PUBLICATIONS OF
THE UNIVERSITY OF EASTERN FINLAND

Dissertations in Forestry and Natural Sciences

UNIVERSITY OF EASTERN FINLAND

SAMI SIERANOJA

Clustering with kNN graph and k-means

Clustering with kNN graph and k-means

Sami Sieranoja

Clustering with kNN graph and k-means

Publications of the University of Eastern Finland
Dissertations in Forestry and Natural Sciences
No 401

University of Eastern Finland
Joensuu
2020

Grano Oy
Jyväskylä, 2020
Editor in-Chief: Pertti Pasanen
Editor: Matti Tedre
Sales: University of Eastern Finland Library
ISBN: 978-952-61-3644-8 (print)
ISBN: 978-952-61-3645-5 (PDF)
ISSNL: 1798-5668
ISSN: 1798-5668
ISSN: 1798-5676 (PDF)

Author's address:	Sami Sieranoja University of Eastern Finland School of Computing P.O. Box 111 80101 JOENSUU, FINLAND email: samisi@cs.uef.fi
Supervisors:	Professor Pasi Fränti, PhD. University of Eastern Finland School of Computing P.O. Box 111 80101 JOENSUU, FINLAND email: franti@cs.uef.fi
Reviewers:	Professor Emre Celebi, Ph.D University of Central Arkansas Department of Computer Science MCS 303 201 Donaghey Ave., Conway, AR 72035, USA email: ecelebi@uca.edu
	Professor Jyrki Kivinen, Ph.D University of Helsinki Department of Computer Science P.O. Box 68 00014 UNIVERSITY OF HELSINKI, FINLAND email: jyrki.kivinen@cs.helsinki.fi
Opponent:	Professor Julius Žilinskas Vilnius University Akademijos g. 4 LT-08663 Vilnius, Lithuania email: julius.zilinskas@mif.vu.lt

Sieranoja, Sami
Clustering with kNN graph and k-means
Joensuu: University of Eastern Finland, 2020
Publications of the University of Eastern Finland
Dissertations in Forestry and Natural Sciences 2020; No. 401
ISBN: 978-952-61-3644-8 (print)
ISSNL: 1798-5668
ISSN: 1798-5668
ISBN: 978-952-61-3645-5 (PDF)
ISSN: 1798-5676 (PDF)

Abstract

Increase in the amount, variety and complexity of data has made it more difficult to understand or process information. Data clustering provides one way to help in these challenges. The aim of clustering is to group objects of a dataset so that the objects in the same group are more similar to each other than objects in other groups. It can be used to summarize data, find patterns or preprocess data for other algorithms.

Thousands of clustering algorithms have been developed over the years and many new ones are introduced each year, but one of the first clustering algorithms, k-means, is still used very widely today. It is known to have problems, but it has been unknown in which conditions it works and when it fails. In this thesis, we study the situations when it succeeds and when it fails, investigating properties of the data such as dataset size, dimensionality and overlap of clusters. We also study the most common ways of improving performance of k-means, such as repeating the algorithm or using better initialization. We find that lack of overlap of clusters is the most common property of data that causes k-means to fail. Combining a better initialization technique with repeating the algorithm 100 times can reduce errors from 15% to 1%.

Large datasets are problematic for many clustering algorithms which become too slow or inefficient to cluster them. One way to speed up clustering algorithms is to use a structure called kNN graph which connects every object to the k most similar objects in the same dataset. In this thesis, we have developed two fast methods for constructing the kNN graph. The first is targeted for high dimensional data. It constructs the graph by using one-dimensional mapping with a Z-order curve. The second is more general and can work for any type of data where a distance function is defined. It works by hierarchical random point division. We use the kNN graph to speed up a clustering algorithm called Density Peaks. This faster variant can cluster data of one million in size and achieves a speedup of 91:1 for datasets of size 100,000 or more.

Universal Decimal Classification: 004.021, 004.421, 004.6, 004.93, 519.1, 519.237.8

Library of Congress Subject Headings: Data mining; Data sets; Big data; Cluster analysis; Automatic classification; Algorithms

Yleinen suomalainen ontologia: tiedonlouhinta; big data; klusterianalyysi; graafit; algoritmit

Acknowledgements

This thesis contains the results of research conducted at the School of Computing, University of Eastern Finland during 2017-2020. The University has provided an interesting work environment with the opportunity to meet many new people from different countries. These encounters have shaped the direction of my work and my outlook on life in general.

I would like to thank my supervisor Pasi Fränti, who gave me the possibility to start this journey and provided lots of support during the many challenges involved in PhD work. I am especially grateful for all the detailed comments and suggestions for improvements that greatly improved the quality of my work.

I wish to thank all those colleagues with whom I worked during these years, including Radu Mariescu-Istodor, Jiawei Yang, Gulraiz I. Choudhary, Himat Shah, Nancy Fazal, Masoud Fatemi, Jehad Aldahdooh, Tomi Kinnunen and Tiina Laatikainen.

I would also like to thank the pre-examiners of the thesis, Professors Emre Celebi and Jyrki Kivinen for their valuable feedback. Also, I am honoured to have Professor Julius Žilinskas as my opponent.

I would also like to thank my girlfriend Sini Pirinen, who has given me encouragement and strength to face the many challenging events during this journey.

I am also very grateful to my parents and my family whose support, care and teaching since my childhood has given me a solid foundation that has helped me in every moment of my life.

Joensuu, 4th November 2020
Sami Sieranoja

List of abbreviations

kNN	*k* nearest neighbors
SSE	Sum of Squared Errors
MSE	Mean Squared Errors
CI	Centroid Index
NMI	Normalized Mutual Index
ZNP	Z-order Neighborhood Propagation
RP-Div	Random Pair Division

List of original publications

P1 S. Sieranoja and P. Fränti, “Constructing a high-dimensional kNN-graph using a Z-order curve”, ACM Journal of Experimental Algorithmics, 23 (1), 1.9:1-21, October 2018. https://doi.org/10.1145/3274656

P2 S. Sieranoja and P. Fränti, “Fast and general density peaks clustering”, Pattern Recognition Letters, 128, 551-558, December 2019. https://doi.org/10.1016/j.patrec.2019.10.019

P3 P. Fränti and S. Sieranoja, “K-means properties on six clustering benchmark datasets.”, Applied Intelligence (2017), 1-17, 2018. https://doi.org/10.1007/s10489-018-1238-7

P4 P. Fränti and S. Sieranoja, “How much k-means can be improved by using better initialization and repeats?”, Pattern Recognition, 93, 95-112, 2019. https://doi.org/10.1016/j.patcog.2019.04.014

Throughout the thesis, these papers will be referred to by [**P1**]-[**P4**]. These papers are included at the end of this thesis by the permission of their copyright holders.

Author’s contribution

The idea of papers [**P1**]-[**P2**] originated from the author. The author implemented and tested all new methods and performed all experiences. The articles were jointly written with the co-author. The author generated all graphics for the articles and did most of the writing.

The idea of papers [**P3**]-[**P4**] originated from the co-author. The author implemented and tested all new methods and performed all experiences. The articles were jointly written with the co-author. The author generated most of the graphics for the articles and did a smaller part of the writing.

Table of contents

LIST OF FIGURES

1 Introduction

As part of the ongoing digitalization, there has been an increase in the amount, variety and complexity of data. In this setting, there arises many challenges in understanding data and processing it in an efficient way. Data clustering provides one way to help in these challenges.

Clustering algorithms aim at grouping objects of a dataset so that the objects in the same group are more similar to each other than objects in other groups. Clustering can serve as an efficient exploratory data analysis tool in 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].

Clustering can work on any type of data such as images [5], text [**P2**,6,7], products [8], people [1] or genes [9]. Different algorithms have different limitations, but generally the only thing that is needed for clustering to work is a way of calculating either similarity or distance between the data objects.

Many clustering techniques can be divided into a cost function which defines the goal of clustering and an algorithm which optimizes the cost function [10]. The Sum of squared errors (SSE) is one of the most well known cost function. Algorithms that optimize this include k-means [11,12], Ward's method [13], Genetic algorithm [14] and Random Swap [15]. Some clustering algorithms are heuristics that do not optimize any clearly defined goal or cost function. These include the Density Peaks algorithm [5], DBSCAN [16] and Mean Shift [17].

K-means optimizes the SSE by first selecting an initial *k* random data points to represent the clusters. Then it iteratively fine-tunes the location of those points in a hill climbing manner, always improving the SSE in each iteration until convergence. Ward's method optimizes the SSE by first putting each point in separate clusters and repeatedly merging the pair of clusters with a smallest increase in SSE until there is only one cluster. From a user perspective, the main difference to k-means is that Ward's method returns clustering for all possible numbers of clusters, whereas for k-means the number of clusters needs to be fixed as a parameter.

One limitation of the traditional SSE optimizing algorithms like k-means is that it works well mainly with spherical data, i.e. data consisting of roughly ball shaped clusters, and cannot recognize non-spherical shapes like cigars [18], spirals or nested clusters [10]. Since the clusters in real life do not always follow spherical shapes, new methods have been introduced to cluster data having arbitrary shape clusters. These include density based clustering [16,5,2], graph based methods [1,19], exemplar based clustering [20,21], support vector clustering [22] and kernel k-means [23].

DBSCAN [16] and Density Peaks [5] are examples of density based heuristics. DBSCAN detects core points which lie in high density areas. It then merges points into the same cluster if they are within R-radius of a core point. All other points are considered outliers. The Density Peaks method forms clusters by first detecting peaks of density as cluster centers, i.e. those points that are located in a high density area and have large distance to higher density points. Other points are merged to the same cluster with the nearest higher density point.

1.1 Better clustering by using a kNN graph

The current trend of ever larger datasets is a problem for many clustering algorithms which become slow or inefficient for big datasets. In both Density Peaks and Ward's method, a major bottleneck is the calculation of the full distance matrix which requires $O(N^2)$ calculations and memory space. Spectral clustering has even higher complexity of $O(N^3)$ [24].

One way to improve clustering algorithms, both in terms of speed and quality, is to use a structure called kNN graph (Figure 1). It is a data structure where objects are connected to the *k* most similar objects in the same dataset. It has been used to speed up Ward's method to $O(n \log n)$ complexity [25]. Also DBSCAN has been enhanced using a kNN graph [26]. Density peaks has been previously improved by using a kNN graph in terms of quality [27], and recently in terms of speed also [**P2**]. In addition to their use in clustering, kNN graph has also many other applications such as classification [28], k-nearest neighbor search [29], dimensionality reduction [30], outlier detection [31] and computer graphics [32].

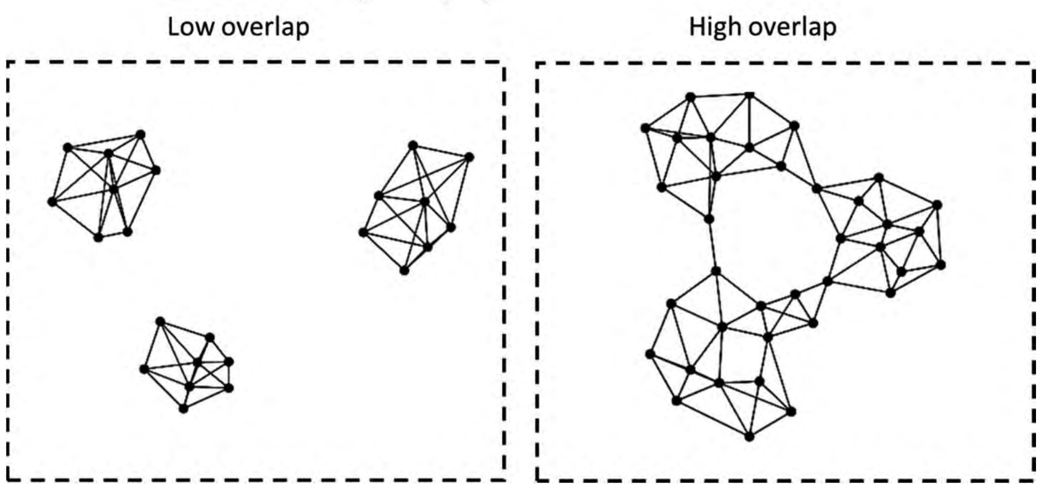


Figure 1. Constructing a kNN graph (k=4) for data can reveal the cluster structure of the data. In case of completely separated clusters (left), the connected components of the graph form the clusters. In case of more overlapping clusters (right), the cluster structure is more difficult to determine from the graph.

The trivial brute-force algorithm constructs a kNN-graph in $O(N^2)$ time by calculating distances between all pairs of points and selecting the k smallest distances for every point. This can be practical for small datasets consisting of up to tens of thousands of points. However, for larger datasets, consisting of millions of points, the brute-force algorithm becomes too slow.

In Chapter 3 we present two fast methods for constructing a kNN graph. The first, called *Z-order neighborhood propagation* (ZNP) [**P1**], uses one-dimensional mapping with a Z-order curve to construct an initial graph and then improves this using neighborhood propagation. This has been targeted for and tested with high dimensional data sets.

The second method, called *Random Pair Division* (RP-Div) [33], constructs an initial graph hierarchically by random point division and improves this using neighborhood propagation. It is more general than the ZNP method and can work with any type of data where a distance function is defined. In [**P2]**, we show a way of using it to speed up the Density Peaks algorithm.

1.2 Clustering with k-means

The k-means algorithm was introduced already in 1965 by Forgy [12]. And although thousands of clustering algorithms have been developed since then [10], k-means is still the most widely used. It is well known that k-means has problems [18,34-36]. For example, it does not work well with unbalanced data sizes [18,34,35] or when the data has outliers [18]. It has also been unclear which errors of k-means originate from the SSE cost function and which from the iterative process of the algorithm.

To counter the problems of k-means, it has been proposed to either repeat the algorithm multiple times [37] or use better initial centroids [38-41]. K-means++ [38] is the most well known initialization method and is almost as popular as the original k-means [**P4**]. Some of the initialization methods [39] are so complex that they can be considered new algorithms themselves.

One problem of k-means is that it may produce significantly worse results, in terms of the SSE cost function, compared to algorithms like Ward's or Random Swap. This problem is often referred to as k-means getting stuck in a local minimum [18], but it does not tell much about the problem. Since clustering is a NP hard problem, no practical algorithms can guarantee an optimal solution. Yet, k-means is still commonly used and quite often with satisfactory results. This raises the question: What are the conditions when k-means works and when does it fail?

