Pattern Recognition Letters 128 (2019) 551–558

Contents lists available at ScienceDirect

Pattern Recognition Letters

journal homepage: www.elsevier.com/locate/patrec

Fast and general density peaks clustering ☆

Sami Sieranoja*, Pasi Fränti

School of Computing, University of Eastern Finland, Box 111, Joensuu FIN-80101, Finland

ARTICLE INFO

Article history:
Received 16 November 2018
Revised 7 August 2019
Accepted 18 October 2019
Available online 19 October 2019

Keywords:
Data clustering
Density peaks
k-nearest neighbors (kNN)
Large-scale data

ABSTRACT

Density peaks is a popular clustering algorithm, used for many different applications, especially for non-spherical data. Although powerful, its use is limited by quadratic time complexity, which makes it slow for large datasets. In this work, we propose a fast density peaks algorithm that solves the time complexity problem. The proposed algorithm uses a fast and generic construction of approximate k-nearest neighbor graph both for density and for delta calculation. This approach maintains the generality of density peaks, which allows using it for all types of data, as long as a distance function is provided. For a dataset of size 100,000, our approach achieves a 91:1 speedup factor. The algorithm scales up for datasets up to 1 million in size, which could not be solved by the original algorithm at all. With the proposed method, time complexity is no longer a limiting factor of the density peaks clustering.

1. Introduction

Clustering algorithms aim at grouping points so that the points in the same group are more similar to each other than points in other groups. Clustering can serve as an efficient exploratory data analysis tool in the fields such as physics [1] and bioinformatics [2], or as a preprocessing tool for other algorithms in e.g. road detection [3] and motion segmentation [4].

Traditional clustering methods like k-means are constrained to cluster data with spherical clusters. Since the clusters in real life data are not always restricted to follow spherical shapes, new methods have been introduced to cluster data having arbitrary shape clusters. These include density based clustering [2,5,6], graph based methods [1,7,8], exemplar based clustering [9–11], and support vector clustering [12,13].

In this paper, we focus on the *Density peaks* (*DensP*) [6] clustering algorithm, which detects clusters based on the observation that cluster centers are usually in dense areas and are surrounded by points with lower density. The algorithm first calculates densities of all points, and then the distances to their nearest point with higher density (*delta*). The cluster centers are selected so that they have a high value of both delta and density. After that, the remaining points are allocated (*joined*) to the clusters by merging with the nearest higher density point.

The algorithm has been widely used for many applications, including autonomous vehicle navigation [3], moving object detection [4], electricity customer segmentation [14], document summarization [15] and overlapping community detection [16]. Although being popular, its use has been limited by the $O(N^2)$ time complexity. This slowness originates from two different bottlenecks: the need to calculate density, and to find the nearest neighbors with higher density. These make the algorithm slow for very large data sets.

Some attempts have been done to improve the speed of density peaks. Wang et al. [14] use sampling with adaptive k-means to reduce the number of data points. Xu et al. [17] also limit the size of data by using grid-based filtering to remove points in sparse areas. They reported speed-up factors from 2:1 to 10:1 for data of sizes $N = 5000–10{,}000$. However, both of these methods work only with numerical data, which reduces the generality of the original density peaks algorithm.

In addition to speed-up, there have also been attempts to improve the quality of density peaks. This has two major directions: using different density function [18–21], and using different strategies to allocate the remaining points to the clusters [20–22].

In the original density peaks algorithm, the densities are calculated by using a *cut-off kernel*, where neighborhood is defined by a given *cutoff distance* (*dc*). This defines a *dc*-radius hyper ball in the *D*-dimensional space. The algorithm then counts how many data points are within this ball.

Several authors have suggested alternatives to the cut-off kernel. Mehmood et al. [18] use a kernel variant based on

☆ Handled by editor Andrea Torsello.

* Corresponding author.
E-mail address: samisi@cs.uef.fi (S. Sieranoja).

https://doi.org/10.1016/j.patrec.2019.10.019

heat-diffusion. Du [19], Xie et al. [20] and Yaohui [21] all use a combination of exponential kernel and k nearest neighbors.

Xu et al. [22] proposed a novel joining strategy based on support vectors. Xie et al. [20] allocates the points using k nearest neighbors. The points are processed by a breadth first search starting from the cluster centers. Yaohui [21] proposed to join the points to the clusters if they are *density reachable*.

Most of the proposed approaches apply only to numerical data, which limits the usefulness of density peaks. However, the original density peaks algorithm does not have such limitation. Instead, it can operate with any given distance matrix regardless of the type of data. The only requirement is that some kind of distance function can be formulated. Du et al. [23] have used density peaks for a mix of categorical and numerical data, by introducing a new distance measure. Liu et al. [24] and Wang [15] use density peaks for text data by vectorizing the input data.

In this work, we focus on solving the slowness problem of the density peaks. We propose a new fast density peaks algorithm called *FastDP*. It first creates a *k-nearest neighbor* (kNN) graph. The density and delta values are estimated using this graph. The standard joining strategy of density peaks is then applied to obtain the final clustering. The proposed algorithm is significantly faster than the original density peaks while keeping its generality

Contrary to the existing attempts to speed up density peaks [14,22], our approach is not limited to numerical data but it applies, as such, to any type of data for which a distance function can be formulated. To demonstrate the algorithm's capabilities for non-vectorial data, we apply it for clustering of strings.

Our main contributions are the following:

- We present RP-Div algorithm to create kNN graph fast.
- We utilize the graph to calculate the delta values fast.
- We show that the algorithm applies also to text data.

In terms of the other aspects of the algorithm, we use the original point allocation (joining) strategy. For density estimation we use the k-nearest neighbors as proposed in [19].

2. Density peaks clustering

We first recall the original density peaks algorithm and its main variations. We use the following definitions:

x	Data point
N	Number of data points
K	Number of clusters
k	Number of neighbors in kNN graph.
$d(x,y)$	Distance between data points x and y
$kNN(x)$	The k nearest neighbors of x
$Dens(x)$	Density of the point x
$BigBrother(x)$	Nearest point y to x for which $Dens(y) > Dens(x)$
$Delta(x)$	Distance to $BigBrother(x)$
$Gamma(x)$	$= Delta(x) \cdot Dens(x)$
Local peak	Point x for which $BigBrother(x) \notin kNN(x)$
Slope	Point that is not a *Local peak*

2.1. Density

There are two widely used approaches to estimate density: *distance-based* and *kNN-based*. Distance-based approach takes a distance threshold as a parameter and counts how many points there are within this distance. Original density peaks algorithm uses this approach [6].

