XNN Graph

Pasi Fränti[1(✉)], Radu Mariescu-Istodor[1], and Caiming Zhong[2]

[1] University of Eastern Finland, Joensuu, Finland
franti@cs.uef.fi
[2] Ningbo University, Ningbo, China

Abstract. K-nearest neighbor graph (KNN) is a widely used tool in several pattern recognition applications but it has drawbacks. Firstly, the choice of k can have significant impact on the result because it has to be fixed beforehand, and it does not adapt to the local density of the neighborhood. Secondly, KNN does not guarantee connectivity of the graph. We introduce an alternative data structure called XNN, which has variable number of neighbors and guarantees connectivity. We demonstrate that the graph provides improvement over KNN in several applications including clustering, classification and data analysis.

Keywords: KNN · Neighborhood graph · Data modeling

1 Introduction

Neighborhood graphs are widely used in data mining, machine learning and computer vision to model the data. Few applications examples are listed below.

- KNN classifier
- Manifold learning
- 3D object matching
- Clustering
- Outlier detection
- Traveling salesman problem
- Word similarity in web mining

Two popular definitions are *ε-neighborhood* and *k–nearest neighbor* (KNN) graph [1]. In the first one, two points are neighbors if they are within a distance ε of each other. KNN neighborhood of a point is defined as its k nearest other data points in the data space. In the corresponding graph, all neighboring points are connected. The ε–neighborhood graph is undirected whereas KNN is directed graph.

The first problem, both in KNN and also ε-neighborhood, is how to select parameters ε and k. The larger the neighborhood, the better the local structures can be captured but the more complex the graph and the higher the processing time. An upper limit is when k equals to the size of the data ($k = N$), which leads to a complete graph. Having fixed value of k, there can be unnecessary long edges in sparse areas that do not capture anything essential about the local structure.

The second problem is that these definitions do not guarantee connectivity of the graph. In Fig. 1, an isolated component is created, which leads to wrong clustering

A. Robles-Kelly et al. (Eds.): S+SSPR 2016, LNCS 10029, pp. 207–217, 2016.
DOI: 10.1007/978-3-319-49055-7_19

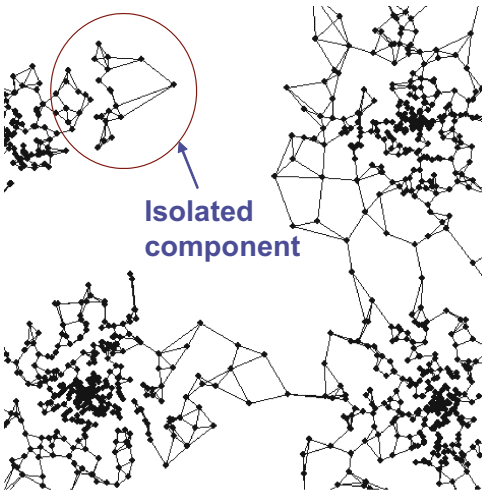

Fig. 1. Example part of 3NN graph where connectivity is broken.

result. With this data the problem can be solved by setting $k = 4$ at the cost of increased complexity. However, with higher dimensional data the value must usually be set to $k = 20$ or even higher. This overwhelms the computation.

In general, there is no good way to set k automatically, and it does not adapt to the local density. According to [2], k has to be chosen rather high, order of N rather than log$N$. In [3] this neighborhood was reduced by eliminating edges to nodes that were reached via alternative shorter detour via a third node. But it does not solve the connectivity and requires large k to start with. Despite these drawbacks, KNN has become popular – mainly because lack of better alternatives.

In this paper, we introduce a new neighborhood graph called *XNN*. The key idea is to model the local structures in the same way as KNN, but instead of having a fixed value of k, the size of neighborhood is variable and depends on the data locally. In a dense area, there would be more edges than in sparse areas so that both the local structures are captured and the connectivity of the graph is guaranteed.

2 Neighborhood Graphs

Besides KNN, another commonly used structure is *minimum spanning tree* (MST), which guarantees connectivity. However, it is optimized for minimizing the sum of all distances and may lead to complicated chains that do not capture the real structure of the data. MST contains only one edge per data point, on average. This corresponds to 1NN-graph by the size. It is enough to keep the graph connected but not able to capture more complex local properties such as density, or estimate clustering structure.

Good heuristic was proposed in [4] to use 2 × MST. Once the first MST is created, another MST is constructed from the remaining unused edges and the union of the two MSTs makes the neighborhood graph. Intuitively, kind of *second choices* are selected,

similarly as the 2nd nearest in 2NN-graph. This doubles the total number of edges in the graph, and can significantly improve the graph. The idea generalizes to any number of k by repeating the MST algorithm k.

The main advantage of k-MST is that it guarantees connectivity. However, it is still possible that important connections are missed leading to wrong analysis of the data because the connectivity may come via long chains that do not capture the local structure of the data. A good neighborhood graph would have the benefit of MST to keep the graph connected, but at the same time, having more edges to capture the structures both in sparse and dense areas. The choice of the value k remains an open problem also with k-MST.

Another well known structure is so-called *Delaunay triangulation*. It is based on *Voronoi* diagram, which is directly related to the partition concept in clustering. Clusters that share the same Voronoi vertex are considered as neighbors. The benefit is that the entire data space will be partitioned and the resulting graph will be naturally connected.

Slightly more practical variant is so-called *Gabriel graph* [5]. It is an old invention but deserves much more attention that it has gained because of two properties shown in [6]: (1) it includes MST as sub graph, and thus guarantees connectivity; (2) it is sub graph of the Delaunay, shares most of its properties but more practical. Elliptical variant was introduced in [7], and an additional density criterion considered in [8].

Both Delaunay and Gabriel graphs suffers the problem that the number of neighbors becomes too high when dimension increases. Another problem is the high time complexity of the graph construction. Delaunay has exponential dependency on the dimensionality, $O(N^{d/2})$, whereas Gabriel graph takes $O(N^3)$. These problems have been studied but general solution is still an open problem [9, 10].

3 X-Nearest Neighborh Graph

The resulting graph should have the following properties:

- Number of neighbors should be decided automatically.
- It should be small, preferably close to a constant.
- The graph will be connected.
- Constructing the graph can be done efficiently.

We define XNN graph as *any sub-graph of Gabriel graph that retains connectivity*, i.e. there are no isolated sub-graphs. We consider three alternatives:

- Full XNN
- Hierarchically-built XNN
- k-limited XNN

We give next their definitions and study their properties with data size, cluster overlap and dimension. Some other design alternatives for this principal idea are also discussed.

3.1 Full XNN

Input is a set of N data points $\{x_1, x_2, \ldots, x_N\}$ each consisting of d attributes. The points are often in a metric space so that the distance between any two points can be calculated. However, the idea generalizes to similarity measures as well.

Gabriel graph (GG) is defined for points in Euclidean space as follows. Any two points a and b are neighbors if the following holds for all other points c:

$$ab^2 < ac^2 + bc^2 \tag{1}$$

where ab, ac and bc are the distances of the particular points. We use this definition as the basis for the Full XNN as well. It generalizes to other distance measures that are in the range of [0,1] since the only numerical operation is the squared function. In specific, if the data is in Euclidean space then Full XNN equals to GG.

Alternative rule is to calculate the middle point between a and b, which we denote as m. If there exists another point c that is nearer to m than a and b, then this point must be within the circle. Thus, points a and b are neighbors only if they satisfy the following condition:

$$ab < 2mc \tag{2}$$

where m is the middle point (average of a and b) and c is the nearest point to m. The full graph can be found by the brute force algorithm:

Algorithm 1: Full XNN(data set) → XNN
FOR i=1 to N-1 DO
  FOR j=i+1 to N DO
  Calculate midpoint m ← (x_i+ x_j)/2
  x ← Find nearest point for m
  IF x== x_i OR x== x_j THEN
        Mark x_i and x_j as neighbors.

This formulation is slightly simpler than (1) and no squaring is needed. However, it may not work for non-numerical data because it requires that we can calculate average of any two data points. For instance, taking average of strings "*science*" and "*engineering*" is not trivially defined.

If we have similarities instead of distances, we can convert them to distances by inversion: $d = 1/(s + \varepsilon)$ where $s \in [0,1]$ is a similarity and $\varepsilon = 0.001$ is a small constant to avoid infinite. This leads to the following rule:

$$ab^{-2} < ac^{-2} + bc^{-2} \Leftrightarrow \frac{ab^2 \cdot bc^2 + ab^2 \cdot ac^2}{ac^2 \cdot bc^2} > 1 \tag{3}$$

The main bottleneck of the full graph is that it takes $O(N^3)$ time to compute. Another drawback is that there will be excessive number of neighbors in higher dimensional data.

3.2 Properties

We study next the number of neighbors with the following parameters: data size, dimensionality and overlap. We use the G2 data sets shown in Fig. 2.

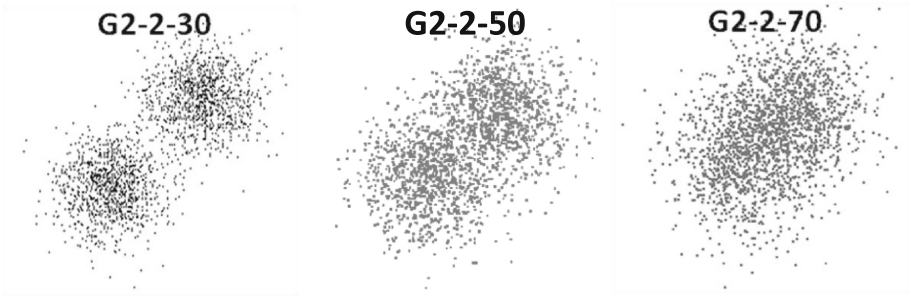

Fig. 2. Series of data sets G2-*D*-*O* where *D* is the dimensionality varying from 2 to 1024, and *O* is the overlap parameter varying from 0 to 100. Data size is N = 2048. http://cs.uef.fi/sipu/data/

Figure 3 shows that there is a mild tendency to have more neighbours when data has higher overlap (therefore more dense) but this effect happens only in higher dimensions. For 8-d data there is hardly any visible difference but for 32-d data the number of neighbours almost doubles when the clusters completely overlap compared to when they are separate.

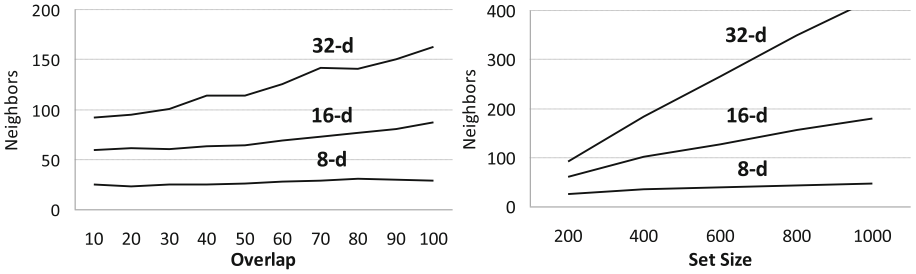

Fig. 3. Average number of neighbors for G2 with increasing overlap (left) and increasing data size (right). Results are for 10 % subsample (left) and overlap is 10 % (right)

The same effect happens when we vary the sub-sample size. For 32-d data, doubling the density (data size) doubles the size of the neighbourhood. The biggest effect, however, comes from the dimensionality, see Fig. 4. The number of neighbours is moderate up to 8-d but then increases quickly and reaches complete graph (1000 neighbours) at latest 256-d. If the clusters are well separated (30 % overlap), it becomes complete graph within the cluster (500 neighbours).

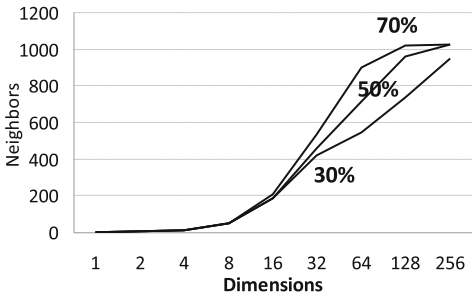

Fig. 4. Average number of neighbors with increasing dimension for G2. Three overlaps of 30 %, 50 % and 70 % were tested for 1 to 256 dimensions. Susample size of 1000 is used.

To sum up, for higher dimensional data we need to limit the size of neighbourhood to keep the graph useful.

3.3 Hierarchical XNN

Hierarchical variant is in Algorithm 2. The idea is to build the graph from clusters instead of data points. We start with only one cluster including all data points. New clusters are then iteratively created by splitting one existing cluster into two. This corresponds to divisive clustering algorithm. However, we do not stop at a certain number of clusters but continue the process until each point belongs to its own cluster. The splitting is done by taking two random points and by applying a few iterations of k–means within the cluster.

During the process, the clusters are represented by their centroids, and the distance between two clusters is calculated between them. The rule (2) is applied. If the centroid cannot be calculated from the data, then average distance (or similarity) between every pair of points in the two clusters and apply the rule (3):

$$\frac{1}{|a| \cdot |b|} \cdot \sum_{i \in a, j \in b} s_{ij} \tag{4}$$

At each dividing step, the XNN links are updated with two main principles. First, a link is always created between a and b. Second, the existing links to the previous cluster ab will be kept. These rules guarantee that if the graph was connected before the split, it will be connected after the split also. The first question is to which one the existing links should be linked to: a or b. We choose the nearest.

The second question is whether the neighbor should be connected to *both*. Consider the situation in Fig. 5. Greedy choice would be to choose only the nearest cluster (c-b, d-b). This would lead to a spanning tree: only one new link is created at every split, and there are N-1 splits in total. Result is a connected graph without redundant links, thus, spanning tree. The other extreme would be to accept both, and link all c-a, c-b, d-a d-b. However, this would lead to a complete graph, which is neither what we want.

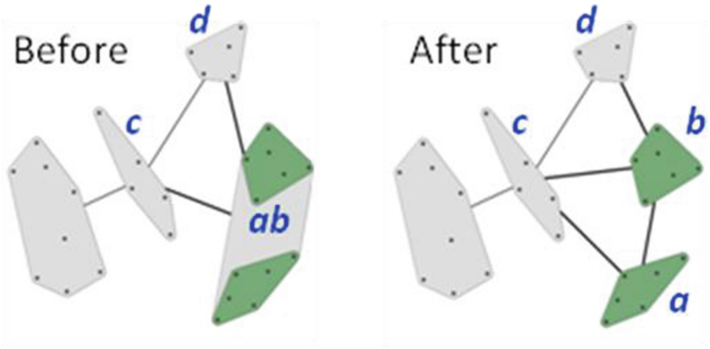

Fig. 5. Example of the hierarhical variant before and after the 4th split. Both *a* and *b* retain the connection to *c*, but only *b* retains the connection to *d*.

Our choice is to accept the second link if the neighbor rule applies. In principle, this can lead to Full XNN, but in practice, the split can have side-effects to other clusters, too. For example, new cluster may break the neighborhood of other clusters. Or vice versa, other clusters may interfere with the neighborhood of *a*, *b*, *c* and *d*. However, taking these side-effects into account would get the time complexity back to $O(N^3)$, which would be too much. We therefore choose to ignore these and settle with approximation. Some links of the Full XNN may therefore be missed, and some extra links may appear. But the key property remains: the graph is connected.

The split is implemented by local k-means. We select two random points in the cluster and iterate k-means 10 times within the cluster. This may not lead to optimal clustering but as it is merely a tool in the process it does not matter that much.

K-means requires $2{\cdot}10{\cdot}n = O(n)$ steps. Since the cluster sizes are progressively decreasing this sums up to $O(N{\cdot}\log^2 N)$ according to [11]. Updating the neighbor links depends on the number of links (X), and on the number of clusters:

$$X \cdot \sum_{i=1}^{N} i = X \cdot N^2 \tag{5}$$

The overall complexity is $O(N{\cdot}\log^2 N) + O(XN^2)$. If we further limited the neighbor rule by considering neighbor of neighbors, it would lead to $O(N{\cdot}\log^2 N) + O(X^2N^2)$.

Algorithm 2: Hierarchical XNN(data) → XNN
Put all in one cluster;
XNN ← ∅;
WHILE |XNN| < N DO
    ab ←SelectLargestCluster(C);
    a, b ← SplitCluster(*ab*);
    XNN ← XNN ∪ (a,b)
    UpdateXNN;

3.4 k-Limited XNN

The hierarchical variant improves the speed of creating the graph, but not necessarily the problem of having too many neighbors. We therefore modify it by having additional global constraint, k, to guide how many neighbors we expect to get. Considering the example in Fig. 5, we link every neighbor of ab to:

- The one which is closer (a or b)
- The remaining neighbors we start by taking the closest so that their total number won't increase $2k$.

Since a and b are linked together, they have one neighbor each already. Thus, $2(k–1)$ more links can still be chosen. Table 1 summarizes the actual numbers for the two variants. We can see that the Full XNN can have rather high number of neighbors with 16-dimensional data. The k-limited variant, on the other hand, keeps the size of neighbors always moderate.

Table 1. Average number of neighbors in the Full XNN and the k–limited (k = 10) variants. (Numbers in parentheses were run for smaller 10 % sub-sampled data)

Dataset	Dim	Full	Hierarchical	k-limited
Bridge	16	68.8	48	6.5
House	3	14.4	22	7.9
Miss America	16	345	94	6.8
Birch1	2	4.0	(3.4)	(3.4)
Birch2	2	(3.7)	(3.4)	(3.4)
Birch3	2	(3.9)	(3.4)	(3.3)
S1	2	3.8	3.4	3.3
S2	2	3.9	3.4	3.4
S3	2	3.9	3.4	3.4
S4	2	3.9	3.4	3.4

4 Applications

We next demonstrate the graph into three applications:

- Path-based clustering
- KNN classifier
- Traveling salesmen problem.

Path-based clustering [12] measures the distance of points in the dataset by finding the minimum path between two points. The cost of a path is defined as the maximum edge in the path. It can be calculated in Euclidean space or using neighborhood structure such as minimum spanning tree. *Robust Path-based clustering* method [13] uses KNN but it is an open question when to use small or large value of k. With some data the choice can be critical as shown in Fig. 6.

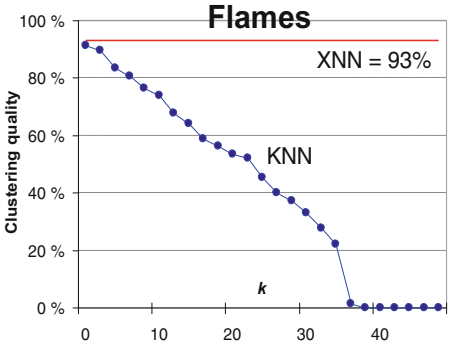

Fig. 6. Clustering quality of *Flames* by Path-based algorithm [13] measured by Pair set index [14].

Second test studied whether XNN has potential for solving traveling salesman problem. We analyzed known optimal solutions of several 2-d problem instances in *TSPLIB*. We counted how many times the same edges were in the optimal solution and also in the neighborhood graph constructed by KNN, k-MST and XNN. Results of KNN and k-MST were obtained by varying k, whereas XNN provides only one result. The aim would be to have as many correct edges in the graph as possible (to allow good optimization), but few other edges (to keep the complexity low). The results in Fig. 7 indicate that XNN has better precision/total ratio.

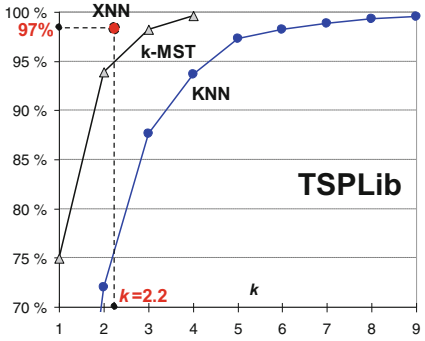

Fig. 7. The number of optimal edges captured by three graphs. The value of XNN is plotted at the location of the average number of neighbors (k = 2.2).

We have checked common edges between the optimal TSP and XNN on several datasets from TSPLIB, the result is: for some datasets, the total edges of XNN is kN, where k is a small number (about 2–3), but the kN edges contain the majority of the optimal TSP edges. This observation indicates that if we search the solution of TSP in XNN graph but not the complete graph, the efficiency could be high (Table 2).

Table 2. Number of edges of XNN graph, and how many are in optimal TSP path.

Dataset	Points	Edges		Common	
	N	Total	Per node	Total	Per point
eil101	101	229	2.3	98	97 %
a280	280	750	2.7	280	100 %
RAT575	575	1213	2.1	552	96 %
PR1002	1002	2060	2.0	957	96 %
PR2392	2392	5127	2.1	2306	96 %

The third application is classical KNN classifier. Figure 8 shows that the accuracy depends on k: with *Breast*, better results are obtained simply by increasing k, whereas with *Ionosphere* optimal result is between k = 20–25. In both cases, XNN classifier finds the same or better result without parameter tuning. Due to its simplicity, it could be worth further studies for example with distance weighted KNN [15].

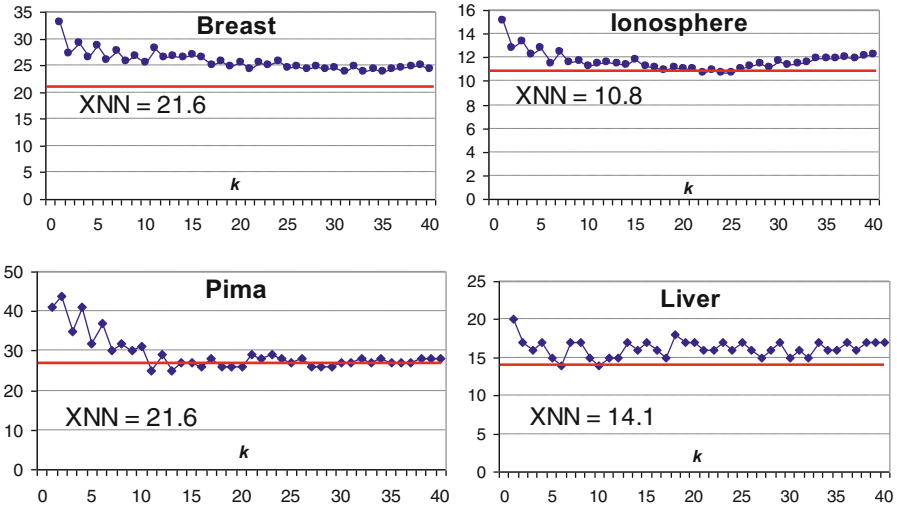

Fig. 8. Classification errors (%) of KNN (blue) and XNN (red); the lower the better. The result of KNN depends on k but XNN has only one value because it has no parameters.

5 Conclusion

New neighborhood graph called XNN is proposed. It is a compromise between fixed size nearest neighborhood graph of KNN, and a much more extensive spatial neighborhood of Gabriel graph. Its main advantages are that it is connected (no isolated components) and that the size of neighborhood is automatically selected for each point separately. Promising results were achieved in three example applications.

References

1. Cover, T., Hart, P.: Nearest neighbor pattern classification. IEEE Trans. Inf. Theory **13**(1), 21–27 (1967)
2. Maier, M., Hein, M., von Luxburg, U.: Optimal construction of kk-nearest-neighbor graphs for identifying noisy clusters. Theoret. Comput. Sci. **410**(19), 1749–1764 (2009)
3. Aoyama, K., Saito, K., Sawada, H., Ueda, N.: Fast approximate similarity search based on degree-reduced neighborhood graphs. In: ACM SIGKDD, San Diego, USA, pp. 1055–1063 (2011)
4. Yang, L.: Building k edge-disjoint spanning trees of minimum total length for isometric data embedding. IEEE Trans. Pattern Anal. Mach. Intell. **27**(10), 1680–1683 (2005)
5. Gabriel, K.R., Sokal, R.R.: New statistical approach to geographic variation analysis. Syst. Zool. **18**, 259–278 (1969)
6. Matula, D.W., Sokal, R.R.: Properties of Gabriel graphs relevant to geographic variation research and the clustering of points in the plane. Geogr. Anal. **12**(3), 205–222 (1980)
7. Park, J.C., Shin, H., Choi, B.K.: Elliptic gabriel graph for finding neighbors in a point set and its application to normal vector estimation. Comput. Aided Des. **38**, 619–626 (2006)
8. Inkaya, T., Kayaligil, S., Özdemirel, N.E.: An adaptive neighborhood construction algorithm based on density and connectivity. Pattern Recogn. Lett. **52**, 17–24 (2015)
9. Cignoni, P., Montani, C., Scopigno, R.: DeWall: a fast divide and conquer Delaunay triangulation algorithm in E^d. Comput. Aided Des. **30**(5), 333–341 (1998)
10. Rezafinddramana, O., Rayat, F., Venturin, G.: Incremental Delaunay triangulation construction for clustering. In: International Conference on Pattern Recognition, ICPR, Stockholm, Sweden, pp. 1354–1359 (2014)
11. Fränti, P., Kaukoranta, T., Nevalainen, O.: On the splitting method for VQ codebook generation. Opt. Eng. **36**(11), 3043–3051 (1997)
12. Fischer, B., Buhmann, J.M.: Path-based clustering for grouping of smooth curves and texture segmentation. IEEE Trans. Pattern Anal. Mach. Intell. **25**, 513–518 (2003)
13. Chang, H., Yeung, D.Y.: Robust path-based spectral clustering. Pattern Recogn. **41**(1), 191–203 (2008)
14. Rezaei, M., Fränti, P.: Set-matching methods for external cluster validity. IEEE Trans. Knowl. Data Eng. **28**(8), 2173–2186 (2016)
15. Gou, J., Du, L., Zhang, Y., Xiong, T.: A new distance-weighted k-nearest neighbor classifier. J. Inf. Comput. Sci. **9**(6), 1429–1436 (2012)