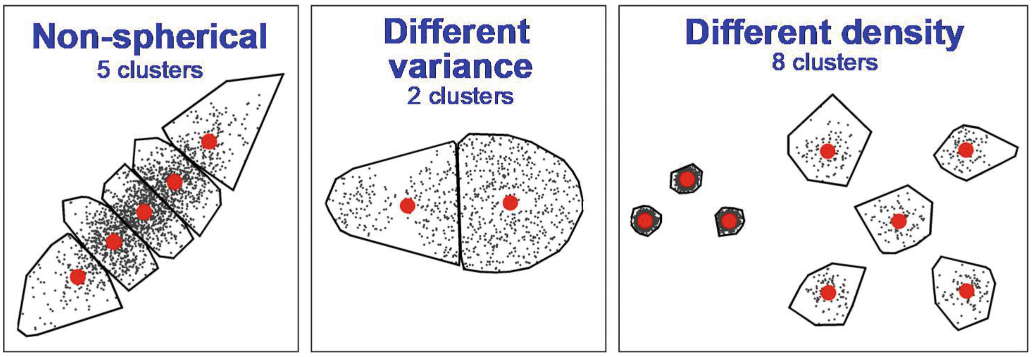


Figure 2. Three examples of clustering results when using the SSE cost function. A Gaussian cluster is split into several spherical clusters (left); mismatch of the variance causes the larger cluster to be split (middle); mismatch of cluster sizes does not matter if the clusters are well-separated. [**P4**]

Often, analysis of the properties of k-means can be misleading. Examples, like the first two cases in Figure 2, are often presented as problems of k-means [34,35]. However, these problems originate from the SSE optimization function. Even better algorithms fail to solve the correct clustering with these datasets if they minimize SSE. The problems of k-means algorithm itself are completely different and will be studied in this thesis.

A slightly better way to explain the problems of k-means is the claim of its strong dependency on the initial solution [10]. This is true, as k-means is a local fine-tuner that improves the given input solution. Originally only random partitions and random centroids initializations were considered as part of k-means. Later a large number of heuristic solutions have been proposed [38-50] to provide better initialization for k-means.

However, providing a good initialization is almost as difficult as the original clustering problem itself. The main problem for a clustering algorithm to solve is the cluster allocation problem, which is to allocate one centroid for each cluster. This is a combinatory problem in nature. If the centroids are located inside the correct clusters, k-means can surely tune the centroid locations (see Figure 3). The challenge is how to find the correct allocation.

Although the main challenge of clustering remains unsolved, there are several open questions that we can answer in this thesis: Are the existing initialization strategies any better than random choice? How much better? And in what conditions do they succeed?

Besides better initialization, another typical approach is to repeat k-means multiple times. This trick is also known as multi-start in optimization literature. It merely requires that there is randomness in the initialization so that different results can be obtained. It can indeed work when each repeat has some chance to find the correct clustering. For example, if we throw a dice aiming to get the number 6, we have only *p*=1/6 to succeed. However, if we repeat this process six times, the percentage of success has been increased already to $p'=1-(1-p)^6=66.5\%$. Repeating k-means works in a similar way, but there are some open questions: How much does the repeating improve results of k-means clustering? When does it work and when not?

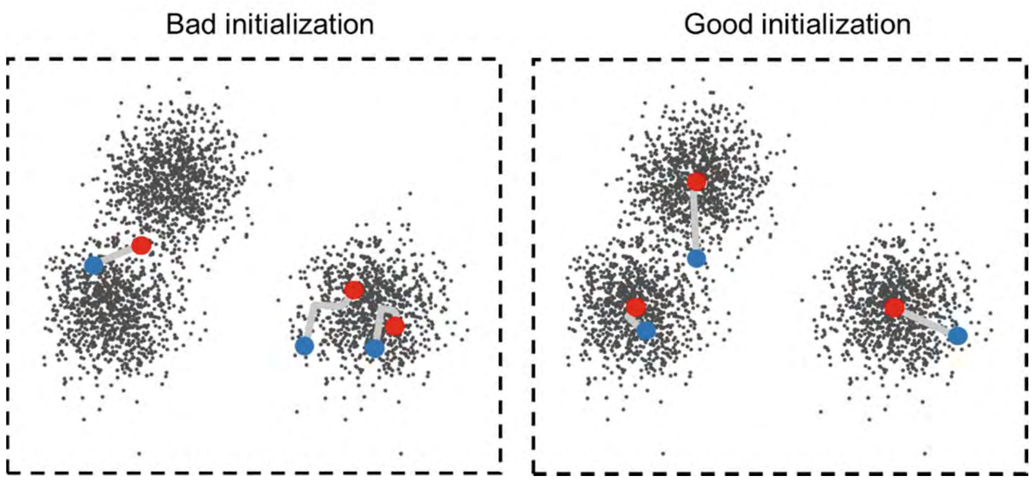


Figure 3. K-means initializes the centroids to random positions (blue dots). The algorithm then iteratively tunes the locations of the centroids until it converges to a local optimum solution (red dots). Success of the algorithm depends on the initial centroids. In the left example, one of the clusters is incorrectly assigned two centroids. In the right example the initial location of the centroids is better and the algorithm converges to the globally optimal solution.

Clustering algorithms may work well for some types of data and fail for others. It is not generally known why and when clustering methods fail. In chapter 5 we study this problem in the case of k-means clustering algorithm. In particular, we study the situations when it fails, investigating properties of the data such as dataset size, number of clusters, unbalance, dimensionality and overlap of clusters.

We also study the most common ways of improving performance of k-means. The simplest way is to just repeat the algorithm multiple times with different random initialization [**P4**]. The other way is to use better algorithms [38,40,47,51] to provide the initialization, which is fine-tuned by k-means.

2 Data and similarity

This thesis has two main emphasis points: clustering methods and *k*-nearest neighbors (kNN). They both belong to a large class of computational problems called *proximity problems*. Indyk defined these as problems whose definitions involve the notion of distance between data points [52]. We define proximity problems as all problems where either distance or similarity is the primary property of the data used in defining the problem. These problems include the *closest pair problem* [52], *minimum spanning tree* [53], *furthest pair* [52], *furthest neighbor* [52] and *travelling salesman problem* [54]. Because the notion of distance is at the heart of these problems, they are all applicable to the same data sets.

Having suitable datasets and understanding the characteristics of those datasets forms the basis of algorithm performance evaluation. In case of clustering algorithms, classification datasets from UCI [55] are often used in benchmarking. Classification and clustering are related because they both divide the data into a certain number of disjoint groups. Still, we refrain from using those datasets because they do not allow systematic control of data properties. Also, classification and clustering are different problems. Clustering is used for many purposes like exploration and data summarization. Consequently, it often reveals different structures from the data than what classification class labels define.

Table 1. Datasets used in this thesis (abbreviation in brackets). In case of text datasets, dimensionality is measured as the number of characters (c).

Dataset	Type	Clusters	Dim.	Size	Ref.
A1-A3	spherical	20,35,50	2	3000-7500	[57]
S1-S4	spherical	15	2	5000	[58]
Dim32-1024	spherical	16	32-1024	1024	[25]
G2	spherical	2	2-1024	2048	[59]
Birch1 (b1)	spherical	100	2	100,000	[60]
Birch2 (b2)	spherical	100	2	100,000	[60]
Unbalance (unb)	spherical	8	2	6500	[61]
RC100k-h (RCh)	spherical	100	128	100,000	[**P2**]
RC100k-l (RCl)	spherical	100	128	100,000	[**P2**]
RC1M (RCm)	spherical	100	128	1 million	[**P2**]
Worms2D (W2)	shape	35	2	105,600	[**P2**]
Worms64D (W64)	shape	25	64	105,00	[**P2**]
Flame (fla)	shape	2	2	240	[2]
Aggregation (agg)	Shape	7	2	788	[62]
Spiral (spi)	shape	3	2	312	[63]
DS6_8 (DS6)	shape	8	2	2000	[19]
Countries	text	48	8.1 c	6000	[**P2**]
English words	text	-	9.4 c	466,544	-
Tweets	text	-	90 c	544,133	[64]

There has been a clear lack of good benchmark datasets. Previously only Steinley created properly controlled datasets [56], but he did not publish the data. Contrary to this, the datasets documented in this thesis are all publicly available[1], with the exception of the tweets dataset and DS6_8.

In this section, we introduce the datasets that have been studied in this thesis and analyze their properties. We have mainly applied clustering of three different types of data: spherical data, shape data and text data. In sections 2.1-2.3, we discuss the different types of data. In section 2.4 we discuss the properties of data that are relevant for clustering. In section 2.5

1 http://cs.uef.fi/sipu/datasets/

we show different ways of calculating distance or similarity. In chapter 2.6 we define the goals of clustering. In chapter 2.7 we discuss how to evaluate the results of clustering.

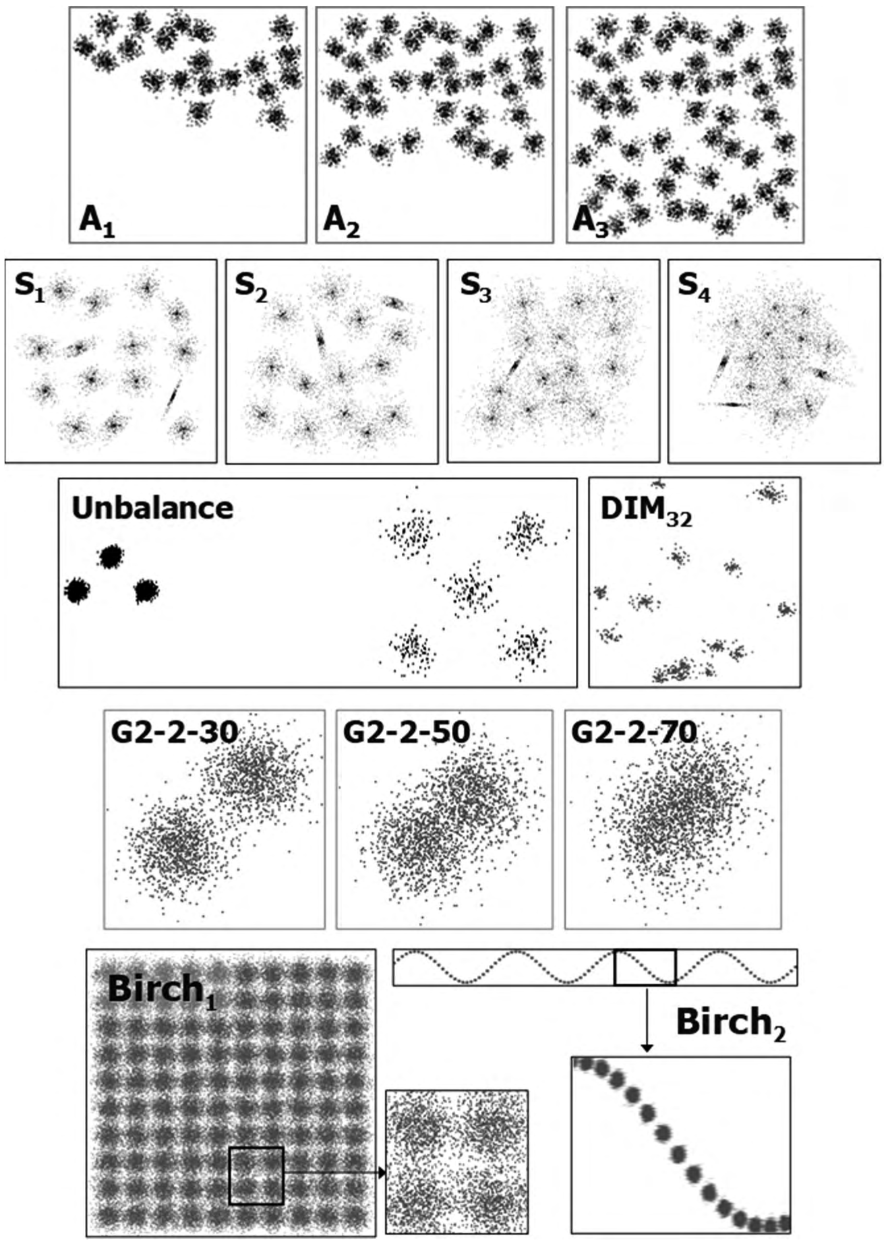


Figure 4. Some of the spherical datasets used in this thesis. **[P3]**

2.1 Spherical data

Most of the spherical datasets were documented in [**P3**] as part of the clustering basic benchmark (see Figure 4). They have been selected so that the SSE objective function can be used for clustering them. They are challenging enough that most simple heuristics will fail, but easy enough that a good clustering algorithm can solve them.

All of the spherical datasets are artificially generated data. Therefore, they have also ground truth clustering, which correctly represents the original parameters used in generating the dataset, i.e. the number of Gaussian distributions and their center points. The ground truth also matches both the SSE optimal clustering (see Chapter 2.6) for the dataset and human intuition. For real world data and applications there is often no single correct clustering.

One way to analyze properties of a dataset is to use histograms of the pairwise distance values. Steinbach et al. [65] used histograms to estimate whether the data have clusters. See Figures 5-6 for examples. In general, clusters with varying densities and distances from each other will produce multiple peaks, and overlap causes the peaks to merge. Histogram is a convenient way to represent a dataset, especially for high dimensional datasets that are difficult to visualize otherwise.

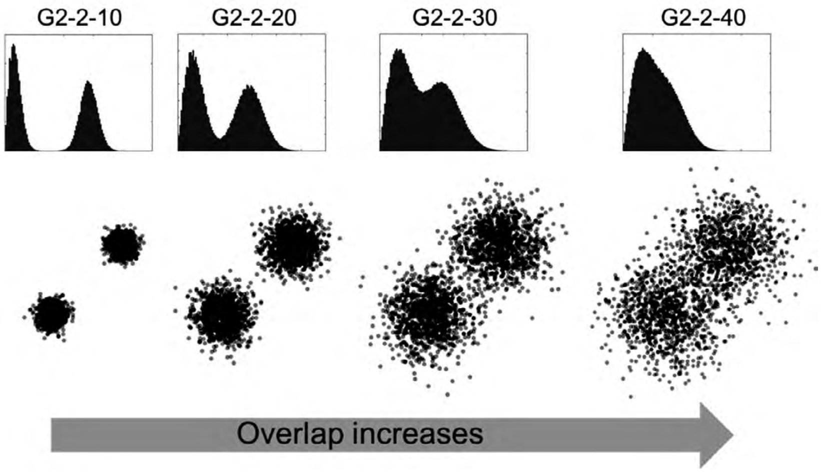


Figure 5. Variants of the G2 dataset and their distance histograms. The first peak is for the distances inside the clusters, the second for distances between clusters. When variance of clusters is increased, they become more overlapped and the distance histograms finally merge into one.

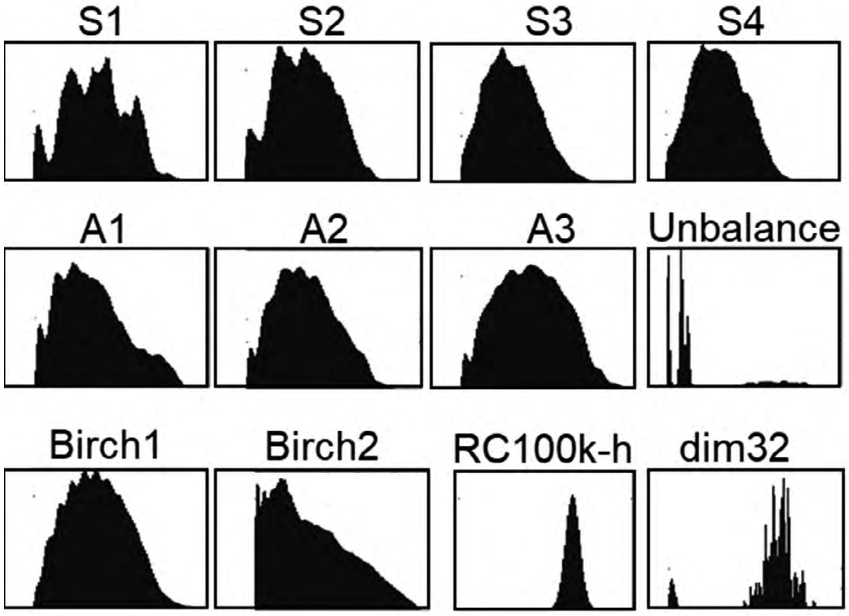


Figure 6. Distance histograms of selected (spherical) datasets. If data has clusters, this usually shows up as peaks in the distance histogram. The first peak contains the smallest distances which are typically inside clusters. Other peaks contain distances between clusters. If clusters have different densities, this may show up as more than one peak of intra-cluster distances, such as in case of the unbalance dataset: local distances in the dense clusters, local distances in the sparse clusters, and the flat area for the global distances.

2.2 Shape data

The shape datasets used in this thesis (Figure 7-8) are also artificial numerical datasets. They do not constrain to spherical clusters but contain any kind of shapes that still appear as distinct clusters to a human observer. The shapes include ellipses, concave shapes and squiggly, worm like, lines.

The Worms2D dataset is one example of a shape dataset. It was produced for article [**P2**]. It consists of 105,600 points in 35 shapes (clusters) which depict trails of random movement in 2D space. The data contains 35 individual shapes that start from a random position and move towards a random direction. At each step, points are drawn from the Gaussian distribution to produce a cloud around the current position. The direction of movement is continually altered to an orthogonal direction and collision detected to prevent completely overlapping clusters. In previous works [19,62,63], artificial shape data has mainly restricted to two dimensional datasets. We also generated a high dimensional (64D) version of the Worms dataset where shapes depict random movement in high dimensional space.

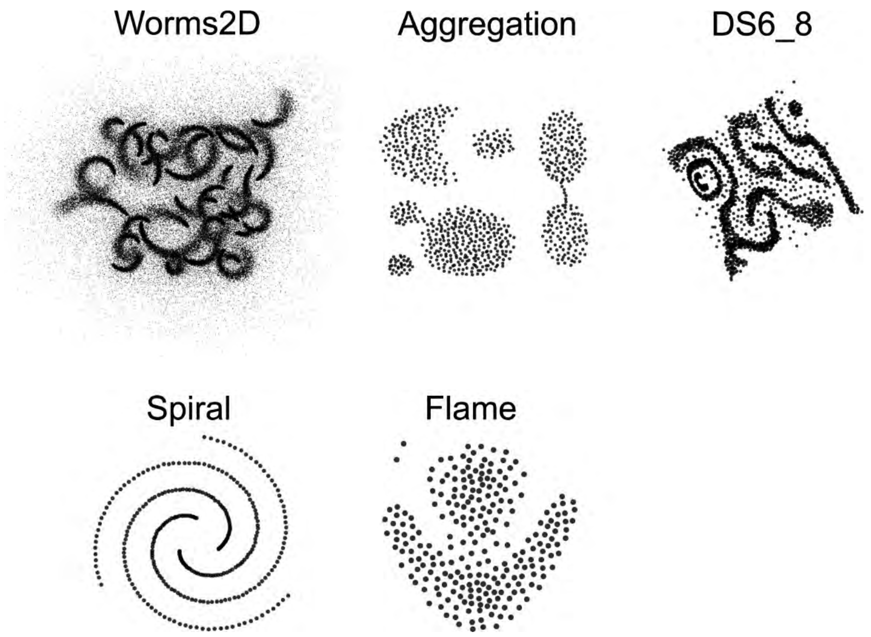


Figure 7. Shape datasets contain non-spherical data. They are not suitable to cluster with k-means, but can be better clustered using density based methods such as Density Peaks or DBSCAN.

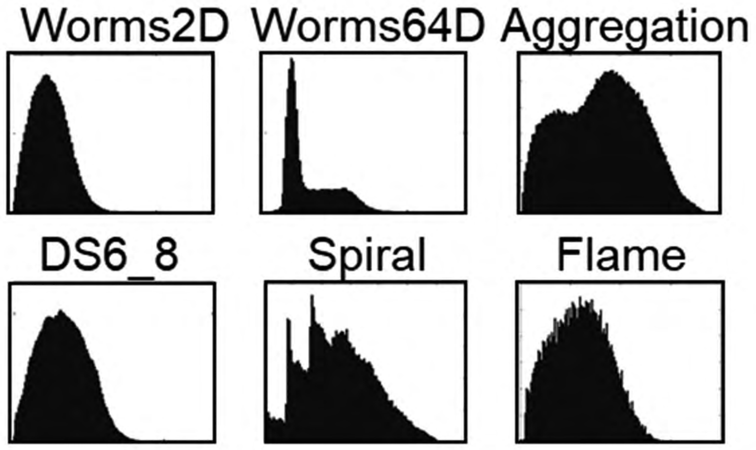


Figure 8. Histograms for shape data do not have as clear first peak as spherical datasets. This is because, contrary to spherical datasets, points in the same cluster can be very far from each other. Points belong to the same cluster when other points, which are near each other, form a “chain” between them. This is highlighted in the case of the spiral dataset.

2.3 Text data

The text datasets (Table 2) contain short text strings. The countries dataset is artificially generated and has a ground truth clustering where country names like Spain, Moldova and Hungary are considered true centroids and other strings are randomly modified versions of these. The other two, English words and Tweets contain real life data and have no specific correct clustering, but can still be clustered in a meaningful way.

Table 2. Ten random samples from each of the text datasets.

Words	**Countries**	**Tweets**
hemilaminectomy	hkujndiyry	Kristersson pratar om bidragstak. Vad är bidrag enligt Moderaterna? Viktigt att säga är att färre nu lever på försö… https://t.co/tpuEGiHIcr
noninterdependently	bipain	TO-MORRO 🌞 🌸 https://t.co/4d2lMoXJsb
overtheorized	ulovezsa	I'm kidding this is how I'm going to sleep tonight not knowing which of the 25 Dinguses America's Dingus Sweetheart… https://t.co/h6Sr7QDhX8
inselberge	osloenia	@tedlieu And who is enforcing the application of international human rights laws in the United States, with the big… https://t.co/629M8v3KnU
tonn	mosldova	Are they hosting with snipers aiming at them? Why are they so uncomfortable? 😊 #esc2018
Cuculiformes	celn	And if #UK doesn't get to sing again I'M SUING! #Eurovision
Hillard	nynthecrands	Haha!! Ni som har barn, har ni sett Djurparken på HBO? På Toonix 😊 Vilken jävla pärla alltså. Kolla in! https://t.co/j7iQFRB3jF
dichromatic	mkonmtnegrv	Er så lættis å se hvordan politimenn tror de er guder. Bedre enn alle andre
domesticized	snorwac	@_chaigal Hahahahahahahahha
attemperate	acedoniax	Wind 0,0 m/s NNW. Barometer 1019,0 hPa, Rising slowly. Temperature 10,9 °C. Rain today 3,6 mm. Humidity 69%

2.4 Properties of data

In papers [**P2**,**P3**,**P4**] we studied how well clustering algorithms work in different circumstances. Specifically, we focused on the following four aspects of data: (1) Number of clusters, (2) dimensionality, (3) overlap and (4) unbalance.

The clustering-problem is generally understood to become more difficult as the *number of clusters* increases. However, at least in case of k-means, the relationship between the number of clusters and the difficulty is not previously known. In [**P3**] we will show that the difficulty for k-means grows linearly with the number of clusters.

The *dimensionality* of data makes many computational problems more difficult [66,67]. This has also been noted in case of k-means [36] and the clustering problem in general [68]. The dimensionality of data is usually understood to be the number of attributes in the data representation (e.g. elements in a vector). However, some data may have low intrinsic dimensionality although the data representation has high dimensionality. In some cases, this can be automatically detected with tools like principal component analysis or based on variance of distances [69].

Intuitively, *overlap* can be understood as an inverse of separation, a measure of how close clusters are to each other or if there is a lack of empty space between them. It is often assumed that clustering algorithms perform better when clusters are more separated (less overlap) [65,22]. We vary the overlap property using the S-sets and G2 sets.

In [**P3**] we introduced a formal definition for overlap (Figure 9). We measure overlap by calculating the distance from every point *x* to its nearest centroid (d_1) and to its nearest point in another cluster (d_2). If any point from another cluster is closer to *x* than its own centroid ($d_1 > d_2$), then the point is considered an evidence of overlap. Overlap of the dataset is then defined as the number of evidences relative to the total number of points:

$$\text{Overlap} = \frac{1}{N} \cdot \sum ov(d_1, d_2) \tag{1}$$

$$\text{where } ov(d_1, d_2) = \begin{cases} 1, & d_1 > d_2 \\ 0, & d_1 \leq d_2 \end{cases}$$

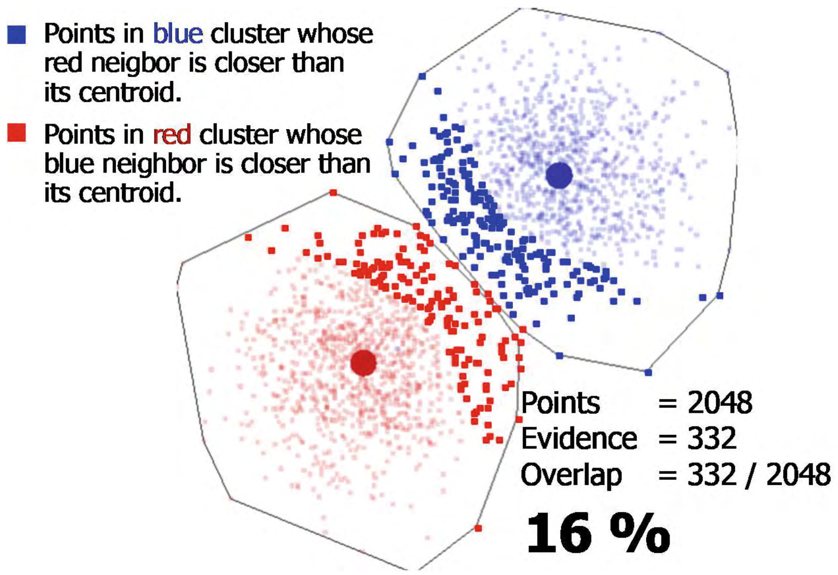


Figure 9: Overlap measured for the G2-2-30 dataset. [**P3**]

The unbalance of cluster sizes means that some clusters in the dataset have much more points than others. Most datasets studied in this thesis contain clusters of roughly same size. The exception is the Unbalance dataset (Figure 4) which has 20:1 size ratio between the smallest and the largest cluster. Since the clusters with fewer points have also larger variance, the dataset has strong unbalance also in density.

Using artificial datasets allows to systematically control these properties of the data. The number of clusters can be varied using the A1-A3 datasets and subsets of the birch2. Overlap increases steadily in the variants of the S-sets (S1-S4); also the RC100k has high and low overlap versions. The Dim-datasets contain clearly separated clusters with different dimensionality ($32 < D < 1024$). The G2 sets vary both dimensionality and overlap, but only in the special case of two clusters.

2.5 Similarity or distance

Many different distance measures have been developed, but the Euclidean distance (Equation 1) is still the most widely used. It is defined for vector data of $1 \leq D < \infty$ dimensions:

$$\boldsymbol{L_2(x, y)} = \left(\sum_{i=1}^{D} |\boldsymbol{x_i} - \boldsymbol{y_i}|^2\right)^{\frac{1}{2}} \tag{2}$$

A more general distance measure is the Minkowski distance, which is almost identical, but takes the additional parameter *p*:

$$\boldsymbol{L_p(x, y)} = \left(\sum_{i=1}^{D} |\boldsymbol{x_i} - \boldsymbol{y_i}|^p\right)^{\frac{1}{p}} \tag{3}$$

Here Euclidean distance corresponds to Minkowski distance with $p = 2$. Other forms of the Minkowski distance are Manhattan distance with $p = 1$ and Chebyshev distance with $p \rightarrow \infty$. Use of values $0 < p < 1$ are less common, but have been shown to work better for high dimensional data [66].

Algorithms are often designed to work with only specific distance measures. For example, the k-means algorithm can work with any forms of the Minkowski distance, but the SSE cost function is properly optimized only using the Euclidean distance. Some other methods such as Density Peaks [5] can take a full distance matrix as input and do not even need access to the original data or distance measure.

In this thesis, we use text datasets in addition to numerical ones to test how algorithms work when the type of data is very different. A large number of different text similarity measures have been introduced; we point to [70] for a good review. Most commonly used measure for text data is edit distance [71]. It calculates the minimum number of edit operations needed to transform a string *x* to string *y*. The edit operations include insertion, deletion, and substitution.

However, edit distance has time complexity of O(n^2) for strings of length *n* and is therefore slow for larger strings. Set matching based measures like the Dice coefficient [72] can work faster, in linear time (assuming set representation is precalculated). In Dice coefficient, each string is represented as a set of bigrams. For example, word *string* would become {*st*, *tr*, *ri*, *in*, *ng*}. Similarity is measured as the size of the intersection divided by the average cardinality of the two sets:

$$d_{Dice}(x,y) = \frac{2 \cdot |x \cap y|}{|x| + |y|} \tag{4}$$

2.6 Cost functions

Clustering algorithms can use a cost function to determine the goodness of clustering. These include *the mean squared error* (MSE) and *mean absolute error* (MAE). The mean squared error is the most popular. Another variant of it is the *sum of squared errors* (SSE), which is almost equal but lacks the scaling via division with data size. Given a dataset X = {$x_1, x_2 \ldots, x_N$} and a list of cluster centroids C = {$c_1, c_2 \ldots, c_k$}, where c_j is the nearest centroid to x_i, the MSE is defined as:

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} \left\| x_i - c_j \right\|^2 \tag{5}$$

Mean absolute error is similar, but without the squaring. It has been used especially for the k-medoids algorithm [73]. It is defined as:

$$\text{MAE} = \frac{1}{N} \sum_{i=1}^{N} \left\| x_i - c_j \right\| \tag{6}$$

In the above two cases the only difference is in how distance between data point x_i and nearest centroid c_j is calculated. Euclidean distance (L_2) is used in case of MSE and L_1 in case of MAE. But often these can be substituted with some other distance function. For example, the MATLAB implementation of k-means allows to choose from the following distance functions: L_2, L_1, cosine, correlation and hamming.

2.7 Measuring clustering quality

After a dataset has been clustered, there remains the question of how good is the clustering? For real usage scenarios, this question cannot usually be answered since the correct clustering is generally unknown. Still, in case of artificial data with known ground truth, it is possible to compare the result of clustering algorithms to the ground truth in order to test how well the algorithms perform.

Many ways exist to measure clustering quality. The *Normalized Mutual Index* (NMI) and *Adjusted Rand Index* (ARI) are two of the most popular ones. For a good review, we point to [61]. However, these commonly used quality measures have the problem that the values they produce (such as 0.79 or 0.54) don't have a clear meaning and are difficult to interpret.

For this reason, we use the *Centroid Index* (CI) instead [74,75] as our primary measure of success. The CI values provide a clear understanding of how many real clusters have errors. That is, how many clusters are missing a centroid (see Figure 10).

Given a ground truth solution (*G*) and a clustering solution (*C*), Centroid Index counts how many real clusters are missing a center. This calculation is done by performing a nearest neighbor mapping between the clusters in C and G. The nearest neighbor mapping is done in both directions, *C→G* and *G→C*. The clusters that aren't the nearest neighbor of any cluster in the other solution are considered orphans. The number of orphans is counted for each mapping and the maximum number is taken as the CI-value. This provides a much clearer intuition about the result. Specifically, if CI=0, we conclude that the result is correct clustering. We can say then that the algorithm solves the problem.

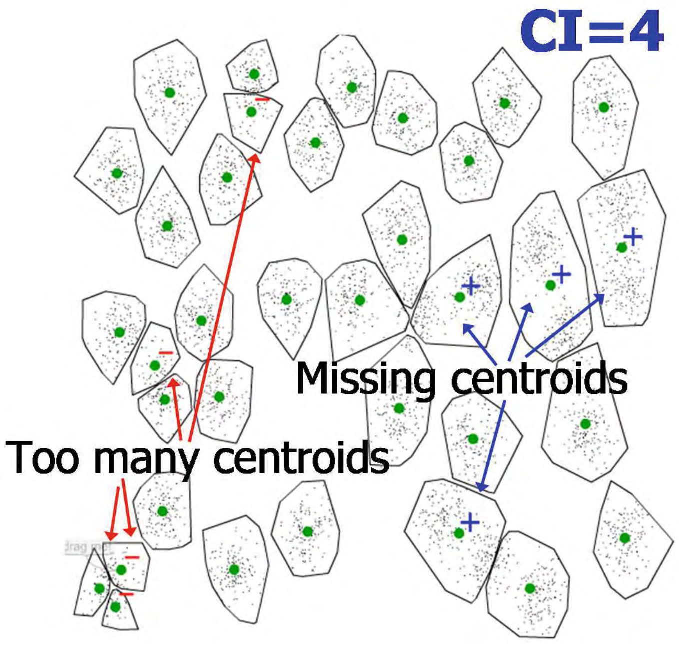


Figure 10: Example of a typical k-means result for the A2 dataset. The corresponding measures for this are: CI=4, SSE=3.08. [**P3**]

3 kNN graph

Given a set of *N* points *X* = {*x*1, *x*2...,*xN*} in some *D*-dimensional space *S*, the k-nearest neighbor problem (*k*NN) is to find the *k* points in X that are closest to a given query point *q* ∈ *S* according to some distance function *d*. A search for the *k* nearest neighbors for all points in *X* yields a directed graph called *k*NN-graph (Figure 11) where the vertices correspond to points in the data set and edges connect each point to its *k* nearest points in the data set.

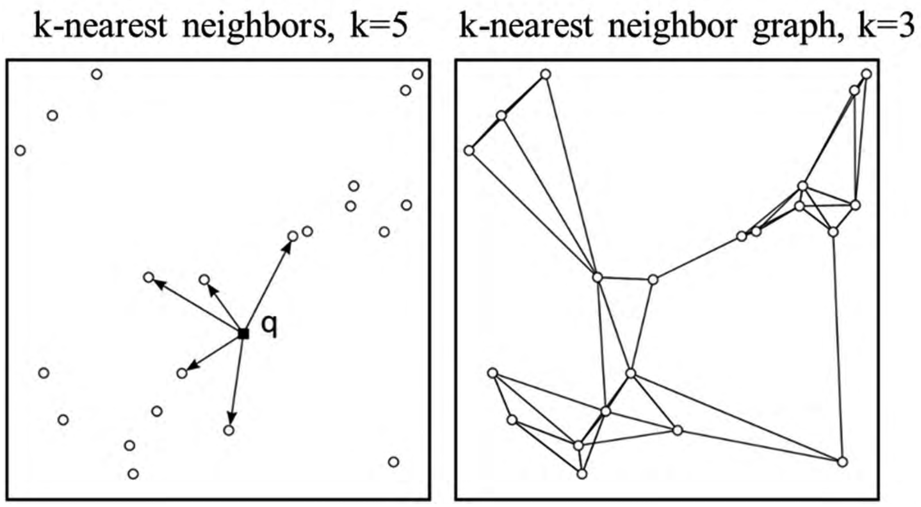


Figure 11. The kNN graph is formed by finding the k-nearest neighbors for all points in the dataset. Therefore any kNN search method can also be used to construct a kNN graph. However, the reverse is not possible since in the kNN search problem the query point (q) can be any unknown point outside the original dataset.

In the example in Figure 11, the graph is created for an artificial dataset consisting of points on a 2D plane. However, it can also be created for any type of data as long as there is some way to calculate distance or similarity between the data objects. Consequently, it has been used for many different types of data, including text [76], images [77] and music [78]. An example of a kNN graph for small words is shown in Figure 12.

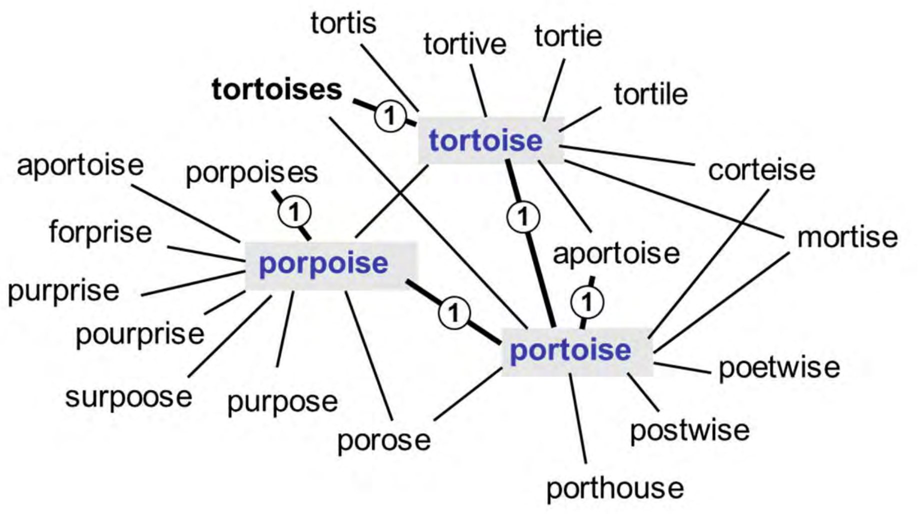


Figure 12. Example of *k*=10 nearest neighbors for the words porpoise, tortoise and portoise. This is part of a larger edit distance *k*NN graph on 466,544 words dataset. Here only the neighbors of the three words are shown. All distances in the graph are 2, except those marked by number 1. [33]

3.1 Exact methods and the problem of dimensionality

Exact kNN graph can be calculated fast for datasets that are either small in size or low dimensional. For large and high dimensional datasets there exists no efficient exact methods. In these cases approximate methods are needed.

For small datasets, the brute-force algorithm can be used to construct a kNN-graph in O(N^2) time. It works simply by calculating distances between all pairs of points and selecting the *k* smallest distances for every point.

Faster methods exist for low dimensional data. For example, kd-trees [79] or z-order curve [32] can be used to calculate a kNN graph in O(*n* log *n*) time. However, all these methods fail for high dimensional data. To understand why this happens, consider the following example of kd-trees.

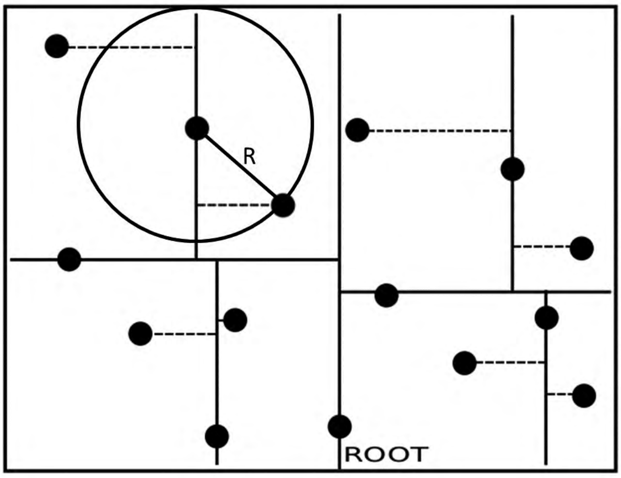
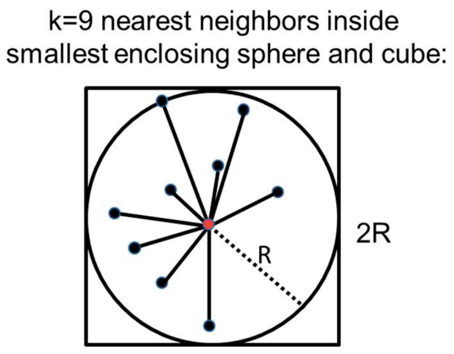


Figure 13. kNN searching in kd-tree.

Kd-trees are constructed by recursively dividing the data space on the median point of a selected dimension, altering the dimension on each division (Figure 13). This results in a tree which consists of nested hyperrectangles. Searches in this tree involve finding a candidate set of kNN points and then checking all points inside the smallest hyperrectangle that encloses the ball of kNN candidates. This works well in 2D cases because the rectangle is usually not much larger than the circle.

However, as dimensionality increases, the ratio of sphere volume to cube volume goes rapidly towards zero (see Figure 14). Already with 10 dimensions the volume of the hyperspehere is only 0.25% of the volume of a same width hypercube. If there are *k*=9 points in the candidate kNN ball, then it is expected that 9/0.25%=4000 points would be inside a similar width hypercube.


k=9 nearest neighbors inside
smallest enclosing sphere and cube:
R
2R

Volume of hypersphere:

$$V = R^D \frac{\pi^{(D/2)}}{\Gamma(D/2+1)}$$

D	Volume(Sphere)/ Volume(Cube)	Points inside hypersphere	Points expected inside hypercube
2	79%	10	13
3	52%	10	19
5	16%	10	62
10	0.25%	10	5000
100	1.9e-66 %	10	5.3e+68

Figure 14. Volume of a hypersphere in relation to volume of a same width hypercube goes to zero very quickly as dimensionality increases. This comes mainly from the O(D!) gamma function which is the denominator in the formula. Number of points are expected to be in relation to volume of container. Therefore, assuming uniform distribution and fixed number of points inside a hypersphere, the number of points inside a hypercube is expected to grow exponentially as dimensionality increases.

In addition to kd-trees, many other exact search methods, such as z-order search [32], mean order partial distance search [80] and principal axis trees [80] work using a similar approach of searching the contents of a hyperrectangle containing the kNN ball. The main difference between these methods is in how the hyperrectangles are constructed. Therefore, all of them generalize poorly for high dimensional data. For this reason, approximation methods are needed in case of high dimensional data.

3.2 Neighborhood Propagation

Neighborhood propagation is one method to construct an approximate kNN-graph. It works by repeatedly measuring distance from each point of a graph to all of its neighbors' neighbors and keeping the ones with *k* smallest distance. It is based on the observation that if a point *y* is a neighbor of *x* and point *z* is a neighbor of *y*, then *z* is also likely to be a neighbor of *x*. Different variants of neighborhood propagation have been used in many approximate kNN graph construction methods [**P1**,**P2,**29,78,81-83**]** to refine the quality of the graph after constructing an initial coarse approximation using some other method.

Nearest Neighbor Descent (NNDES) developed by Dong et. al. [84] is one variant of neighborhood propagation which also works as a standalone method. It starts from a random kNN graph and gradually builds an approximate kNN graph by refining it with neighborhood propagation. Compared with other methods like those using principal component analysis (PCA) [77,81], it has the benefit that it doesn't require the data to be in numerical form. Since the process only requires a distance or similarity measure, it can work with almost any type of data.

The algorithm iteratively improves the quality of the graph. In each iteration, the neighbors of neighbors are tested for each point $x \in X$. If any of them are closer than the furthest of the current neighbors, the neighbor list is updated accordingly. The algorithm is iterated until a specified stop condition is met. For example, it can be run just for a fixed number of iterations or as long as the method is able to improve the graph.

Since each point has k^2 neighbors of neighbors, the propagation requires $O(k^2N)$ distance calculations per iteration. The total time complexity is therefore $O(k^2NI)$ where I represents the number of iterations and is a small number, usually roughly 20.

This time complexity makes the method very slow for kNN graphs with a large number of neighbors (e.g. k=100). For this reason, in [**P1**] we run the neighborhood propagation only for m nearest neighbors where $m < k$. We used the rule $m = \sqrt{jk}$, where j is a small number. We used the value j=10. Therefore, in case of a graph with k=10 neighbors, NNDES would be run for the *m*=√*jk*=√(10*10)=10 nearest neighbors. And in case of graph with *k*=100 neighbors, NNDES would be run for the *m*=√*jk*=√(10*100)=32 nearest neighbors. This way, the time complexity per iteration can be kept linear for *k*, at O(m^2N)=O(*kj*N). Although the NNDES search gets run for the *m* nearest neighbors, it is the values of the whole *k* nearest neighbors that gets updated.

3.3 Z-order neighborhood propagation

In this section we summarize the *Z-order neighborhood propagation* (ZNP) method from [**P1**], which constructs an approximate kNN-graph for high-dimensional data. It uses one-dimensional mapping with a Z-order curve to construct an initial graph and then continues to improve this using the NNDES algorithm.

$$\begin{array}{rl} & \text{interleaved bits} \downarrow \\ (3,5) = (011_2, 101_2) & \rightarrow 01\ 10\ 11_2 = 27 \\ & \rightarrow 10\ 01\ 11_2 = 39 \\ \uparrow \text{2D vector} & \qquad\qquad \text{z-value} \uparrow \end{array}$$

Figure 15. Bit-interleaving is used to produce the Z-values.

The Z-order curve (Figures 15-16) is a function which maps multidimensional points to one dimension by interlacing the bits of the binary representation of the vector components. This one-dimensional value is referred to as Z-value. When multidimensional points are ordered by their Z-values, this order is called the Z-order. The Z-order curve has been independently invented by Morton [85], Orenstein [86] and Tropf and Herzog [87]. The Z-order curve has been previously used to construct a kNN graph, but only for low-dimensional data [32].

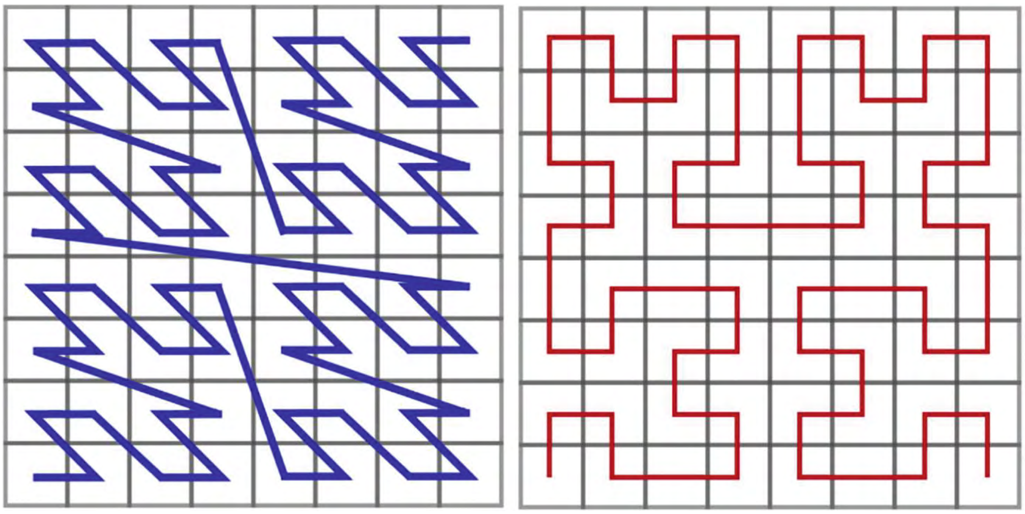


Figure 16. Space filling curves impose an ordering for multidimensional discrete space. The Z-order -curve (left) and Hilbert curve (right) are the two most common space filling curves. Both are self-similar, which means that the same pattern repeats recursively in three different levels on the 8x8 grid. [**P1**]

A kNN graph can be constructed using the z-ordering of points (see Figure 17) by processing the points using a sliding window along the z-order. The exact distance is then calculated between all points inside the sliding window. This produces a low quality approximation of a kNN graph. This approximation can be improved by repeating the process multiple times for a randomly transformed data set. A simple transformation is to shift the pointset to a random direction.

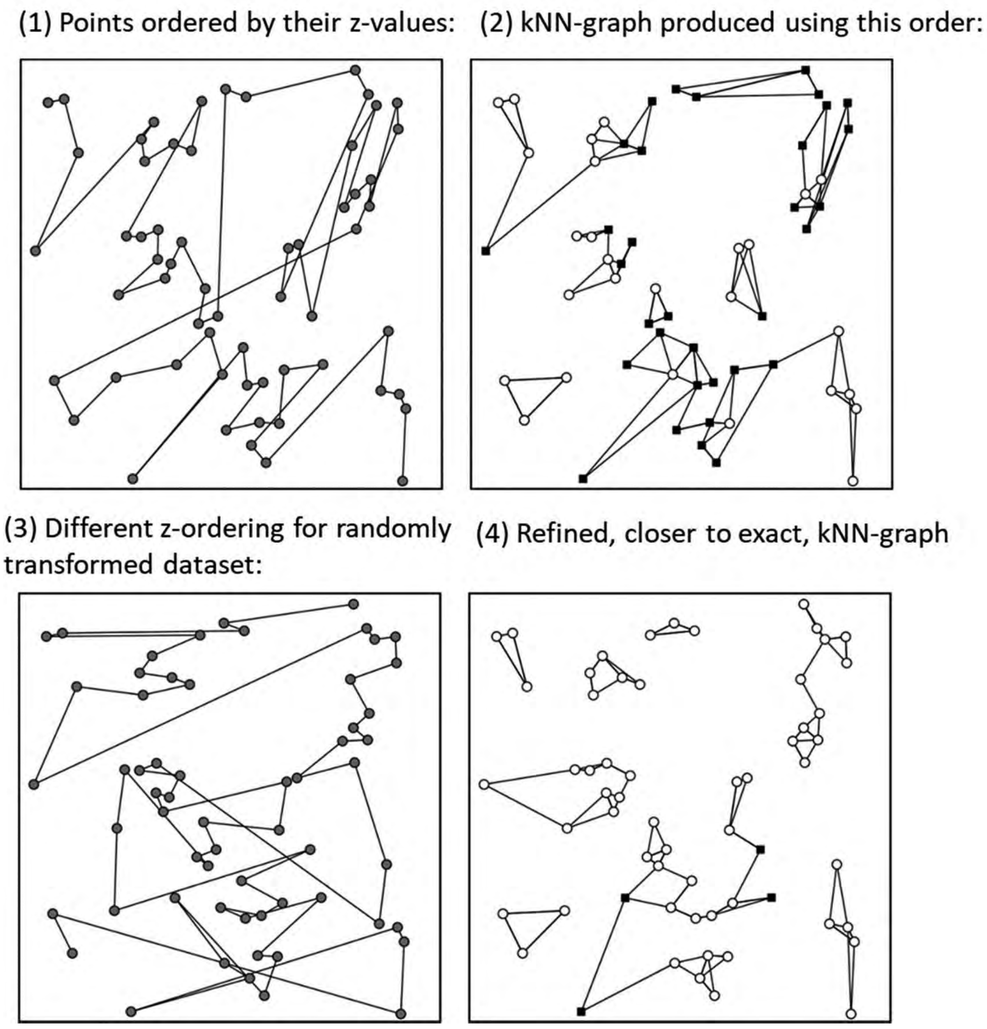

(1) Points ordered by their z-values:
(2) kNN-graph produced using this order:
(3) Different z-ordering for randomly transformed dataset:
(4) Refined, closer to exact, kNN-graph

Figure 17. Multiple different z-orderings are needed to produce a high quality kNN graph. Error points are shown as black rectangles. [**P1**]

The Z-order curve has been previously used to construct a kNN graph, but only for low-dimensional data [32]. Applying it for a higher number of dimensions can be problematic. One of the problems is that the z-values become very large with a high number of dimensions. For example, for a data set with dimensionality D = 1000, and bit-length b = 32 bits per dimension, the Z-values would need to be represented by D · b = 32000 bit integers. Calculating, storing and comparing such large integers would become a bottleneck in the algorithm.

To speed up the z-value handling in case of high dimensional data, we introduced a simple dimensionality reduction method. It works by dividing the dimensions into random subsets with roughly equal sizes and then constructing new subvectors corresponding to the subsets of the dimensions. Each subvector is mapped to one dimension by projecting them to the diagonal axis.

This process is related to *Johnson-Lindenstraus transform* (JLT) [88], *Neighborhood embedding* [89] and *Multidimensional Scaling* [90]. The main difference with JLT and Multidimensional Scaling is that they aim to preserve the distances between points in the projected space, whereas we are concerned only on preserving the neighbor connections. Neighborhood embedding on the other hand, aims at dimensionality reduction while preserving the neighbor connections, but it is used mainly for visualization purposes.

3.4 Random point division

ZNP, the Z-order method, was fast but restricted to vectorial data and only tested with Minkowski distance measures. In this section, we present another algorithm called *Random Point Division* (RP-Div) which doesn't have this limitation and can work for any type of data as long as a distance or similarity measure is provided. It has been documented in article [**P2**] and somewhat more extensively in [33]. To demonstrate its generality, we have used it with short strings (~10 chars) and Levenshtein distance or longer strings (~140 chars) and dice distance with bigrams.

The RP-Div algorithm constructs the graph by applying hierarchical subdivision (demonstrated in Figures 18-19). The dividing works by first selecting two random points: *a* and *b*. 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 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, which calculates distances between all possible pairs and updates the list of *k* nearest points found.

One iteration of the algorithm produces a crude approximation of the kNN graph. The graph is improved by repeating the hierarchical subdivision several times (Figure 19) 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 random pair division provides a moderate quality approximation (50-80% accuracy) of a kNN graph fast. But if a higher quality graph is wanted, we use the NNDES algorithm [84] to refine the graph further.

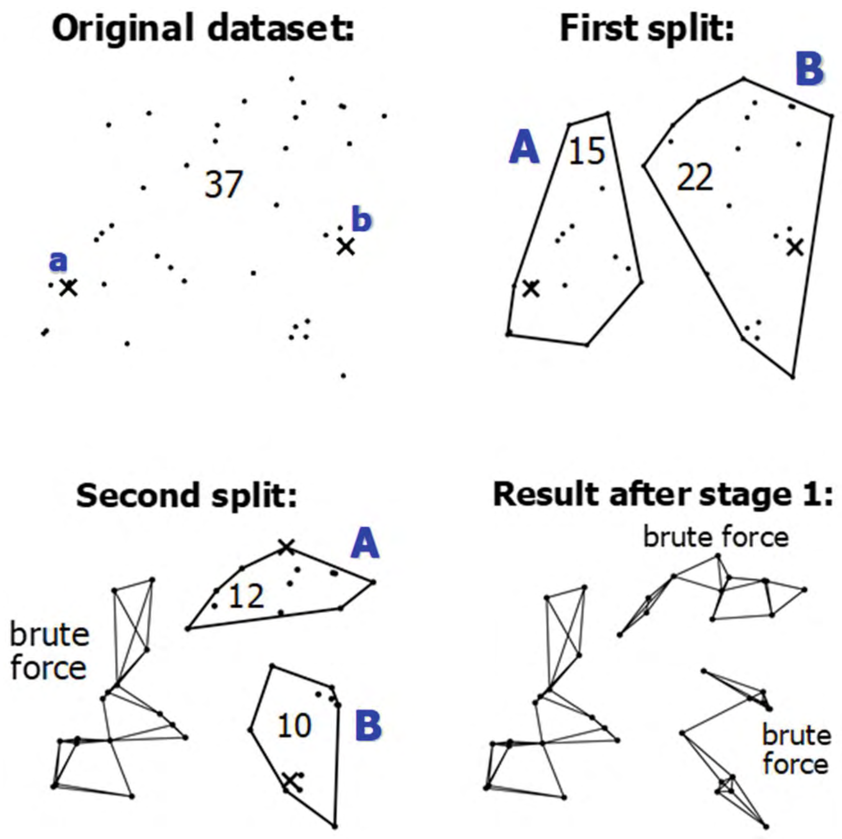


Figure 18. The RP-div algorithm recursively subdivides the dataset of size *N*=37 by first choosing two random points (a,b). The dataset is split based on which of the two points is nearer. After the first split, the size of the subset A is smaller than threshold *W*=20, and is solved by brute force. The subset B is further divided into two more subsets, which both are smaller than *W* and now solved by brute force. **[P2]**

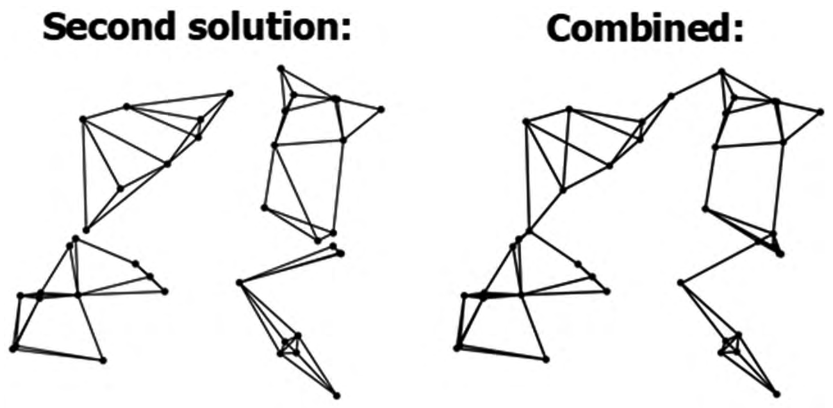


Figure 19. After repeating the random pair division, a new solution is obtained. This solution is merged with the previous one to form a new improved kNN graph. [**P2**]

4 Density Peaks clustering using a kNN graph

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 [**P2**], we propose a fast density peaks algorithm that solves the time complexity problem.

The proposed algorithm speeds up the density and delta calculation of density peaks by using a kNN graph. The graph is constructed using the random point division method presented in chapter 3.4. 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.

4.1 Density Peaks

The Density Peaks clustering algorithm [5] 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 Density Peaks method has four main steps:

1. Calculate the density values
2. Find big brothers
3. Identify cluster centers
4. Join the remaining points

The original Density Peaks article [5] did not specify exactly how clusters should be selected based on ρ and δ (step three). Different approaches have been considered. For example, it could be done manually by the analyst using a density vs. delta plot (Figure 20). In [**P2**] we follow the gamma strategy [91,92], which uses the product of the two features ($\gamma = \rho\delta$). Cluster centroids are then selected as the *k* points with the highest γ value (Figure 20). After the cluster centroids (density peaks) have been determined, the big brother pointers are used to merge the remaining points to the same cluster as their big brother (step four).

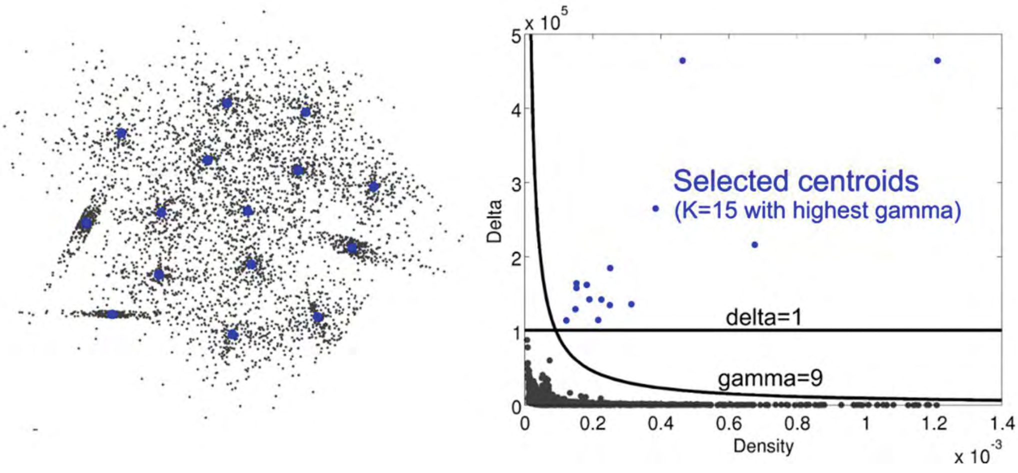


Figure 20. Different cluster selection strategies based on the density-vs-delta plot for the S4 dataset. Cluster centroids typically have both high density and high distance to a higher density point (delta). Therefore, thresholding based on a combination of delta and density (gamma) is expected to work better than using the delta values alone. [**P2**]

4.2 Fast Density Peaks using a kNN graph

The Density Peaks algorithm has two computational bottlenecks:

- Calculating the densities
- Finding the big brother points.

In the original Density Peaks, these steps required comparing all possible pairs of points and thus took O(N^2) time. However, both of these steps can be calculated fast by using a kNN graph (Figure 21). Still, kNN graph construction itself also takes O(N^2) by using a straightforward brute force method.

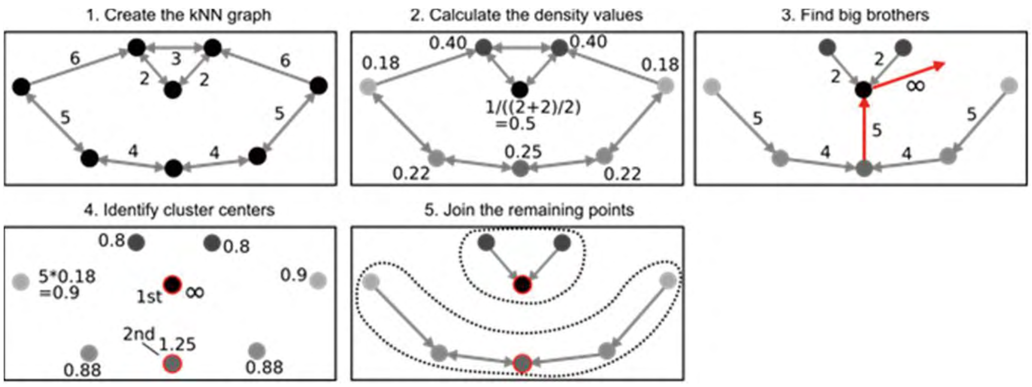


Figure 21. Illustration of the Fast Density Peaks algorithm. (1) For a given data set, the kNN graph is constructed. (2) Densities are calculated as inverse of the mean distance to the neighbors. (3) Nearest higher density point (big brother) is (in case of gray lines) found in the kNN graph. For others (red lines) a slower full search is performed. (4) Cluster centers are identified as the two points that have highest gamma (delta*dens) value. (5) Clusters are formed by joining other points to the same cluster as with their big brother. [**P2**]

Faster methods for kNN graph construction exist. We use the random point division method presented in chapter 3.4 because it can work for any type of data. It is also faster than NNDES [84] which is the only other generic kNN graph construction method [33].

The graph can be used to calculate all the information needed by Density Peaks. Density values can be calculated trivially by taking the inverse of mean distance of the *k* nearest neighbors. The graph can also be used to find most of the big brother points.

Finding the big brother 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, the majority of points have their big brothers located within their kNN neighborhood. We call them 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.

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. These are also the points among which the final centroids will be chosen.

The time complexity of the big brother search 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 $pN^2 + (1\text{-}p)kN = O(pN^2)$. The speed-up is therefore inversely proportional to *p*.

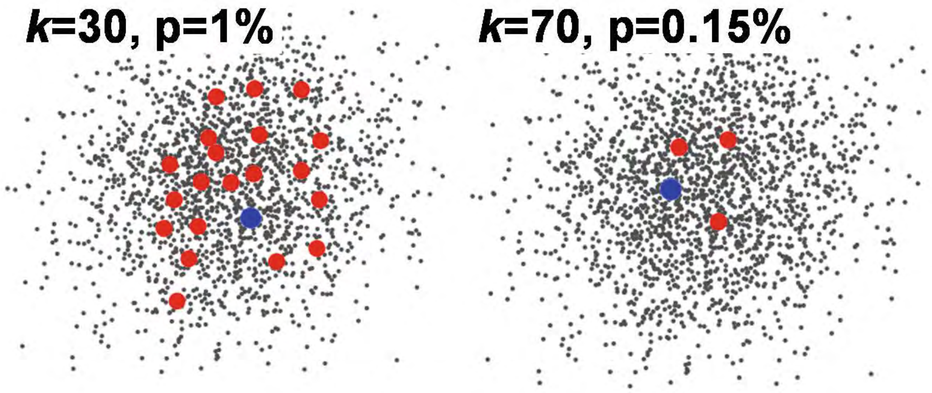


Figure 22. Distribution of slope points (gray) and local peaks (red) inside an example cluster. One of the local peaks (blue) is the resulting cluster centroid (global peak). The case of *k*=30 (left) and *k*=70 (right) are shown. When the number of neighbors *k* in the kNN graph is increased, the number of local peaks decrease. **[P2]**

Figure 22 shows an example of the distribution of the local peak points with an example dataset. In general, the higher the value of *k*, the less there are peak points, and the faster is 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:

$$\begin{aligned} kpN + pN &\le N \\ kp + p &\le 1 \\ p &\le 1/(\mathrm{k}+1) \\ p &\le 1/k \end{aligned} \qquad (7)$$

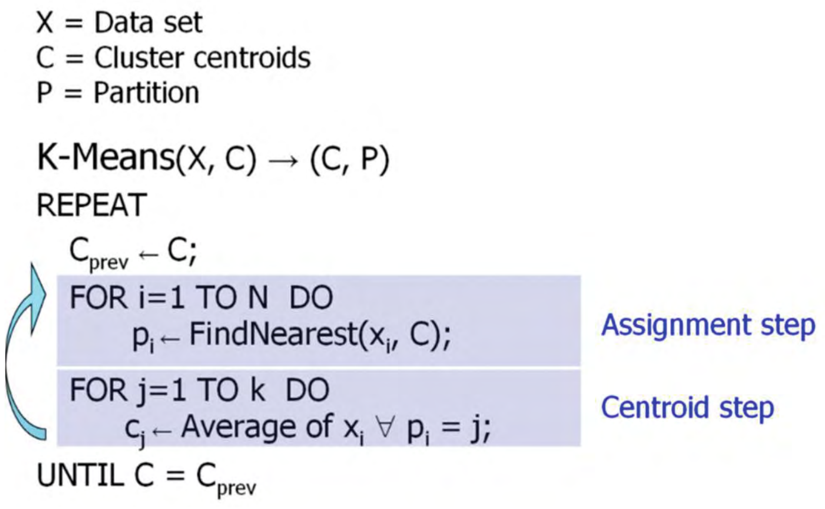
Therefore, the time complexity of O(pN^2) can also be written in terms of *k* as O(N^2/k). When combining with the O(*rkN*) time complexity of the kNN graph construction (see Section 3.4), this leads to a total time complexity of O($N^2/k + rkN$), where *r* is a small number.

5 k-means

The k-means algorithm [11,12,93] groups *N* data points into *k* clusters by minimizing the sum of squared distances between every point and its nearest cluster center (*centroid*). This objective function is called *sum-of-squared errors* (SSE). For a dataset X = {$x_1, x_2 \ldots, x_N$} and a list of cluster centroids C = {$c_1, c_2 \ldots, c_k$}, it is defined as:

$$\text{SSE} = \sum_{i=1}^{N} \left\| x_i - c_j \right\|^2 \tag{8}$$

where x_i is the data point and c_j is its nearest centroid.



```
X = Data set
C = Cluster centroids
P = Partition

K-Means(X, C) → (C, P)
REPEAT
    C_prev ← C;
    FOR i=1 TO N  DO
        p_i ← FindNearest(x_i, C);
    FOR j=1 TO k  DO
        c_j ← Average of x_i ∀ p_i = j;
UNTIL C = C_prev
```

Assignment step
Centroid step

Figure 23. The k-means algorithm.

K-means optimizes this objective by first selecting *k* random data points as the initial centroids and then iteratively fine-tuning those locations (see Figure 3). Each iteration consists of two steps, the *assignment* step and the *update* step. In the assignment step, every point is put into the cluster whose centroid is closest. In the update step, the centroids are re-calculated by taking the mean of all data points assigned to the same cluster. The iterations continue a fixed number of times or until no further improvement is obtained (*convergence*).

The success of k-means depends on the goodness of the initial centroids (see Figure 3). Selecting random centroids can sometimes provide good enough initial centroids so that k-means can fine tune them to a correct solution. For simple datasets, it is enough to repeat the algorithm several times to find the correct solution (Chapter 5.1). For more challenging datasets this approach is not enough. For these situations, many have suggested new techniques to provide better initial centroids (Chapter 5.2).

In Chapter 5.3 we discuss how the different approaches of applying k-means (normal, repeated, better initialization) perform depending on the properties of the datasets.

5.1 Repeated k-means (RKM)

Repeated k-means performs k-means multiple times starting with different initialization, and then keeping the result with lowest SSE-value. This is sometimes referred to as multi-start k-means. The basic idea of the repeats is to increase the probability of success. Repeated k-means can be formulated as a probabilistic algorithm as follows. If we know that k-means with a certain initialization technique will succeed with a probability of p, the expected number of repeats (*R*) to find the correct clustering would be:

$$R = 1/p$$

In other words, it is enough that k-means succeeds even sometimes (p>0). It is then merely a question of how many repeats are needed. Only if $p \approx 0$ the number of repeats can be unrealistically high. For example, standard k-means with random centroids succeeds 6-26% of the time with the S1-S4 datasets. These correspond to R=7 to 14 repeats, on average.

If the initialization technique is deterministic (no randomness), then it either succeeds (*p*=100%) or fails (p=0%) every time. To justify the repeats, a basic requirement is that there is some randomness in the initialization so that the different runs produce different results.

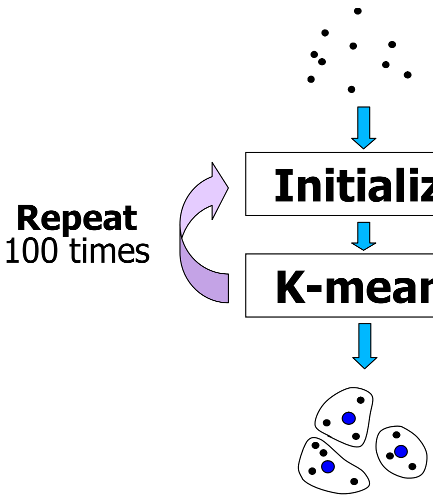


Figure 24: General principle of repeated k-means (RKM). The key idea is that the initialization includes randomness to produce different solutions at every repeat.

5.2 Initialization methods

Any clustering algorithm could be used as an initialization technique for k-means. However, solving the location of initial centroids is not significantly easier than the original clustering problem itself. Therefore, for an algorithm to be considered as initialization technique for k-means, in contrast to being a standalone algorithm, it should fulfil three requirements [**P4**]:

- Be simple to implement
- Have lower (or equal) time complexity than k-means
- Be free of additional parameters

Random centroids [11,12] is the most common initialization technique. It simply selects *k* random data objects as the set of initial centroids. This can be done fast, in O(*N*) steps. It is well suitable to repeating k-means since different random centroids lead to very different end results. It also guarantees that every cluster includes at least one point.

Another very trivial method is random partitions [93]. Every point is put into a randomly chosen cluster and their centroids are then calculated. The positive effect is that it avoids selecting outliers from the border areas. The negative effect is that the resulting centroids are concentrated around the mean of the dataset. This degrades performance of k-means when the data is well separated and centroids cannot easily move between clusters. It also reduces the benefits from repeating k-means because there is very little variation in the initial solutions, and therefore, also the final solutions often become identical.

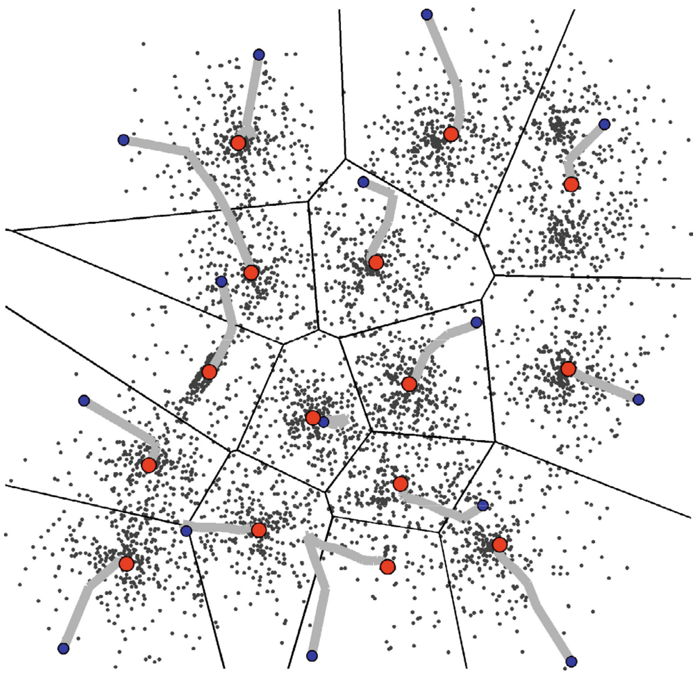


Figure 25. Example of the maxmin heuristic for S3 dataset. The blue dots are the initial and the red dots the final centroids. The trajectories show their movement during the k-means iterations.

Another popular technique is Maxmin [40], also known as the furthest point heuristic (see Figure 25). It selects an arbitrary point as the first centroid and then adds new centroids one by one. At each step, the next centroid is the point that is furthest (max) from its nearest (min) existing centroid. Each of the *k* steps processes the whole dataset, resulting in O(*kN*) time complexity. There are different heuristics to choose the first centroid. For example, in [94], the point furthest from the origin is chosen as the first centroid. However, this type of methods make the algorithm deterministic and therefore unsuitable for repeated k-means. In our implementation in [**P4**], we select the first centroid randomly to make the method work better with repeated k-means.

Maxmin has many variants [38,40,94,95]. The most well known is k-means++ [38] which selects the next centroid probabilistically with weights based on minimum distance to selected centroids. It is more randomized and therefore better suitable to use with repeated k-means.

Sorting heuristics sort the data points according to some criteria like distance to center point [46], density [41], centrality [43] or value for the attribute of greatest variance [42]. Centroids can be selected from the sorted list by taking the first k elements, or dividing the list evenly to *k* buckets and taking one point from each bucket. These strategies are very deterministic and therefore not well suited to use with repeated k-means.

Projection-based heuristics sort points similarly as sorting heuristics, but they use projection as the basis for sorting. Projection can be done using e.g. principal axis [48] or by taking two random points and using the line defined by them [49].

5.3 Results

In papers [**P3**-**P4**], we have run benchmarks to study clustering performance in varying circumstances for the different k-means variants: normal k-means, advanced initialization techniques and repeated k-means. Specifically, we have studied the factors of cluster overlap, number of clusters, dimensionality and unbalance.

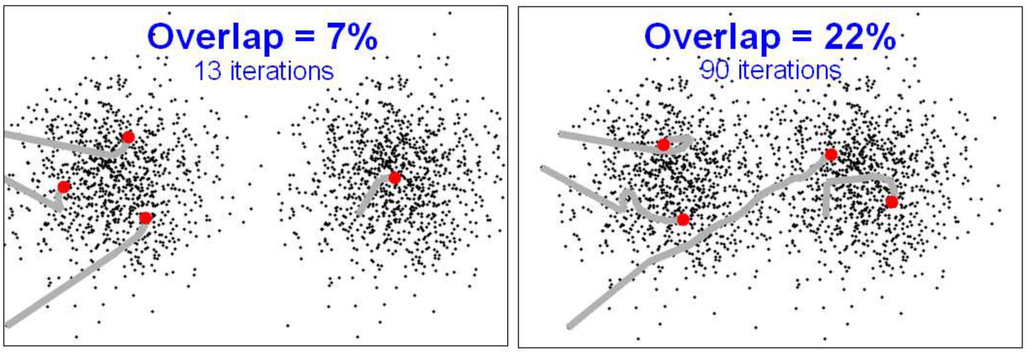


Figure 26. Illustration of the positive effect of overlap for k-means. The gray trajectories show the movement of the centroids during the iterations. In both cases, only one initial centroid is on the rightmost cluster and only when there is sufficient overlap, one additional centroid can move across the clusters.

Cluster overlap is the biggest factor for successful clustering (Figures 26-27**)**. If there is high overlap, k-means iterations work well regardless of the initialization. If there is no overlap, then the success depends completely on the initialization technique: if it fails, k-means will also fail.

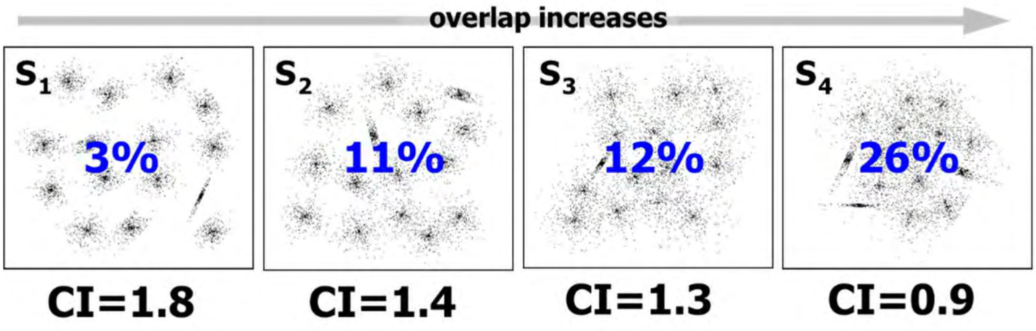


Figure 27. Performance of k-means increases when overlap increases. Performance is measured as success rate (%) and CI-values.

All k-means variants perform worse when the number of clusters increases. Success of the k-means depends linearly on the number of clusters. The more clusters, the more errors there are.

Dimensionality does not have a direct effect. It has a slight effect on some initialization techniques but k-means iterations are basically independent on the dimensions. However, with many types of data, increasing dimensionality also decreases cluster overlap and consequently reduces k-means performance.

Unbalance of cluster sizes (Figure 28) can be problematic especially for the random initializations but also for the other techniques. Only the Maxmin variants with 100 repeats could overcome this problem.

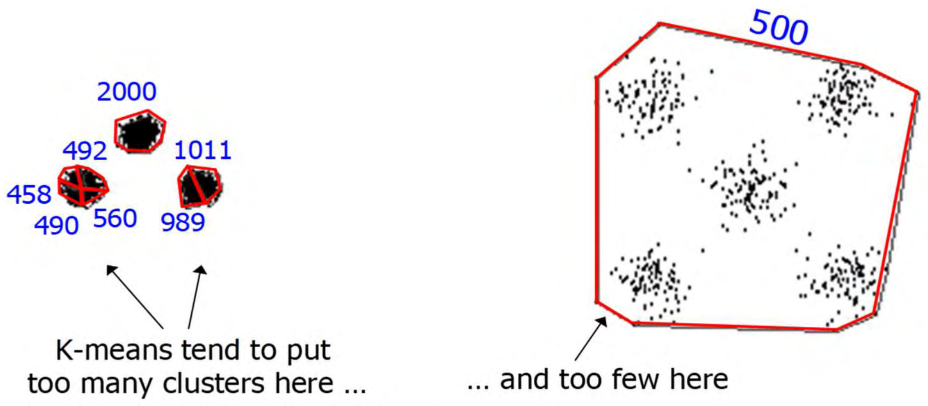


Figure 28. Effect of unbalance for k-means performance demonstrated using the Unbalance dataset. Random initialization of k-means tends to put too many centroids in the dense clusters and too few in the sparse clusters. This results in average CI of 3.9. This dataset cannot be successfully clustered even with 100 repeats. [**P3**]

In [**P4**], we did a benchmark for 9 different k-means initialization methods: Random partition (Rand-P) [93], Random centroids (Rand-C) [11], Maxmin [40], kmeans++ [38], Bradley [37], Sorting [46], Projection [49], Luxburg [39] and Split [**P4**].

Random partition and random centroids are the most trivial and the oldest and therefore considered as baseline methods. Luxburg and Split are more complicated and therefore considered standalone clustering algorithms instead of true initialization methods. The other ones are newer initialization methods.

We found that:

- K-means works better when the clusters overlap. This is the most important factor to predict the success of k-means. Increased data dimensionality only indirectly affects results by reducing overlap.
- On average, k-means caused errors with about 15% of the clusters (CI=4.5). By repeating k means 100 times these errors were reduced to 6% (CI=2.0). Repeats are only useful with those initialization methods that have enough randomness.

Using a better initialization technique (Maxmin), the corresponding numbers were 6% (CI=2.1) with k-means as such, and 1% (CI=0.7) with 100 repeats.

6 Summary of contributions

In this chapter we give a summary of the original publications. In publications [**P1**,**P2**] we introduced new methods for constructing a kNN graph. In publication [**P2**] we also used this graph to speed up Density Peaks clustering. In publications [**P3**,**P4**] we studied the properties and initialization methods of k-means clustering algorithm.

[**P1**]: An exact kNN graph can be constructed fast for low-dimensional data. In case of high-dimensional data, some approximate methods have been developed. In this publication we developed a new approximate kNN construction method that is first such to work using space filling curves. In the experiments, it also performs better than the compared methods. We also show that errors in the approximate kNN-graphs originate more often from outlier points. The proposed method provides average speed-up of 100:1 with the 1,000-dimensional datasets. This is 50% more than the best existing method.

[**P2**]: Density peaks is a clustering algorithm which is used especially for non-spherical data. Its use is limited by quadratic time complexity, which makes it slow for large datasets. In this work, we introduce a new kNN graph construction method and use it to develop a fast variant of Density peaks. This Fast density peaks algorithm is also general in the sense that it can work with all types of data where a distance or similarity between the data objects can be calculated. The proposed algorithm provides speed-up of 100:1 compared to the original Density peaks algorithm with only minor degradation of clustering quality.

[**P3**]: Although k-means is a very popular algorithm, it has problems solving some data sets. Previously it has been unclear whether these problems originate from the algorithm or from the cost function (SSE) that the algorithm optimizes. In this work, we show that K-means fails even if SSE itself would work. We show that the problems arise especially in data where the clusters are well separated. This is unexpected and contrary to many other algorithms which work better when the clusters are more clearly separate.

[**P4**]: Performance of k-means is often tried to improve simply by repeating the algorithm, or by using a more advanced initialization technique. In this article, we study how well these approaches work, depending on the properties of the data. We show that when the data has overlapping clusters, k-means can improve the results of any initialization technique. However, when the data has well separated clusters, the performance of k-means depends completely on the initialization. Among the nine studied initialization techniques, the simple furthest point heuristic (Maxmin) is shown to work best, reducing the clustering error of k-means from 15% to 6%, on average.

7 Summary of results

In this section we give a summary of the clustering results from papers **P2**-**P4**. Results are reported in Table 3. We picked the most important datasets from papers **P2** and **P3** and run those for selected algorithms. All datasets can not be accurately represented by centroids and do not have centroid ground truth. We therefore used the partition version of CI [74] for evaluation. Processing times are measured as run on a single thread.

We selected two main groups of algorithms. First group consists mainly of algorithms studied in [**P3**-**P4**], which optimize the SSE cost function: K-means-RandC, Maxmin, Repeated Maxmin and Random swap. K-means-RandC is the normal k-means algorithm with random centroids initialization. Maxmin is K-means with Maxmin initialization. Repeated Maxmin is the case where K-means/Maxmin was repeated 100 times and best of those results taken as final. Random Swap is chosen as the best known SSE optimizing algorithm.

Second group of algorithms consists of two variants of the Density Peaks algorithm. FastDP was introduced in [**P2**] as a faster variant of original Density Peaks.

All of the SSE optimizing methods and FastDP are non-deterministic and contain some randomness. Results for those were therefore run 50 times and the mean value of those runs are shown in the result table (Table 3). DensityPeaks is a fully deterministic algorithm, so repeating this doesn’t produce any different results. FastDP contains some randomness, but this only manifests in the approximation quality of the kNN graph.

Table 3. Summary of results. Performance is measured as both CI and NMI. In the case of CI, smaller values are better. With NMI values range from 0.0 to 1.0, where larger is better. Each of the sub tables has the same structure: Spherical datasets on the left side. Shape datasets on the right side. SSE-optimizing algorithms on upper part. Density Peaks variants on bottom.

	CI													
	s1	**s2**	**s3**	**s4**	**a3**	**unb**	**b2**	**RCh**	**W2**	**W64**	**agg**	**spi**	**fla**	**DS6**
K-means-RandC	2.1	1.4	1.2	0.9	6.5	3.9	17.0	8.1	11.1	5.1	1.5	1.0	0.0	3.9
Maxmin	0.6	1.0	0.6	1.1	2.8	1.0	7.4	5.8	11.4	20.6	1.6	1.0	0.0	4.1
Repeated Maxmin	0.0	0.0	0.0	0.0	0.6	0.0	3.7	2.1	11.0	2.7	1.0	1.0	0.0	4.0
Random swap	0.0	0.0	0.0	0.0	0.0	0.0	0.0	8.1	10.8	0.0	1.0	1.0	0.0	4.0
FastDP	0.0	0.0	0.0	0.0	0.0	0.0	0.0	18.2	7.0	0.0	0.0	0.0	0.0	3.0
DensityPeaks	0.0	0.0	0.0	0.0	0.0	0.0	0.0	18.0	7.0	0.0	0.0	0.0	0.0	3.0

	NMI													
	s1	**s2**	**s3**	**s4**	**a3**	**unb**	**b2**	**RCh**	**W2**	**W64**	**agg**	**spi**	**fla**	**DS6**
K-means-RandC	0.93	0.90	0.77	0.71	0.95	0.78	0.96	0.98	0.61	0.70	0.85	0.00	0.43	0.53
Maxmin	0.97	0.91	0.78	0.71	0.97	0.79	0.98	0.98	0.60	0.13	0.85	0.00	0.41	0.52
Repeated Maxmin	0.99	0.95	0.79	0.72	0.99	1.00	0.99	0.99	0.60	0.72	0.88	0.00	0.40	0.55
Random swap	0.99	0.95	0.79	0.72	1.00	1.00	1.00	0.98	0.61	0.75	0.88	0.00	0.40	0.55
FastDP	0.99	0.94	0.79	0.72	0.99	1.00	1.00	0.82	0.62	0.74	1.00	1.00	1.00	0.60
DensityPeaks	0.99	0.94	0.79	0.72	0.99	1.00	1.00	0.80	0.62	0.72	1.00	1.00	1.00	0.60

	Time (seconds)													
	s1	**s2**	**s3**	**s4**	**a3**	**unb**	**b2**	**RCh**	**W2**	**W64**	**agg**	**spi**	**fla**	**DS6**
K-means-RandC	0	0	0	0	0	0	1	16	4	16	0	0	0	0
Maxmin	0	0	0	0	0	0	1	18	6	4	0	0	0	0
Repeated Maxmin	1	1	2	3	3	1	108	1631	527	484	0	0	0	0
Random swap	5	6	7	7	9	14	103	8003	373	1833	1	0	0	2
FastDP	0	0	0	0	0	0	7	80	9	78	0	0	0	0
DensityPeaks	1	1	1	1	1	1	271	2656	198	1192	0	0	0	0

The datasets S1-RCh are spherical datasets. They are expected to be solved by good SSE optimizing algorithms. The best of these algorithms, Random Swap can indeed solve all these datasets, except Rch. Density Peaks variants can also solve almost all spherical datasets. The exception is the RCh dataset which is high dimensional and has very high overlap between clusters.

The datasets W2-DS6_8 are non-spherical shape datasets. For these datasets, SSE optimizing algorithms like k-means are not expected to work well. This is indeed true in case of most datasets.

The main exception is the W64 dataset where the shapes are similar as in W2, but are isolated due to the sparsity of high dimensional space. For this reason, random swap can solve this dataset. Maxmin on the other hand, performs poorly, even compared to the random centroids method. This can be explained by the high number of outliers in the dataset which Maxmin tends to pick as initial centroids (see Fig. 25).

Also, all methods are able to solve the flame dataset, according to the CI measure. However, the NMI measure shows that the result partitions still have major differences.

As expected, Density Peaks performs clearly better for all of the shape datasets. There is no significant difference between the quality of FastDP, compared to original DensityPeaks. In terms of speed, FastDP is much faster, with 1:25 mean speedup factor.

8 Conclusions

In this work, we introduced two new methods for fast approximate kNN graph construction. A method called ZNP works well for high dimensional numerical data. It uses a combination of space-filling curves and neighborhood propagation to construct the graph. Second algorithm, RP-Div has less limitations and works for any kind of data where a distance measure is available. Both perform well in comparison to previous state of the art methods.

We used the kNN graph from RP-Div algorithm to speed up the well known Density Peaks algorithm and allow it to cluster datasets up to 1 million in size. Experiments showed an average speed-up of 91:1 on datasets of size ≥ 100,000. The algorithm is also very general and works for all types of data as long as a distance function is provided. As a case study of using text data, we managed to cluster a Nordic Tweet dataset of size 500,000 in 31 minutes.

K-means is a very popular algorithm, but it often performs poorly compared to other algorithms like Random Swap or Ward's method. In this thesis, we studied the situations when k-means works and when it fails (Figure 29). The most important factor turned out to be cluster overlap. When there is more overlap, and less empty space between clusters, k-means works better. We also provided a formula to estimate overlap numerically.

Many studies have tried to improve k-means by various strategies. Better initialization methods or just repeating the algorithm are two most common ones. We studied how well these work in different situations. On average, k-means produces errors in about 15% of the clusters. Better initialization technique (Maxmin) reduced this to 6%. Combining this with 100 repeats reduced the error further to just 1%. However, in case of high overlap, k-means worked well even without better initialization techniques. In case of low overlap, it was the initialization technique that solved the clustering. K-means iterations themselves had only a minor effect.

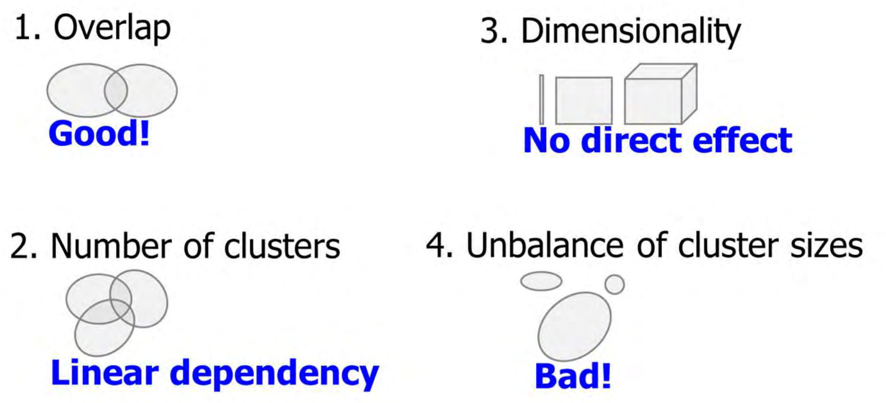


Figure 29. How different properties of a dataset affect the success of k-means clustering.

The findings of this thesis suggest new research areas to explore:

- Lack of overlap is the primary cause of poor k-means performance. This suggests that performance of k-means might be improved by artificially introducing more overlap to a dataset which is lacking it. This would need a suitable way of detecting lack of overlap, without access to clustering ground truth. Also different overlap decreasing transformations should be studied, e.g. noise models or neighborhood embedding.
- We were able to improve the speed of Density Peaks using a kNN graph. Could k-means also be made faster using a kNN graph? In the assignment step, distances need to be calculated from every point to all *k* centroids. This could be reduced to just the kNN neighbors of the old nearest centroid. This would make the algorithm significantly faster in case of large *k*, but would performance degrade too much?
- We also showed that errors in the approximate kNN-graph originate more likely from outlier points, and those can be detected during runtime. It might be possible to use this to improve results of any approximate kNN-graph algorithm by focusing a more extensive search on outlier points.

References

[1] S. Fortunato, Community detection in graphs, Physics reports, 486 (2010), pp. 75—174.

[2] L. Fu and E. Medico, FLAME, a novel fuzzy clustering method for the analysis of DNA microarray data, BMC bioinformatics, 8 (2007), p. 3.

[3] K. Lu, S. Xia, and C. Xia, Clustering based road detection method, in Control Conference (CCC), 2015 34th Chinese, IEEE, 2015, pp. 3874—3879.

[4] Y. Zhang, G. Li, X. Xie, and Z. Wang, A new algorithm for fast and accurate moving object detection based on motion segmentation by clustering, in Machine Vision Applications (MVA), 2017 Fifteenth IAPR International Conference on, IEEE, 2017, pp. 444—447.

[5] A. Rodriguez and A. Laio, Clustering by fast search and find of density peaks, Science, 344 (2014), pp. 1492—1496.

[6] H. Liu, H. Guan, J. Jian, X. Liu, and Y. Pei, Clustering based on words distances, Cluster Computing, (2017), pp. 1—9.

[7] B. Wang, J. Zhang, Y. Liu, and Y. Zou, Density peaks clustering based integrate framework for multi-document summarization, CAAI Transactions on Intelligence Technology, 2 (2017), pp. 26—30.

[8] A. Clauset, M. E. Newman, and C. Moore, Finding community structure in very large networks, Physical Review E, 70 (2004), p. 066111.

[9] S. Kaski, J. Nikkila, J. Sinkkonen, L. Lahti, J. E. Knuuttila, and C. Roos, Associative clustering for exploring dependencies between functional genomics data sets, IEEE/ACM Transactions on Computational Biology and Bioinformatics, 2 (2005), pp. 203—216.

[10] A. K. Jain, Data clustering: 50 years beyond K-means, Pattern Recognition Letters, 31 (2010), pp. 651—666.

[11] J. MacQueen et al., Some methods for classification and analysis of multivariate observations, in Proceedings of the fifth Berkeley Symposium on Mathematical Statistics and Probability, vol. 1, Oakland, CA, USA, 1967, pp. 281—297.

[12] E. Forgy, Cluster analysis of multivariate data: Efficiency vs. interpretability of classification, Biometrics, 21 (1965), pp. 768—769.

[13] J. H. Ward Jr, Hierarchical grouping to optimize an objective function, Journal of the American Statistical Association, 58 (1963), pp. 236—244.

[14] P. Fränti, Genetic algorithm with deterministic crossover for vector quantization, Pattern Recognition Letters, 21 (2000), pp. 61—68.

[15] P. Fränti, Efficiency of random swap clustering, Journal of Big Data, 5 (2018), p. 13.

[16] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al., A density-based algorithm for discovering clusters in large spatial databases with noise, in AAAI conference on Knowledge Discovery and Data Mining, 1996, pp. 226—231.

[17] D. Comaniciu and P. Meer, Mean shift: A robust approach toward feature space analysis, IEEE Transactions on Pattern Analysis and Machine Intelligence, 24 (2002), pp. 603—619.

[18] A. K. Jain and R. C. Dubes, Algorithms for clustering data, Prentice-Hall, Inc., 1988.

[19] Y. Qin, Z. L. Yu, C.-D. Wang, Z. Gu, and Y. Li, A Novel clustering method based on hybrid K-nearest-neighbor graph, Pattern Recognition, (2018), pp. 1—14.

[20] B. J. Frey and D. Dueck, Clustering by passing messages between data points, science, 315 (2007), pp. 972—976.

[21] C.-D. Wang, J.-H. Lai, C. Y. Suen, and J.-Y. Zhu, Multi-exemplar affinity propagation, IEEE transactions on Pattern Analysis and Machine Intelligence, (2013), pp. 2223—2237.

[22] A. Ben-Hur, D. Horn, H. T. Siegelmann, and V. Vapnik, Support vector clustering, Journal of Machine Learning Research, 2 (2001), pp. 125—137.

[23] I. S. Dhillon, Y. Guan, and B. Kulis, Kernel k-means: spectral clustering and normalized cuts, in Proceedings of the tenth ACM SIGKDD international conference on Knowledge Discovery and Data Mining, 2004, pp. 551—556.

[24] D. Yan, L. Huang, and M. I. Jordan, Fast approximate spectral clustering, in Proceedings of the 15th ACM SIGKDD international conference on Knowledge Discovery and Data Mining, 2009, pp. 907—916.

[25] P. Fränti, O. Virmajoki, and V. Hautamäki, Fast agglomerative clustering using a k-nearest neighbor graph, IEEE Transactions on Pattern Analysis and Machine Intelligence, 28 (2006), pp. 1875—1881.

[26] Y. Chen, L. Zhou, S. Pei, Z. Yu, Y. Chen, X. Liu, J. Du, and N. Xiong, Knn-block dbscan: Fast clustering for large-scale data, IEEE Transactions on Systems, Man, and Cybernetics: Systems, (2019).

[27] M. Du, S. Ding, and H. Jia, Study on density peaks clustering based on k-nearest neighbors and principal component analysis, Knowledge-Based Systems, 99 (2016), pp. 135—145.

[28] X. Zhu, Semi-supervised learning with graphs, PhD thesis, Carnegie Mellon University, Language Technologies Institute, School of Computer Science, 2005.

[29] K. Hajebi, Y. Abbasi-Yadkori, H. Shahbazi, and H. Zhang, Fast approximate nearest-neighbor search with k-nearest neighbor graph, in IJCAI Proceedings-International Joint Conference on Artificial Intelligence, vol. 22, 2011, p. 1312.

[30] M. Belkin and P. Niyogi, Laplacian eigenmaps for dimensionality reduction and data representation, Neural Computation, 15 (2003), pp. 1373—1396.

[31] V. Hautamäki, I. Kärkkäinen, and P. Fränti, Outlier Detection Using k-Nearest Neighbour Graph, in International Conference on Pattern Recognition (3), 2004, pp. 430—433.

[32] M. Connor and P. Kumar, Fast construction of k-nearest neighbor graphs for point clouds, IEEE Transactions on Visualization and Computer Graphics, 16 (2010), pp. 599—608.

[33] S. Sieranoja and P. Fränti, Fast random pair divisive construction of kNN graph using generic distance measures, in Proceedings of the 2018 International Conference on Big Data and Computing, 2018, pp. 95—98.

[34] L. Morissette and S. Chartier, The k-means clustering technique: General considerations and implementation in Mathematica, Tutorials in Quantitative Methods for Psychology, 9 (2013), pp. 15—24.

[35] J. Liang, L. Bai, C. Dang, and F. Cao, The K-Means-Type Algorithms Versus Imbalanced Data Distributions, IEEE Transactions on Fuzzy Systems, 20 (2012), pp. 728—745.

[36] A. Hinneburg and D. A. Keim, Optimal grid-clustering : Towards breaking the curse of dimensionality in high-dimensional clustering, in Proceedings of the 25 th International Conference on Very Large Databases, 1999, 1999, pp. 506—517.

[37] P. S. Bradley and U. M. Fayyad, Refining initial points for k-means clustering, in International Conference on Machine Learning, 1998, pp. 91—99.

[38] D. Arthur and S. Vassilvitskii, k-means++: The advantages of careful seeding, in Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, 2007, pp. 1027—1035.

[39] U. Von Luxburg et al., Clustering stability: an overview, Foundations and Trends in Machine Learning, 2 (2010), pp. 235—274.

[40] T. F. Gonzalez, Clustering to minimize the maximum intercluster distance, Theoretical Computer Science, 38 (1985), pp. 293—306.

[41] D. Steinley and M. J. Brusco, Initializing k-means batch clustering: A critical evaluation of several techniques, Journal of Classification, 24 (2007), pp. 99—121.

[42] M. B. Al-Daoud, A new algorithm for cluster initialization, in World Enformatika Conference, 2005.

[43] F. Cao, J. Liang, and G. Jiang, An initialization method for the K-Means algorithm using neighborhood model, Computers & Mathematics with Applications, 58 (2009), pp. 474—483.

[44] M. E. Celebi, H. A. Kingravi, and P. A. Vela, A comparative study of efficient initialization methods for the k-means clustering algorithm, Expert Systems with Applications, 40 (2013), pp. 200—210.

[45] M. M.-T. Chiang and B. Mirkin, Intelligent choice of the number of clusters in k-means clustering: an experimental study with different cluster spreads, Journal of Classification, 27 (2010), pp. 3—40.

[46] J. A. Hartigan and M. A. Wong, Algorithm AS 136: A k-means clustering algorithm, Journal of the Royal Statistical Society. Series C (Applied Statistics), 28 (1979), pp. 100—108.

[47] J. M. Pena, J. A. Lozano, and P. Larranaga, An empirical comparison of four initialization methods for the k-means algorithm, Pattern Recognition Letters, 20 (1999), pp. 1027—1040.

[48] X. Wu and K. Zhang, A better tree-structured vector quantizer, in Data Compression Conference, IEEE, 1991, pp. 392—401.

[49] S. Sieranoja and P. Fränti, Random projection for k-means clustering, in International Conference on Artificial Intelligence and Soft Computing, Springer, 2018, pp. 680—689.

[50] M. E. Celebi and H. A. Kingravi, Deterministic initialization of the k-means algorithm using hierarchical clustering, International Journal of Pattern Recognition and Artificial Intelligence, 26 (2012), p. 1250018.

[51] J. He, M. Lan, C.-L. Tan, S.-Y. Sung, and H.-B. Low, Initialization of cluster refinement algorithms: A review and comparative study, in 2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No. 04CH37541), vol. 1, IEEE, 2004, pp. 297—302.

[52] P. Indyk, High-dimensional computational geometry, PhD thesis, Stanford University, Palo Alto, CA, 2000.

[53] C. Zhong, M. Malinen, D. Miao, and P. Fränti, A fast minimum spanning tree algorithm based on k-means, Information Sciences, 295 (2015), pp. 1—17.

[54] L. Sengupta and P. Fränti, Predicting difficulty of tsp instances using mst, in Proceedings of the IEEE International Conference on Industrial Informatics (INDIN), Helsinki, Finland, 2019, pp. 848—852.

[55] D. Dua and C. Graff, UCI machine learning repository, 2017.

[56] D. Steinley, Local optima in K-means clustering: what you don't know may hurt you., Psychological Methods, 8 (2003), p. 294.

[57] I. Kärkkäinen and P. Fränti, Dynamic local search algorithm for the clustering problem, Tech. Rep. A-2002-6, Department of Computer Science, University of Joensuu, Joensuu, Finland, 2002.

[58] P. Fränti and O. Virmajoki, Iterative shrinking method for clustering problems, Pattern Recognition, 39 (2006), pp. 761—775.

[59] P. Fränti, R. Mariescu-Istodor, and C. Zhong, XNN graph, in Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR), vol. 10029, Springer, 2016, pp. 207—217.

[60] T. Zhang, R. Ramakrishnan, and M. Livny, BIRCH: A new data clustering algorithm and its applications, Data Mining and Knowledge Discovery, 1 (1997), pp. 141—182.

[61] M. Rezaei and P. Fränti, Set matching measures for external cluster validity, IEEE Transactions on Knowledge and Data Engineering, 28 (2016), pp. 2173—2186.

[62] A. Gionis, H. Mannila, and P. Tsaparas, Clustering aggregation, ACM Transactions on Knowledge Discovery from Data (TKDD), 1 (2007), pp. 4—es.

[63] H. Chang and D.-Y. Yeung, Robust path-based spectral clustering, Pattern Recognition, 41 (2008), pp. 191—203.

[64] M. Laitinen, J. Lundberg, M. Levin, and A. Lakaw, Utilizing Multilingual Language Data in (Nearly) Real Time: The Case of the Nordic Tweet Stream, Journal of universal computer science, 23 (2017), pp. 1038—1056.

[65] M. Steinbach, L. Ertoz, and V. Kumar, The challenges of clustering high dimensional data. New Vistas in Statistical Physics: Applications in Econophysics, Bioinformatics, and Pattern Recognition, 207312 (2003).

[66] C. C. Aggarwal, A. Hinneburg, and D. A. Keim, On the Surprising Behavior of Distance Metrics in High Dimensional Spaces, in Proceedings of the 8th International Conference on Database Theory, ICDT '01, London, UK, 2001, Springer-Verlag, pp. 422—434.

[67] A. Hinneburg, C. C. Aggarwal, and D. A. Keim, What is the nearest neighbor in high dimensional spaces?, in Proceedings of the 26th International Conference on Very Large Databases, Cario, Egypt, 2000.

[68] C. Domeniconi, D. Gunopulos, S. Ma, B. Yan, M. Al-Razgan, and D. Papadopoulos, Locally adaptive metrics for clustering high dimensional data, Data Mining and Knowledge Discovery, 14 (2007), pp. 63—97.

[69] E. Chávez and G. Navarro, A probabilistic spell for the curse of dimensionality, in Workshop on Algorithm Engineering and Experimentation, vol. 2153, Springer, 2001, pp. 147—160.
[70] N. Gali, R. Mariescu-Istodor, D. Hostettler, and P. Fränti, Framework for syntactic string similarity measures, Expert Systems with Applications, 129 (2019), pp. 169—185.
[71] V. I. Levenshtein, Binary codes capable of correcting deletions, insertions, and reversals, in Soviet Physics Doklady, vol. 10, 1966, pp. 707—710.
[72] S. Jimenez, C. Becerra, A. Gelbukh, and F. Gonzalez, Generalized mongue-elkan method for approximate text string comparison, in International Conference on Intelligent Text Processing and Computational Linguistics, Springer, 2009, pp. 559—570.
[73] L. Kaufman and P. J. Rousseeuw, Clustering by means of medoids. Statistical data analysis based on the l1 norm, Y. Dodge, Ed, (1987), pp. 405—416.
[74] P. Fränti and M. Rezaei, Generalizing centroid index to different clustering models, in Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR), Springer, 2016, pp. 285—296.
[75] P. Fränti, M. Rezaei, and Q. Zhao, Centroid index: cluster level similarity measure, Pattern Recognition, 47 (2014), pp. 3034—3045.
[76] T. Debatty, P. Michiardi, O. Thonnard, and W. Mees, Scalable graph building from text data, in Proceedings of the 3rd International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications, 2014, pp. 120—132.
[77] J. Wang, J. Wang, G. Zeng, Z. Tu, R. Gan, and S. Li, Scalable k-NN graph construction for visual descriptors, in Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on, IEEE, 2012, pp. 1106—1113.
[78] A. Flexer and J. Stevens, