q/10}$ to enforce an approximately constant density of 10 points over a 256×256 region, as the number of points varies. The total number of points in Q and P are $n_Q = n_Q^i + n_Q^o$ and $n_P = n_P^i + n_P^o$. The parameter σ controls the level of deformation between two point sets, while n_P^o and n_Q^o control the numbers of outliers in P and Q, respectively. As pointed out in Leordeanu and Hebert (2005), it is a challenging problem because of the homogeneity of data and the large search space.

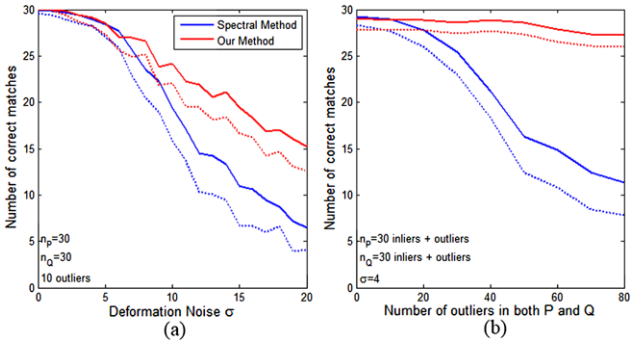

Fig. 11 (Color online) Performance curves for our method vs. the spectral method in Leordeanu and Hebert (2005). The *solid lines* show the mean performance, and the *dotted lines* show one *std* below the mean. Our method is shown *in red*, while the spectral method in Leordeanu and Hebert (2005) is shown *in blue*

To compare with the spectral method in Leordeanu and Hebert (2005), we do the experiments on the same scale, that is, we fix $s = 1$. Figure 11 compares the performance curves of our proposed method and the spectral method in Leordeanu and Hebert (2005). We keep the sensitivity parameter fixed, $\sigma_d = 5$. All algorithms ran on the same data sets over 30 trials for each value of the varying parameter, and both the mean performance curves and the curves of one standard deviation below the mean are plotted. We score the performances of these two methods by counting how many matches agree with the ground truths. In Fig. 11(a), we vary the noise σ from 0 to 20 (in step of 1). Obviously, our method works much better, since our method tries to obtain a cluster as coherent as possible, and such coherent cluster is less sensitive to noises. In Fig. 11(b), we keep the deformation parameter σ and the number of inliers fixed, and change the number of outliers from 0 to 80. The performance curve clearly shows the advantages of our method, which is nearly not affected by outliers, while the performance of the spectral method is continuously degrading as the number of outliers increases. This is because our method automatically excludes outliers, while spectral method cannot.

In Fig. 12(a), we fix the deformation parameter $\sigma = 5$, keep the number of inliers and outliers equal, and change the total number of points in P and Q. The performance is measured by matching rate, which is the ratio of the number of obtained correct correspondences to the number of all correct correspondences. As the figure shows, the perfor-

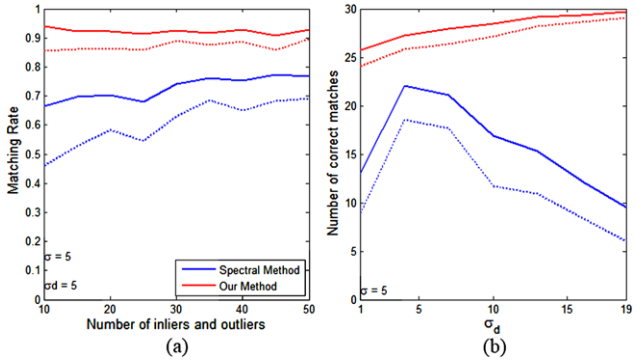

Fig. 12 (**a**) Demonstrates the performance curves as the number of inliers and outliers changes for our algorithm and the spectral clustering method, (**b**) shows the performance's sensitivity to the parameter σ_d

mance of all algorithms improves as the number of points increases, this is because as the size of common point pattern increases, the common point pattern is more robust to noises and outliers. In Fig. 12(b), we also test the sensitivity of both methods to parameter σ_d. In this experiment, both the number of inliers and outliers are 30. Obviously, our method is nearly not affected by the scaling parameter σ_d, while the spectral method is very sensitive to σ_d. Note that σ_d controls the weights of edges and many graph-based algorithms are notoriously sensitive to it. Instead, by setting a proper δ, our method overcomes this problem.

Our method can simultaneously detect multiple correspondence configurations. In the first row of Fig. 13, the point sets P and Q have two one-to-one correspondence configurations. In the second row, the point sets P and Q have one-to-two correspondence configuration, that is, P contains a cluster of points, and Q contains two similar copies of this cluster. Each correspondence configuration has 30 correspondences, and we test the performance of our method with varying deformation noises and varying number of outliers. The performance curves of the two correspondence configurations detected by our method are shown in red and blue, respectively. In this experiment, we fix $\sigma_d = 5$. As Fig. 13 shows, our method can simultaneously detect multiple correspondence configurations, and this experiment also shows encouraging robustness of our method to outliers.

In all previous experiments for point set matching, we set the minimum size of the cluster to be the number of correct correspondences in the correspondence configuration. To test the effect of δ, we fix σ and σ_d, and test the performance of our method under different δ. The result is shown in Fig. 14 and x axis represents $1/\delta$. Obviously, as $1/\delta$ approaches the real size of the cluster (30), the results become much better. Note that the best result appears when $1/\delta > 30$, which is due to the fact that some outliers fall into the clusters.

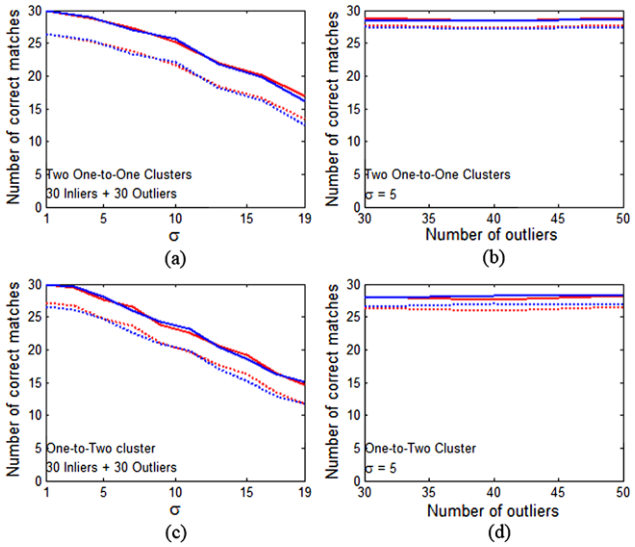

Fig. 13 Performance curves of our method on two one-to-one correspondence configurations and a one-to-two correspondence configuration. *First row*: two one-to-one clusters. *Second row*: a one-to-two correspondence configuration. The *red* and *blue solid curves* are the mean performance of these two mappings and the *dashed curves* are one *std* below the mean

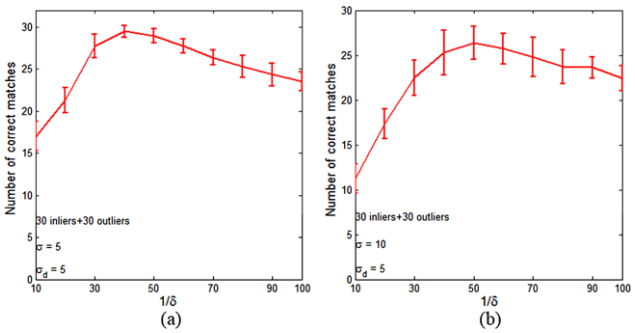

Fig. 14 Performance curve of our proposed method under different δ

5.3.2 *Clustering on Popular Datasets*

Clustering is in fact a labeling problem, that is, assigning cluster indices to objects. Precisely speaking, clustering is a mapping $\Psi : O \to L$, where $O = \{o_1, \ldots, o_n\}$ is a set of n objects and $L = \{l_1, \ldots, l_s\}$ is a set of s indices.

All possible assignments form a set, denoted by W. Since each object can be assigned any label, the number of assignments in W is ns, that is, $|W| = ns$. Every assignment w is a pair composed of an object and a label, denoted by o_w and l_w, respectively. In general, the distances between objects in the same cluster should be small, while the distances between objects in different clusters should be large. Thus, we can define the *compatibility score* of two assignments, w and v, in the following way:

$$f_2(w,v) = \begin{cases} \exp(-\frac{d^2(o_w,o_v)}{2\sigma_d^2}) & \text{if } l_w = l_v, \\ \exp(-\frac{\xi^2}{2\sigma_d^2 d^2(o_w,o_v)}) & \text{if } l_w \neq l_v, \end{cases} \quad (16)$$

where $d(o_w, o_v)$ is the Euclidean distance between o_w and o_v, σ_d is the scaling factor as in the experiments of point set matching, and ξ is a parameter to balance the influence of the distance under two situations: with the same or different labels.

Based on W, we can construct a graph G as follows: each vertex of G represents an assignment in W, and the weight of the edge between vertex i and vertex j is set to $f_2(i, j)$. Thus, the clustering problem of original data is transformed into the problem of finding a most coherent cluster on graph G, under the constraint that each object can only appear in on vertex.

From each vertex (an assignment) of the graph G, we can find a local coherent cluster. Obviously, this cluster is a good candidate for clustering. Since the size of obtained coherent cluster can be controlled by k, we can vary k to get clusters of different sizes. To precisely control the size of a cluster, in this task, we set the weights of all self-links to 1. Thus, the size of obtained cluster is precisely $k + 1$.

We conduct our experiments on four popular datasets for clustering, Iris, Wine, Breast-cancer and Australian. The first three datasets are in the UCI machine learning repository (Frank and Asuncion 2010) and the fourth dataset is from libsvm dataset (Chang and Lin 2001). The Iris dataset consists of 50 samples from each of three species of Iris flowers (Iris setosa, Iris virginica and Iris versicolor), thus, it has 150 samples in total. Four features were measured from each sample, they are the length and the width of sepal and petal, in centimeters. The Wine dataset contains 178 samples of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The number of samples in the first, second and third clusters are 59, 71, 48, respectively. And each sample contains the quantities of 13 constituents, which determine the types of wines. The Breast-cancer dataset has two classes, with the first class having 444 samples, and the second class having 239 samples. The Australian dataset also has two classes, with the first class having 383 samples, and the second class having 307 samples.

Since the kDN of each vertex (an assignment) can be considered to be a clustering, therefore, we only consider the kDNs of the first vertex as the cluster of such vertices, which corresponds to assign label 1 to the first data point in the dataset. Euclidean distance is used and the parameter σ_d is set to be the mean distance of 1NN of all data points. The parameter ξ is set to 1, 0.4, 3 and 4, respectively, on Iris, Wine, Breast-cancer and Australian dataset. Figures 15, 16, 17 and 18 illustrate the clustering results on Iris, Wine, Breast-cancer and Australian dataset, respectively. The precision is calculated based on data points which are assigned

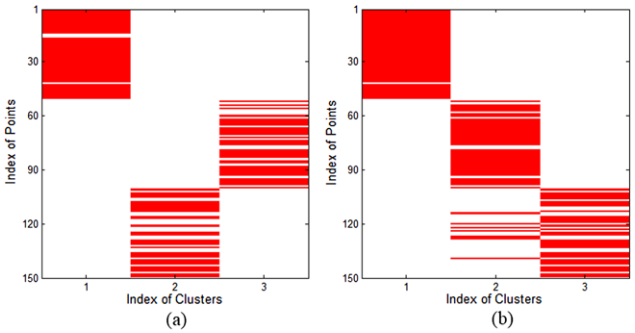

Fig. 15 Clustering result of the kDN of the first assignment (assigning label 1 to the first data point) on Iris dataset, with $k = 110$ and $k = 130$ in (**a**) and (**b**), respectively

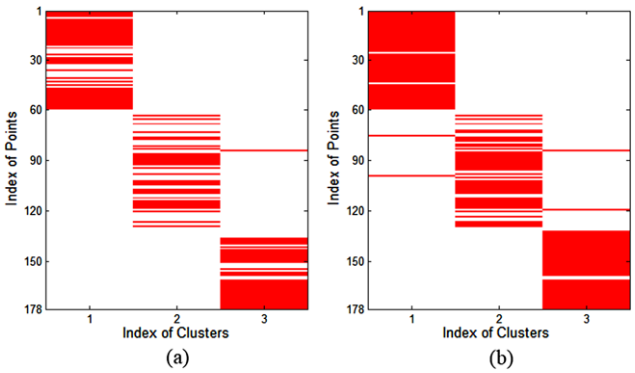

Fig. 16 Clustering result of the kDN of the first assignment (assigning label 1 to the first data point) on Wine dataset, with $k = 110$ and $k = 150$ in (**a**) and (**b**), respectively

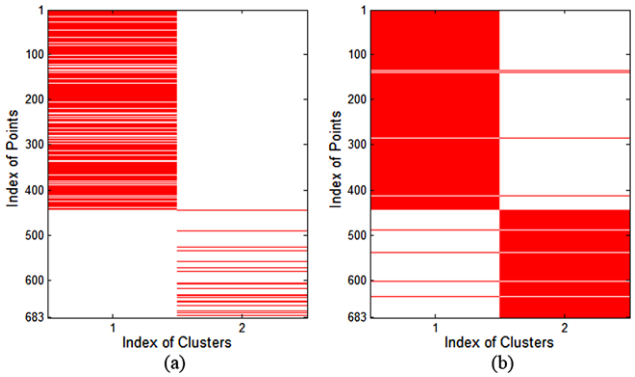

Fig. 17 Clustering result of the kDN of the first assignment (assigning label 1 to the first data point) on Breast-cancer dataset, with $k = 400$ and $k = 682$ in (**a**) and (**b**), respectively

labels, since our method only select $k+1$ points to label, and left other points un-grouped. In all four datasets, when k is small, the clustering precisions are very high. When k become larger, the performances drop. This phenomenon implies that our method has a very good property: it can automatically distinguish "easy" points and "hard" points and select the most easiest $k + 1$ points to cluster. The "hard"

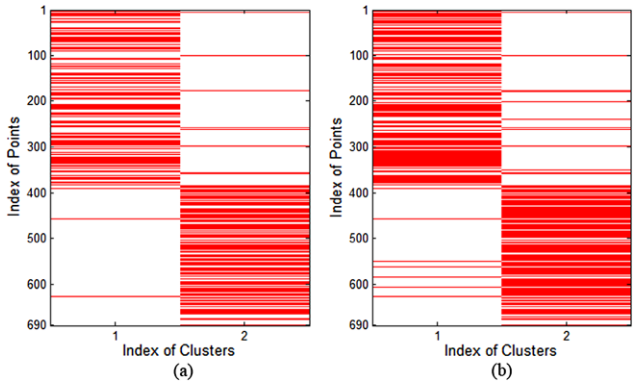

Fig. 18 Clustering result of the kDN of the first assignment (assigning label 1 to the first data point) on Australian dataset, with $k = 400$ and $k = 500$ in (**a**) and (**b**), respectively

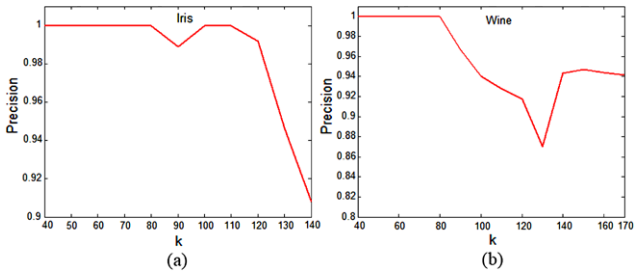

Fig. 19 Performance curves of the kDNs of the first assignment (assigning label 1 to the first data point) under varying k, on both Iris (**a**) and Wine (**b**) datasets

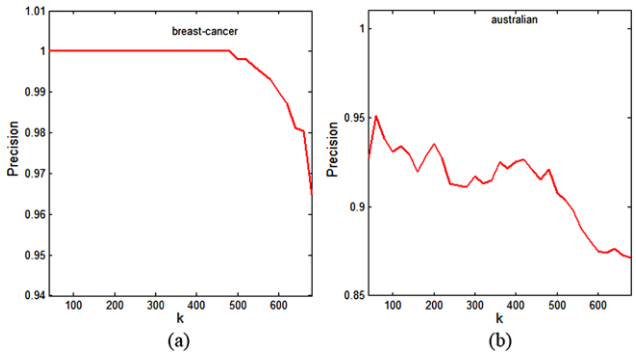

Fig. 20 Performance curves of the kDNs of the first assignment (assigning label 1 to the first data point) under varying k, on both Breast–cancer (**a**) and Australian (**b**) datasets

points are either outliers or points near boundaries of clusters, and our method offers a tool to discover them. Figure 15 also reveals a fact that in Iris dataset, the first cluster is well separated from the other two clusters, while the second and the third clusters are intervening each other.

In Figs. 19 and 20, we vary k and plot the performance curves of Iris, Wine, Breast-cancer and Australian dataset. As expected, with small value of k, since all the selected points are "easy" points, the precision of clustering is usually very high. As k increases, the precision drops, since "hard" points are gradually selected; however, it does not drop monotonously, and this effect may result from the varying strength of the ensemble effect under different k.

Although distinguishing between "easy" points and "hard" points is interesting and important in many applications, such as outlier detection, in some cases, we do need to classify every points into a cluster. In Fig. 18(b), for the Breast-cancer dataset, when $k = 682$, all points are clustered. The precision is 98.83% for our method. In comparison, on this dataset, the clustering precisions of k-means and spectral clustering are 96.05% and 97.07%, respectively. When some points are un-grouped, based on our method, it is easy to accomplish the clustering task: first clustering "easy" points to get high precision clusters, then simply assigning "hard" points to these clusters based on some criterion, such as 1NN. We cluster all data points in Iris, Wine and Australian datasets following this strategy, and set $k = 120$, $k = 150$ and $k = 500$ respectively. The overall clustering precision is 92.56% on Iris, 95.36% on Wine and 89.63% on the Australian dataset. In comparison, the clustering precision of k-means is 89.33% on Iris, 91.57% on Wine and 85.51% on the Australian dataset. This result demonstrates that, the clustered "easy" points may help to cluster "hard" points, thus much better results can be achieved.

6 Conclusions

In this paper, we define the notion of k-dense neighborhood (kDN), analyze its properties and propose an efficient approximate algorithm to compute it. We demonstrate the advantages of kDN over kNN and b-matching method on both affinity data and vectorial data. Because kDN is a local coherent cluster, it is a natural tool for cluster analysis. We propose a novel point set matching algorithm based on kDN and demonstrate its excellent performance, especially when there exists a large amount of outliers. At the same time, kDN based clustering method can automatically delineate the cluster structure underlying the data and discover outliers and border points. Good clustering results are achieved on four popular datasets, Iris, Wine, Breast-cancer and Australian. Currently we are working towards a more effective way of constructing initializations in order to increase the probability of the initializations lying in the attractive basin of the global maximum of (8).

References

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

Asahiro, Y., Hassin, R., & Iwama, K. (2002). Complexity of finding dense subgraphs. *Discrete Applied Mathematics*, *121*(1–3), 15–26.

Bentley, J. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, *18*(9), 517–525.

Bomze, M., & De Klerk, E. (2002). Solving standard quadratic optimization problems via linear, semidefinite and copositive programming. *Journal of Global Optimization*, *24*(2), 163–185.

Caetano, T., Caelli, T., Schuurmans, D., & Barone, D. (2006). Graphical models and point pattern matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *28*(10), 1646–1663.

Chan, T. (1998). Approximate nearest neighbor queries revisited. *Discrete & Computational Geometry*, *20*(3), 359–373.

Chang, C.-C., & Lin, C.-J. (2001). LIBSVM: a library for support vector machines.

Chapelle, O., Scholkopf, B., & Zien, A. (2006a) Semi-supervised learning.

Chapelle, O., Scholkopf, B., & Zien, A. (2006b). The Benchmark Data Sets. http://www.kyb.tuebingen.mpg.de/ssl-book/benchmarks.html.

Clauset, A., Moore, C., & Newman, M. (2008). Hierarchical structure and the prediction of missing links in networks. *Nature*, *453*(7191), 98–101.

Connor, K. P. M. (2010). Fast construction of k-nearest neighbor graphs for point clouds. *IEEE Transactions on Visualization and Computer Graphics*, *16*(4), 599–608.

Cover, T., & Hart, P. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, *13*(1), 21–27.

Crammer, K., Talukdar, P., & Pereira, F. (2008). A rate-distortion one-class model and its applications to clustering. In *Proceedings of the 25th international conference on machine learning* (pp. 184–191).

Denoeux, T. (2008). A *k*-nearest neighbor classification rule based on Dempster-Shafer theory. In *Classic works of the Dempster-Shafer theory of belief functions* (pp. 737–760).

Dhillon, I., Guan, Y., & Kulis, B. (2004). Kernel *k*-means: spectral clustering and normalized cuts. In *International conference on knowledge discovery and data mining* (pp. 551–556).

Fleishman, S., Cohen-Or, D., & Silva, C. (2005). Robust moving least-squares fitting with sharp features. In *ACM special interest group on graphics and interactive techniques* (pp. 552–560).

Frank, A., & Asuncion, A. (2010). UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

Frey, B., & Dueck, D. (2007). Clustering by passing messages between data points. *Science*, *315*(5814), 972–976.

Georgescu, B., & Meer, P. (2004). Point matching under large image deformations and illumination changes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *26*, 674–688.

Gibson, D., Kumar, R., & Tomkins, A. (2005). Discovering large dense subgraphs in massive graphs. In *Proceedings of the 31st international conference on very large data bases* (pp. 732–741).

Goldstein, M. (1972). *K*-nearest neighbor classification. *IEEE Transactions on Information Theory*, *18*(5), 627–630.

Gupta, G., & Ghosh, J. (2005). Robust one-class clustering using hybrid global and local search. In *Proceedings of the 22nd international conference on machine learning* (pp. 273–280).

Han, E., Karypis, G., & Kumar, V. (2001). Text categorization using weight adjusted *k*-nearest neighbor classification. In *Advances in knowledge discovery and data mining* (pp. 53–65).

Horton, P., & Nakai, K. (1997). Better prediction of protein cellular localization sites with the *k* nearest neighbors classifier. In *International conference on intelligent systems for molecular biology* (Vol. *5*, pp. 147–152).

Hu, H., Yan, X., Huang, Y., Han, J., & Zhou, X. (2005). Mining coherent dense subgraphs across massive biological networks for functional discovery. *Bioinformatics*, *21*, 213–226.

Indyk, P., & Motwani, R. (1998). Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the 30th annual ACM symposium on theory of computing* (pp. 604–613).

Jebara, T., & Shchogolev, V. (2006). B-matching for spectral clustering. In *European conference on machine learning* (pp. 679–686).

Jebara, T., Wang, J., & Chang, S. (2009). Graph construction and b-matching for semi-supervised learning. In *International conference on machine learning* (pp. 441–448).

Kolahdouzan, M., & Shahabi, C. (2004). Voronoi-based *k* nearest neighbor search for spatial network databases. In *Proceedings of the international conference on very large data bases* (pp. 851–860).

Korn, F., Sidiropoulos, N., Faloutsos, C., Siegel, E., & Protopapas, Z. (1996). Fast nearest neighbor search in medical image databases. In *Proceedings of the international conference on very large data bases* (pp. 215–226).

Kuhn, W., & Tucker, A. (1951). Nonlinear programming. In *Proceedings of 2nd Berkeley symposium* (pp. 481–492).

Leordeanu, M., & Hebert, M. (2005). A spectral technique for correspondence problems using pairwise constraints. In *International conference on computer vision* (pp. 1482–1489).

Li, L., Weinberg, C., Darden, T., & Pedersen, L. (2001). Gene selection for sample classification based on gene expression data: study of sensitivity to choice of parameters of the GA/KNN method. *Bioinformatics*, *17*(12), 1131–1139.

Ling, H., & Jacobs, D. (2007). Shape classification using the inner-distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *29*(2), 286–299.

Motzkin, T., & Straus, E. (1965). Maxima for graphs and a new proof of a theorem of Turán. *Canadian Journal of Mathematics*, *17*(4), 533–540.

Ng, A., Jordan, M., & Weiss, Y. (2001). On spectral clustering: analysis and an algorithm. In *Advances in neural information processing systems* (pp. 849–856).

Ouyang, Q., Kaplan, P., Liu, S., & Libchaber, A. (1997). DNA solution of the maximal clique problem. *Science*, *80*, 446–448.

Pavan, M., & Pelillo, M. (2007). Dominant sets and pairwise clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *29*(1), 167–172.

Shi, J., & Malik, J. (2000). Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *22*(8), 888–905.

Yu, C., Ooi, B., Tan, K., & Jagadish, H. (2001). Indexing the distance: An efficient method to *k*NN processing. In *Proceedings of the international conference on very large data bases* (pp. 421–430).

Zaki, M., Parthasarathy, S., Ogihara, M., & Li, W. (1997). New algorithms for fast discovery of association rules. In *International conference on knowledge discovery and data mining* (Vol. 20, pp. 283–286).

Zhou, D., Bousquet, O., Lal, T., Weston, J., & Schlkopf, B. (2004). Learning with local and global consistency. In *Advances in neural information processing systems* (pp. 595–602).

Zhu, X., Ghahramani, Z., & Lafferty, J. (2003). Semi-supervised learning using Gaussian fields and harmonic functions. In *International conference on machine learning* (Vol. 20, pp. 912–919).