The second approach finds the k-nearest neighbors (k-NN), and then calculates the average distance to the neighbor points [19]. According to our experiments, there are no significant differences which of these approaches to use. They both require $O(N^2)$ calculations and provide virtually the same clustering result if correct parameter is used. However, good value for k is easier to determine than to find a suitable distance threshold [25]. We therefore use the kNN based approach for both the original variant of density peaks and the proposed fast variant.

2.2. Density peaks

The density peaks clustering algorithm [6] is based on the idea that cluster centers have usually a high density, and they are surrounded by points with lower density. So they have a large distance to points with higher density. Density peaks uses two features to determine the clusters: the *density* ρ and *delta* δ, which is the distance to the nearest point having a higher density. We denote this point as the *big brother*.

The original algorithm selects k points as the cluster centers based on ρ and δ. This is because cluster centers are expected to have a high value for both of them. However, it was not defined how exactly the selection should be made. Different strategies have therefore been considered by others. We denote the two most common as *Delta strategy* [26], which selects the points with highest delta values, and *Gamma strategy* [26,27], which uses the product of the two features ($\gamma = \rho\delta$). Here we apply the gamma strategy.

One also needs to decide how many points should be selected. The original paper did not give any solution and left it as a user-given parameter. Wang et al. [27] proposed to detect a knee point on the gamma values by finding the intersection of the two lines to most closely fit the curve. In general, the problem is how to threshold the selected feature (either δ or γ). This is an open problem both in centroids-based and density based clustering. Fig. 1 demonstrates the differences between the two selection strategies (delta and gamma). In this paper, we assume K is given by the user.

2.3. General distance

Density peaks has been mostly applied for numerical data in some vector space. However, it is possible to generalize the method to other non-numeric distance functions as well. Here we consider text data as a case study.

Two studies exist where string data has also been used. Liu et al. [24] and Wang [15] apply first string vectorization based on TF-IDF model. *Term frequency* (TF) counts how many times a given word appears in the dataset. It is normalized by counting *inverse document frequency* (IDF). This approach converts the strings into a vector space, and then uses cosine distance to measure the distance between the two strings.

The TF-IDF approach can be highly useful when clustering large text documents. However, short text strings contain only few words, which would result in sparse vectors containing only very few non-zero elements. Our solution to this is to apply a string similarity (or distance) measure directly without vectorization. This is possible because our method does not require the distance function being in metric space, nor does it rely on any other vector space properties.

The choice of the string similarity (or distance) measure depends on the application. We consider here two choices: Levenshtein and Dice. *Levenshtein edit distance* [28] is the most well known string distance measure, and we apply it in the context of short text strings. For tweets, we use *Dice coefficient* [29]. For a more comprehensive comparison of the available measures, we refer to [30].

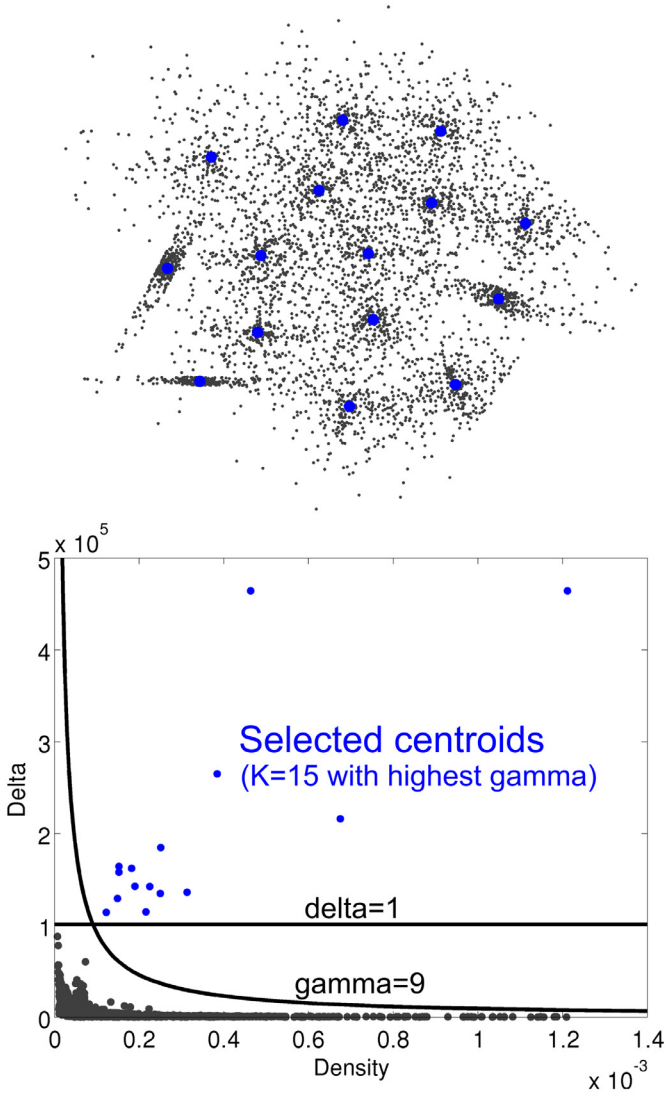

Fig. 1. Different cluster selection strategies based on the density-vs-delta plot for the S4 dataset. Cluster centroids typically have both high density and high distance to higher density point (delta). Therefore, thresholding based on combination of delta and density (gamma) is expected to work better than using the delta values alone.

3. Fast density peaks algorithm

The proposed *Fast density peaks* method is presented in Algorithm 1 and demonstrated in Fig. 2. Source code can be found on web.[1] It consists of five steps:

1. Create the kNN graph (line 1).
2. Calculate the density values (lines 2–3).
3. Find big brothers (line 6).
4. Identify cluster centers (lines 9–10).
5. Join the remaining points (lines 12–13).

Algorithm 1 FastDensityPeaks (X, k, K).

```
1   kNNg = RPDiv(X,k,2*k,1%);
2   FOR i = 1 TO Size(X) DO
3     density[i] = 1/meanDist(X[i],kNNg[i]);
4     part[i] = {i};
5
6   [bigBrother, gamma] = findBigBrothers(kNNg,X);
7   // Select K points with largest gamma
8    X = Sort(X,gamma);
9    FOR i = 1 TO K DO
10     centroid[i] = X[i];
11  // Join the remaining points to partitions
12  FOR i = K+1 TO N DO
13    Merge part[i] and part[bigBrother[i]];
14  RETURN [centroid, part];
```

First, we calculate the k nearest neighbor graph. The graph is then used to calculate all the information needed by density peaks algorithm: (1) density values, (2) distance to the nearest point with higher density (delta), and, (3) pointer to this nearest neighbor (*big brother*). Product of density and delta values (gamma) is used to determine the first K cluster centers (density peaks). The big brother pointers are then used to join the remaining points to the same cluster as their big brother.

The algorithm has two computational bottlenecks:

- Constructing the kNN graph
- Finding the big brother points.

Both of these bottlenecks take $O(N^2)$ time if straightforward solution is used. We next study how to make these two steps faster without sacrificing the quality of clustering.

[1] https://github.com/uef-machine-learning/fastdp.

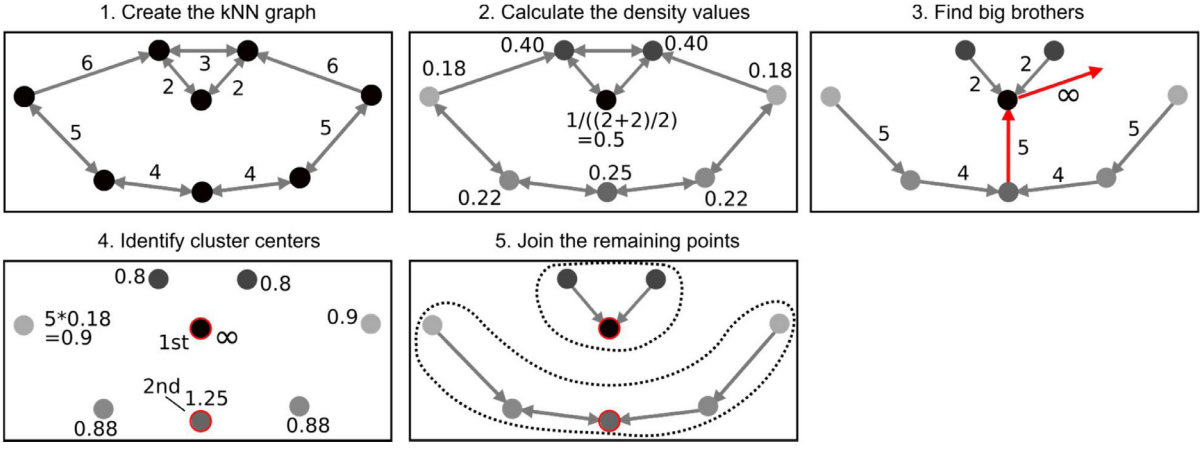

Fig. 2. Illustration of the proposed Fast density peaks algorithm. (1) For a given data set, the kNN graph is constructed. (2) Densities are calculated as inverse of the mean distance to the neighbors. (3) Nearest higher density point (big brother) is (in case of black lines) found in the kNN graph; for others (red lines) slower full search is performed. (4) Cluster centers are identified as the two points that have highest gamma (delta*dens) to the big brother. (5) Clusters are formed by joining other points to the same cluster as their big brother belongs to (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

3.1. Creating kNN graph by RP-div algorithm

To make density peaks faster, we first generate an approximate k-nearest neighbor (kNN) graph by using an iterative algorithm called *RP-Div* (Algorithm 3). Its preliminary version has been presented in [31]. The algorithm contains two loops. In the first loop (lines 1–4), we create a new candidate graph, which is used to improve the graph obtained from the previous iterations. The new graph is generated by an algorithm called *Random Pair Division* (*RP-div*) (Algorithm 4).

The algorithm constructs the graph by applying hierarchical subdivision (demonstrated in Figs. 3 and 4). The dividing works by first selecting two random points: *a* and *b* (Algorithm 4, lines 5-6). The dataset is then split into two subsets (A and B), so that subset A contains all points that are closer to the point *a*, and subset B all points that are closer to the point b. The dividing continues recursively (lines 12–13) until the subset size reaches a predefined threshold (*W*), e.g. $W < 20$. Subsets smaller than the threshold are solved by the $O(N^2)$ brute force algorithm (line 2), which calculates distances between all possible pairs and updates the list of *k* nearest points found (variable kNN).

Algorithm 2 findBigBrothers (kNNg, X, density).

```
1   FOR i = 1 TO Size(X) DO
2     bigBrotherFound = 0;
3     // See if big brother is found in graph
4     // Loop from nearest to farthest
5     FOR j = 1 TO numNeighbors DO
6       neighbor = kNNg[i][j];
7       IF density[i] < density[neighbor]
8         bigBrother[i] = neighbor;
9         bigBrotherFound = 1;
10        BREAK;
11    // For local peaks (not found in graph)
12    // Run O(N) full search
13    IF bigBrotherFound ==0
14      bigBrother[i] = fullSearch(i,density[i],X);
15    delta[i] = d(X[i],X[bigBrother[i]]);
16    gamma[i] = delta[i]*density[i];
17  RETURN [bigBrother, gamma];
```

Algorithm 3 RPDiv (X, k, W, stop).

```
1   REPEAT
2     RandomPairDivision(X,kNN,W);
3     diff = Changes(kNN);
4   UNTIL diff < 10%
5   REPEAT
6     RandomPairDivision(X,kNN,W);
7     NNDES(X,kNN);
8     diff = Changes(kNN);
9   UNTIL diff < stop;
10  RETURN kNN;
```

Algorithm 4 RandomPairDivision (X, kNN, Size).

```
1   IF size(X) < Size THEN
2     BruteForce(X,kNN);
3     RETURN;
4   ELSE
5     a = X[random(1,N)];
6     b = X[random(1,N)];
7   FOR i = 1 TO N DO
8     IF d(x,a) < d(x,b) THEN
9       A = A ∪ x
10    ELSE
11      B = B ∪ x;
12  RandomPairDivision(A,kNN,Size);
13  RandomPairDivision(B,kNN,Size);
```

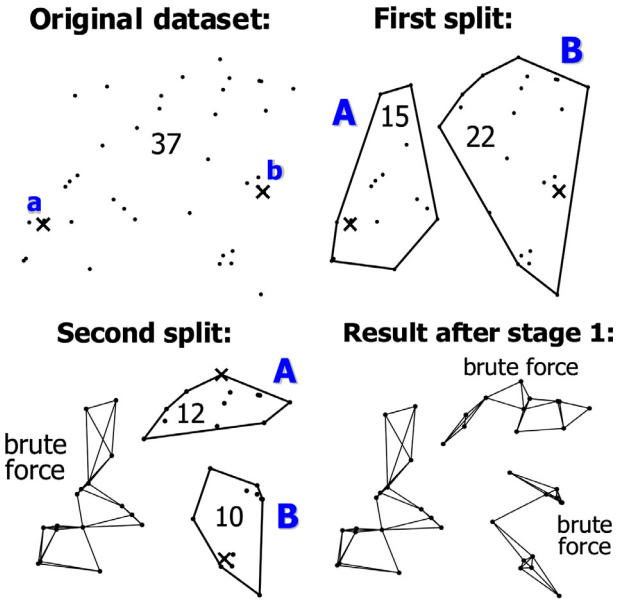

Fig. 3. The RP-div algorithm recursively subdivides the dataset of size $N = 37$ by first choosing two random points (a,b). The dataset is split based on which of the two points is nearer. After the first split, the size of the subset A is smaller than threshold $W = 20$, and is solved by brute force. The subset B is further divided into two more subsets, which both are smaller than *W* and now solved by brute force.

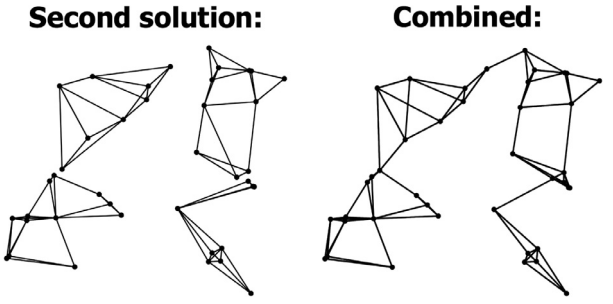

Fig. 4. After repeating the random pair division, a new solution is obtained. This is solution is merged with the previous one to form a new improved kNN graph.

One iteration of the algorithm produces a crude approximation of the kNN graph. The graph is improved by repeating the hierarchical subdivision several times (Fig. 4) using different random pairs for splitting. As a result, several different kNN graphs are created. Every new graph is used to improve the previous solution by adopting those k-nearest neighbor pointers that are shorter than in the previous graph.

The first loop (line 1) in Algorithm 3 continues until the proportion of changed edges drops below 10% (line 4). In the second loop (line 5), we apply the same iterative process as in the first loop, but at this time, we use the NNDES algorithm [32] to examine every point's neighbors of neighbors as kNN candidates. NNDES works better when the graph already has a moderate quality and is therefore used only at the later iterations. We continue until the improvement drops below **stop** = 1% (line 9).

Time bottleneck of the algorithm is the brute force step which requires $O(W^2)$. Assuming all subsets are exactly of size *W*, there will be *N/W* subsets. The total time complexity of single iteration of the algorithm is then $O(N/W \cdot W^2) = O(NW)$. Using $W = 2.5 \cdot k$, this leads to linear $O(rkN)$ time algorithm where the number of repeats (r) is a small constant.

3.2. Solving the Big brother pointers

Once the kNN graph is constructed, it can be used to speed up the bottlenecks of density peaks. The densities can be calculated

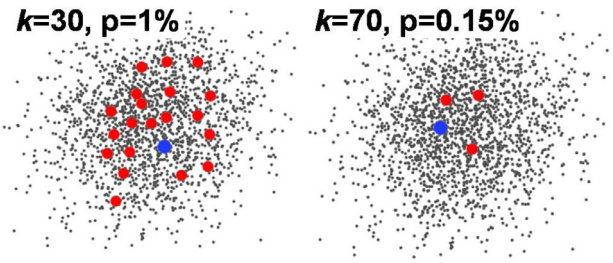

Fig. 5. Distribution of slope points (gray) and local peaks (red) inside an example cluster. One of the local peaks (blue) is the resulting cluster centroid (global peak). The case of $k = 30$ (left) and $k = 70$ (right) are shown. When the number of neighbors k in the kNN graph is increased, the number of local peaks decrease. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

trivially (lines 2–4 in Algorithm 1), and the nearest higher density point (big brother) can be found fast (Algorithm 2).

Finding the big brother (Algorithm 2) is a special case of the nearest neighbor search. However, instead of considering only the distance, we must also select a point with higher density. Fortunately, majority of points have their big brothers located within their kNN neighborhood (line 8). We call them as *slope points*, and all others are denoted as *local peaks*. For slope points, we can find big brothers fast in $O(k)$ steps simply by analyzing the kNN graph (lines 5–10).

Local peaks, on the other hand, are local density maxima and their big brothers cannot be found in the kNN graph. There is no shortcut to find their big brothers and $O(N)$ full search must therefore be performed (lines 11–14). These are also the points among which the final centroids will be chosen. The time complexity of this step depends on how many local peaks there are. Assuming that the proportion of the local peaks is p, the time complexity of this step is $pN^2 + (1 - p)kN = O(pN^2)$. The speed-up is therefore inverse proportional to p.

Fig. 5 shows an example of the distribution of the local peak points with a sample dataset. In general, the higher the value of k, the less there are peak points, and the faster the search for the big brothers.

The proportion of local peaks (p) is bound by the number of neighbors k in the graph. A neighbor of a peak cannot be another peak. If we have pN local peaks, there will be at least kpN slope points. Since all points are either local peaks or slopes, we have the following inequality:

$$kpN + pN < N$$

$$kp + p < 1$$

$$p < 1/(k+1)$$

$$p < 1/k$$

Therefore, the time complexity of $O(pN^2)$ can also be written in terms of k as $O(N^2/k)$. When combining with the $O(rkN)$ time complexity of the kNN graph construction (see Section 3.1), this leads to a total time complexity of $O(N^2/k + rkN)$ for the whole algorithm.

4. Experiments

We use parameters stop = 1% and $k = 30$ for kNN graph generation in FastDP algorithm, unless otherwise noted.

The experiments were run on Red Hat Enterprise Linux Server release 7.5 with 96 processing cores of Intel(R) Xeon(R) CPU E7-4860 v2 @ 3.20GHz and 1.0TB memory. Processing times are reported as run on single thread.

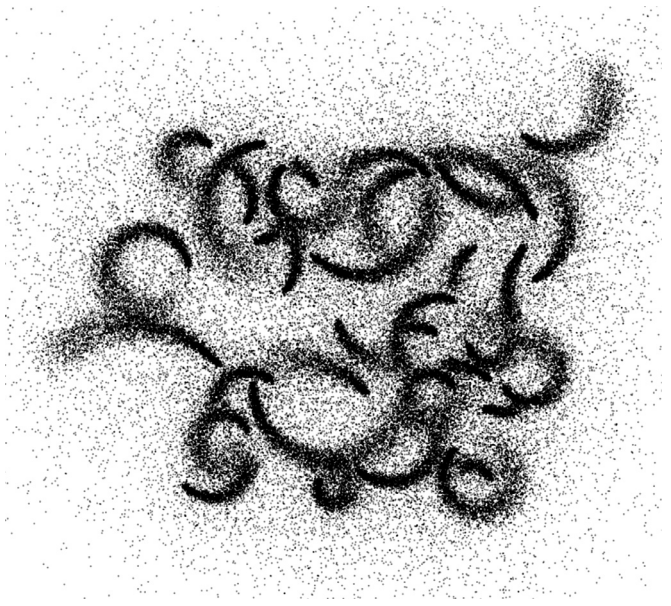

Fig. 6. The Worms2 dataset contains 35 shapes which depict trails of random movement in 2D space.

4.1. Datasets

We test the proposed algorithm with the following four different data types:

- Clustering basic benchmark
- High dimensional random clusters
- Artificial shapes
- Text datasets

We use datasets from the clustering benchmark [33]. So far we know four algorithms that can cluster all these datasets correctly: *global k-means* [34], *genetic algorithm* [35], *random swap* [36] and *density peaks* [6]. We test whether our fast density peaks (*FastDP*) can do the same. We report results for the S1–S4 sets [37] and for the Birch and Birch2 datasets [38].

To show the capabilities of the proposed method with large high-dimensional data, we generated three large *High dimensional random clusters* datasets, RC1M, RC100k-l and RC100k-l and RC1M. One hundred centroids were generated from uniform random distribution, each attribute in range [10..11]. For each cluster, 1000 (RC100k) or 10,000 (RC1M) points were drawn from Gaussian distribution. To study the effect of the cluster variance, we generated two variants for the RC100k dataset: RC100k-h for high variance ($\sigma^2 = 0.50$) and RC100k-l for low variance ($\sigma^2 = 0.05$). For the larger RC1M dataset, the lower variance of 0.05 was used.

Artificial shapes are also used as algorithms minimizing sum-of-squared errors cannot handle this type of datasets but density peaks often can. We use three datasets that the original density peaks is known to succeed: Flame [2], Aggregation [39], and Spiral [40]. Also the dataset DS6_8 provided by the authors of [8] was used.

Furthermore, we also generated two new artificial shape datasets: Worms2 (2D) and Worms64 (64D). The former is shown in see Fig. 6. The data contains 25 individual shapes starting from a random position and moving to a random direction. At each step, points are drawn from the Gaussian distribution to produce a cloud around the current position. The cloud has both a low variance (data) and high variance (noise) component. Their variance increases gradually at each step. The direction of movement is continually altered to an orthogonal direction. In case of the 64D

Table 1
Datasets used in the experiments. In case of text data, average number of characters is reported as dimensionality.

Dataset	Type	Size	Clusters	Dim.
S1	Spherical	5000	15	2
S2	Spherical	5000	15	2
S3	Spherical	5000	15	2
S4	Spherical	5000	15	2
Birch1 (B1)	Spherical	100,000	100	2
Birch2 (B2)	Spherical	100,000	100	2
RC100k-h (RCh)	Spherical	100,000	100	128
RC100k-l (RCl)	Spherical	100,000	100	128
RC1M (RCM)	Spherical	1,000,000	100	128
Flame (Fla)	Shape	240	2	2
Aggregation (Agr)	Shape	788	7	2
Spiral (Spi)	Shape	312	3	2
DS6_8 (DS6)	Shape	2000	8	2
Worms2 (W2)	Shape	105,600	35	2
Worms64 (W64)	Shape	105,000	25	64
Countries (Cou)	Text	6000	48	8.1 c
English words (Eng)	Text	466,544	500	9.4 c
Tweets (Twe)	Text	544,133	500	90 c

version, the orthogonal direction is selected randomly at each step. Collision is detected to prevent completely overlapping clusters.

Three text datasets are also used. Countries dataset has 48 clusters consisting of the names of all European countries. Each cluster contains 50 variations of the country name generated by adding random modifications to the names. English words dataset[2] contains 466,544 words of length varying from 1 to 45 characters (9.4 chars on average). Twitter data consists of tweets collected from Nordic Tweets channel [41]. For the Countries and words datasets, we use edit distance. For the twitter data, we use Dice coefficient of bigrams, which is faster than edit distance, especially for long strings (Table 1).

4.2. Clustering quality

We use the *Centroid Index* (CI) to measure the success at cluster level [42], and *Normalized Mutual Information* (NMI) at point level [43]. For the current state-of-the-art in measuring clustering quality we refer to [44].

Given a ground truth solution (G) and clustering solution (C), centroid index counts how many real clusters are missing a center, or alternatively, how many clusters have too many centers. The CI-value is the higher of these two numbers [42]. Value CI = 0 means that the clustering is correct.

The calculation of CI is done by performing *nearest neighbor mapping* between the clusters in C and G based on centroid distances [42]. After the mapping, we count how many clusters were *not* mapped. These non-mapped clusters (*orphans*) are indicators of missing clusters. The mapping is done into both directions (C→G and G→C). The maximum number of orphans is the CI-value:

$$CI(C, G) = \max\left(Orphans(C \rightarrow G), Orphans(G \rightarrow C)\right) \quad (1)$$

In case of shape and text data, center is not a proper representative of a cluster. We therefore find the most similar cluster instead of the nearest centroid. For this, we use Jaccard coefficient, which calculates how many common points the two clusters have to the total number of unique points in the two clusters [45]:

$$S(a, b) = \frac{|a \cap b|}{|b \cup b|} \quad (2)$$

Normalized mutual information measures the information that the clustering solution (C) shares with the ground truth (G). Since there scale has no upper bound, the result is normalized by the average of the self-entropies of C and G. The better the clustering, the closer the value of NMI is to 1. The exact scale varies across the datasets and it lacks similar intuitive interpretation as the CI-value.

The English words and Twitter data do not have any ground truth, so for them we only provide qualitative samples to estimate the clustering quality.

[2] https://github.com/dwyl/english-words.

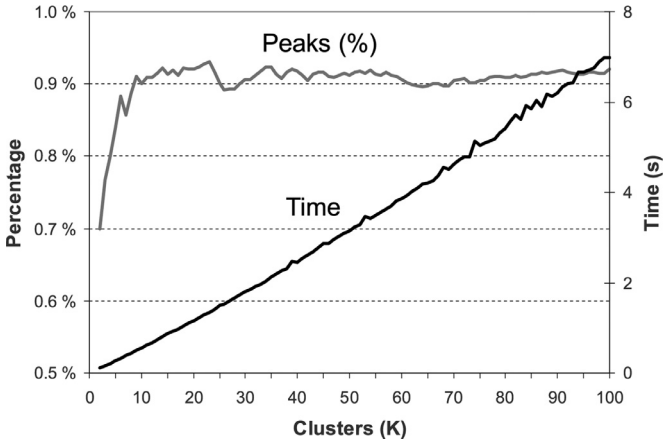

Fig. 7. Dependency of the proportion of local peaks on the number of clusters (K). The corresponding Fast-DensP processing time depends linearly on the data size, which is $N = 1000 \cdot K$.

Table 2
Proportion of local peaks p. When the number of neighbors k in the kNN graph is increased, the proportion of the local peak points decreases.

	Local peaks (p)		
Dataset	$k = 10$	$k = 30$	$k = 70$
S1	2.7%	0.3%	0.3%
S2	3.0%	0.3%	0.3%
S3	3.4%	0.5%	0.3%
S4	3.7%	0.6%	0.3%
Birch1	4.8%	1.1%	0.3%
Birch2	4.3%	0.9%	0.2%
RC100k	0.5%	0.2%	0.1%
RC1M	0.2%	0.1%	0.0%
Flame	2.5%	0.8%	0.8%
Aggregation	4.4%	1.5%	0.6%
Spiral	1.3%	1.0%	0.3%
Countries	1.3%	0.7%	0.2%
English words	1.7%	0.2%	0.0%
Tweets	0.7%	0.1%	0.0%

4.3. Efficiency of the delta calculation

We test the efficiency of finding the big brothers by studying the number of local peak points. We need to perform O(N) time full search only for the local peak points. For all other points, its big brother can be found faster in O(k) time simply by taking the nearest higher density point in its kNN neighborhood. Therefore, the less local peak points, the faster the algorithm.

Fig. 7 shows the proportion of the local peaks for the subsets of the Birch2 where one cluster was removed at a time to produce subsets with number of clusters varying from $K = 1$–100. The corresponding data sizes varies from $N = 1000$ to 100,000. We observe that the proportion of the peaks increases to about 0.9% at $K = 10$ clusters. After that it remains almost constant no matter how many more clusters there are. The total processing times are also shown, and it has near-linear dependency on the size of data.

Table 3
Summary of the processing times and clustering quality. The quality of the kNN graph is varied by running the RP-Div algorithm for different number of iterations. Quality is recorded as NMI. We highlight the first NMI value that is equal (within 0.01 NMI) to the results of OrigDP. The processing times and CI-values are reported for this iteration.

FastDP	NMI																	
Iterations	s1	s2	s3	s4	B1	B2	RCh	RCl	RCM	Fla	Agg	Spi	DS6	W2	W64	Cou	Eng	Twe
1	0.97	0.90	0.76	0.69	0.87	0.92	0.05	0.97	0.54	0.64	0.86	0.02	0.52	**0.61**	0.14	0.51	–	–
2	**0.99**	**0.94**	**0.79**	**0.72**	**0.95**	**1.00**	0.16	**1.00**	0.57	0.91	0.96	0.25	**0.59**	0.63	0.43	0.70	–	–
3	0.99	0.94	0.79	0.72	0.96	1.00	0.27	1.00	0.57	**0.99**	0.98	0.77	0.59	0.63	0.57	0.76	–	–
4	0.99	0.94	0.79	0.72	0.96	1.00	0.37	1.00	0.57	1.00	**1.00**	**1.00**	0.60	0.62	0.62	**0.78**	–	–
5	0.99	0.94	0.79	0.72	0.96	1.00	0.44	1.00	**0.57**	1.00	1.00	1.00	0.60	0.62	0.65	0.79	–	–
10	0.99	0.94	0.79	0.72	0.96	1.00	0.65	1.00	0.57	1.00	1.00	1.00	0.60	0.62	**0.71**	0.79	–	–
20	0.99	0.94	0.79	0.72	0.96	1.00	**0.82**	1.00	0.57	1.00	1.00	1.00	0.60	0.62	0.74	0.80	–	–
30	0.99	0.94	0.79	0.72	0.96	1.00	0.82	1.00	0.57	1.00	1.00	1.00	0.60	0.62	0.73	0.80	–	–
OrigDP	0.99	0.94	0.79	0.72	0.96	1.00	0.80	1.00	–	1.00	1.00	1.00	0.60	0.62	0.72	0.78	–	–
	Processing Time (seconds)																	
	s1	s2	s3	s4	B1	B2	RCh	RCl	RCM	Fla	Agg	Spi	DS6	W2	W64	Cou	Eng	Twe
FastDP	0.04	0.04	0.04	0.04	2.25	1.96	82.97	15.68	713	0.01	0.04	0.01	0.02	3.19	26.47	0.30	2091	1765
OrigDP	0.56	0.57	0.55	0.56	197	282	2656	2658		0.00	0.02	0.00	0.09	210	1310	6.67	–	–
Speedup factor	14:1	15:1	14:1	14:1	87:1	144:1	32:1	170:1	–	0:1	1:1	0:1	6:1	66:1	49:1	22:1	–	–
	CI																	
	s1	s2	s3	s4	B1	B2	RCh	RCl	RCM	Fla	Agg	Spi	DS6	W2	W64	Cou	Eng	Twe
FastDP	0.0	0.0	0.0	0.0	0.0	0.0	18.3	0.0	0.0	0.0	0.0	0.0	3.1	7.5	0.0	10.8	–	–
OrigDP	0.0	0.0	0.0	0.0	0.0	0.0	18.0	0.0	–	0.0	0.0	0.0	3.0	7.0	0.0	14.0	–	–

Table 4
Example clusters. Some of the 500 clusters for the 466,544 words data. Twenty samples from each cluster.

Cluster 41	Cluster 43	Cluster 247	Cluster 292
soft-bil	Livingstone	Slommacky	Kurtz
lsoot-grime	herringbone	crummock	Dinarchy
dsweet-toothe	Burlingham	bummack	myriarchy
dsplit-tongue	Neowashingtonia	mimmock	freshly
dblack-visage	Upington	slammock	triarcuated
dsoft-winge	Hillingdon	bummalos	matriarch
dshort-witte	Lovington	mimmocky	mandriarch
dshort-terme	Arlington	malmock	dyarchic
dstout-arme	Lexington	hommocks	myriarch
dstill-fishin	Herington	earthgrubber	BSLArch
gstiff-limbe	Stringtown	crumhorn	Taxiarch
dswift-stealin	Arrington	malbrouck	Bush
gshort-leave	milliangstrom	krumhorn	Ruthi
dsnotty-nose	Accrington	shammocky	Knuth
divory-bille	Northington	Babcock	fleshy
dhot-mettle	Farlington	plumrock	gush
dsoft-goin	Ellington	fleadock	Thushi
gsnowy-winge	Hartington	cummock	Furth
dsharp-tastin	Belington	Commack	flesh-fly
gstove-warmed	Conyngham	archworker	Bosch

Table 5
Two example clusters from twitter data. Four samples from each cluster. Detected clusters had typically low variation and were mostly produced by bots.

Jobs cluster	Weather cluster
We're #hiring! Click to apply: Technical Specialist - https://t.co/SP1NMxyDhp #Engineering Västra Götaland County #Job #Jobs	shower rain → light shower snow temperature down 3 °C → 2 °C humidity down 64% → 60% wind 9 kmh → 12 kmh
See our latest #kirkkonummi #job and click to apply: SW Developer Intern, IoT Device and Data Management -... https://t.co/5GEkyiMUlh	overcast clouds → light rain temperature down 6 °C → 5 °C humidity up 75% → 93%
If you're looking for work in #Sandarne, Gavleborg County, check out this #job: https://t.co/KwMj7Mp6rn #Netherlands #Labor #Hiring	broken clouds → clear sky temperature up 10 °C → 13 °C humidity down 66% → 54%
Interested in a #job in #HKI, Uusimaa? This could be a great fit: https://t.co/DMYvpOl54i #IT #Hiring #CareerArc	light intensity drizzle rain → scattered clouds temperature up 9 °C → 12 °C humidity down 100% → 66% wind 6 kmh → 11 kmh

The effect of the parameter k in kNN is shown in Table 2. With all datasets, the number of local peaks is small already with $k = 10$, and reduces to about 0.26% if it is increased to $k = 70$.

4.4. Results (Speed v. quality)

We implemented two versions of DensP in C: the original version [6] denoted as *OrigDP*, and the proposed method denoted as *FastDP*. Both variants use the same kNN-based method for density estimation. In terms of clustering quality, the only difference originates from the quality of the kNN graph. In the OrigDP, an exact kNN is used while FastDP uses approximated version from [31].

In the experiments, we vary the number of iterations in RP-DIV (1…30) to obtain different clustering quality. Quality of the approximated kNN graph increases with the number of iterations, and the same is expected to happen for the clustering quality.

From the results in Table 3, we can see that FastDP achieves similar quality as OrigDP but much faster. This is especially true with large datasets (B1, B2, RCh, RCl, W2, W64) where the O(N^2) time complexity of OrigDP results in high speedup factors. With similar size datasets, dimensionality and variance (cluster overlap) has large effect on the results. In case of W2 and W64, the high dimensional version requires much more iterations. In case of RCl vs. RCh, the high variance version requires more iteration.

Overall, the proposed method is much faster than OrigDP. In case of the birch2 dataset ($N = 100{,}000$), the processing time is reduced from 282 s to 1.96 s with no reduction on quality. In case of smaller datasets, there is 14:1 improvement in case of S-sets of size 5000, and no improvement in case of smaller datasets (<1000 points).

The proposed algorithm was also successful with the largest datasets (RC1M, English words, Tweets) which the original density peaks algorithm could not process. The 466,544 strings of the words dataset were clustered using Levenshtein distance to 500 clusters in 2091 s. Constructing the kNN-graph was the main bottleneck with 1779 s. This data set does not have a ground truth

clustering, but one can verify by seeing Table 4 that the results contain meaningful clusters.

The Nordic tweets dataset of size 544,133 was also successfully clustered with parameters $k = 100$, stop = 5%, and NNDES disabled. Two sample clusters are shown in Table 5, which both are meaningful for a human observer. In specific, the two particular clusters were observed to have lower variance, which can be partly explained by the fact that they were produced by bots rather than humans. The weather cluster is produced by one single bot, whereas the jobs cluster contains also human written tweets.

5. Conclusions

Fast density peaks (FastDP) algorithm was proposed. Its main advantage is that it removes the quadratic time complexity limitation of density peaks and allows clustering of very large datasets. The speed-up is achieved without any visible effect on the clustering quality. Experiments showed an average speed-up of 91:1 on datasets of size $\geq$ 100k. Clustering a high dimensional numerical dataset of size 1M took only 12 min. The algorithm works for all types of data as long as a distance function is provided. We managed to cluster a Nordic Tweet dataset of size 500k in 31 min.

Declaration of Competing Interest

Authors declare that they have no conflict of interest.

References

[1] S. Fortunato, Community detection in graphs, Phys. Rep. 486 (3–5) (2010) 75–174.
[2] L. Fu, E. Medico, FLAME, a novel fuzzy clustering method for the analysis of DNA microarray data, BMC Bioinformatics 8 (1) (2007) 3.
[3] K. Lu, S. Xia, C. Xia, Clustering based road detection method, in: 34th Chinese Control Conference (CCC), 2015, IEEE, 2015, pp. 3874–3879.
[4] Y. Zhang, G. Li, X. Xie, Z. Wang, A new algorithm for fast and accurate moving object detection based on motion segmentation by clustering, in: IAPR Int. Conf. on Machine Vision Applications (MVA), 2017, pp. 444–447.
[5] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al., A density-based algorithm for discovering clusters in large spatial databases with noise, Kdd 96 (1996) 226–231.
[6] A. Rodriguez, A. Laio, Clustering by fast search and find of density peaks, Science 344 (6191) (2014) 1492–1496.
[7] C.-D. Wang, J.-H. Lai, J.-Y. Zhu, Graph-based multiprototype competitive learning and its applications, IEEE Trans. Syst., Man, Cybernet., Part C (6) (2011) 934–946.
[8] Y. Qin, Z.L. Yu, C.-D. Wang, Z. Gu, Y. Li, A novel clustering method based on hybrid k-nearest-neighbor graph, Pattern Recognit. 74 (2018) 1–14.
[9] B.J. Frey, D. Dueck, Clustering by passing messages between data points, Science 315 (5814) (2007) 972–976.
[10] C.-D. Wang, J.-H. Lai, C.Y. Suen, J.-Y. Zhu, Multi-exemplar affinity propagation, IEEE Trans. Pattern Anal. Mach. Intell. (9) (2013) 2223–2237.
[11] I.A. Maraziotis, S. Perantonis, A. Dragomir, D. Thanos, K-Nets: clustering through nearest neighbors networks, Pattern Recognit. 88 (2019) 470–481.
[12] A. Ben-Hur, D. Horn, H.T. Siegelmann, V. Vapnik, Support vector clustering, J. Mach. Learn. Res. 2 (Dec) (2001) 125–137.
[13] C.-D. Wang, J. Lai, Position regularized support vector domain description, Pattern Recognit. (3) (2013) 875–884.
[14] Y. Wang, Q. Chen, C. Kang, Q. Xia, Clustering of electricity consumption behavior dynamics toward big data applications, IEEE Trans. Smart Grid 7 (5) (2016) 2437–2447.
[15] B. Wang, J. Zhang, Y. Liu, Y. Zou, Density peaks clustering based integrate framework for multi-document summarization, CAAI Trans. Intell. Technol. 2 (1) (2017) 26–30.
[16] X. Bai, P. Yang, X. Shi, An overlapping community detection algorithm based on density peaks, Neurocomputing 226 (2017) 7–15.
[17] X. Xu, S. Ding, T. Sun, A fast density peaks clustering algorithm based on pre-screening, in: Big Data and Smart Computing (BigComp), 2018 IEEE International Conference on, IEEE, 2018, pp. 513–516.
[18] R. Mehmood, G. Zhang, R. Bie, H. Dawood, H. Ahmad, Clustering by fast search and find of density peaks via heat diffusion, Neurocomputing 208 (October 2016) 210–217.
[19] M. Du, S. Ding, H. Jia, Study on density peaks clustering based on k-nearest neighbors and principal component analysis, Knowl. Based Syst. 99 (2016) 135–145.
[20] J. Xie, H. Gao, W. Xie, X. Liu, P.W. Grant, Robust clustering by detecting density peaks and assigning points based on fuzzy weighted K -nearest neighbors, Inf. Sci. 354 (2016) 19–40.
[21] L. Yaohui, M. Zhengming, Y. Fang, Adaptive density peak clustering based on K nearest neighbors with aggregating strategy, Knowl. Based Syst. 133 (2017) 208–220.
[22] X. Xu, S. Ding, T. Sun, A fast density peaks clustering algorithm based on pre-screening, in: Big Data and Smart Computing (BigComp), 2018 IEEE International Conference on, IEEE, 2018, pp. 513–516.
[23] M. Du, S. Ding, Y. Xue, A novel density peaks clustering algorithm for mixed data, Pattern Recognit. Lett. 97 (2017) 46–53.
[24] H. Liu, H. Guan, J. Jian, X. Liu, Y. Pei, Clustering based on words distances, Cluster Comput. (2017) 1–9.
[25] P. Fränti, S. Sieranoja, Dimensionally distributed density estimation, in: Int. Conf. Artificial Intelligence and Soft Computing (ICAISC), Zakopane , Poland, June 2018, pp. 343–353.
[26] J. Hou, M. Pelillo, A new density kernel in density peak based clustering, in: Pattern Recognition (ICPR), 2016 23rd International Conference on, IEEE, 2016, pp. 468–473.
[27] J. Wang, Y. Zhang, X. Lan, Automatic cluster number selection by finding density peaks, in: Computer and Communications (ICCC), 2016 2nd IEEE International Conference on, IEEE, 2016, pp. 13–18.
[28] V.I. Levenshtein, Binary codes capable of correcting deletions, insertions, and reversals, Soviet Phys. Doklady 10 (8) (1966) 707–710.
[29] C. Brew, D. McKelvie, Word-pair extraction for lexicography, in: Proceedings of the 2nd International Conference on New Methods in Language Processing, 2017, pp. 45–55.
[30] N. Gali, R. Mariescu-Istodor, D. Hostettler, P. Fränti, Framework for syntactic string similarity measures, Expert Syst. Appl. 129 (2019) 169–185.
[31] S. Sieranoja, P. Fränti, Fast random pair divisive construction of kNN graph using generic distance measures, Int. Conf. on Big Data and Computing (ICBDC), April 2018.
[32] W. Dong, C. Moses, K. Li, Efficient k-nearest neighbor graph construction for generic similarity measures, in: Proceedings of the 20th international conference on World wide web, ACM, 2011, pp. 577–586.
[33] P. Fränti, S. Sieranoja, K-means properties on six clustering benchmark datasets, Appl. Intell. 48 (12) (2018) 4743–4759.
[34] A. Likas, N. Vlassis, J.J. Verbeek, The global k-means clustering algorithm, Pattern Recognit. 36 (2003) 451–461.
[35] P. Fränti, Genetic algorithm with deterministic crossover for vector quantization, Pattern Recognit. Lett. 21 (1) (2000) 61–68.
[36] P. Fränti, Efficiency of random swap clustering, J Big Data 5 (13) (2018) 1–29.
[37] P. Fränti, O. Virmajoki, Iterative shrinking method for clustering problems, Pattern Recognit. 39 (5) (May 2006) 761–765.
[38] T. Zhang, R. Ramakrishan, M. Livny, BIRCH: a new data clustering algorithm and its applications, Data Min. Knowl. Discov. 1 (2) (1997) 141–182.
[39] A. Gionis, H. Mannila, P. Tsaparas, Clustering aggregation, ACM Trans. Knowl. Discov. Data (TKDD) 1 (1) (2007) 1–30.
[40] H. Chang, D.Y. Yeung, Robust path-based spectral clustering, Pattern Recognit. 41 (1) (2008) 191–203.
[41] M. Laitinen, J. Lundberg, M. Leving, A. Lakaw, Utilizing multilingual language data in (nearly) real time: the case of the Nordic tweet stream, J. Univ. Comput. Sci. 23 (2017) 1038–1056.
[42] P. Fränti, M. Rezaei, Q. Zhao, Centroid index: cluster level similarity measure, Pattern Recognit. 47 (2014) 3034–3045.
[43] T.O. Kvalseth, Entropy and correlation: some comments, IEEE Trans. Syst., Man Cybernet. 17 (1987) 517–519.
[44] M. Rezaei, P. Fränti, Set matching measures for external cluster validity, IEEE Trans. Knowl. Data Eng. 28 (2016) 2173–2186.
[45] P. Fränti, M. Rezaei, Generalized centroid index to different clustering models, in: Joint Int. Workshop on Structural, Syntactic, and Statistical Pattern Recognition (S+SSPR 2016), Merida, Mexico, LNCS 10029, November 2016, pp. 285–296.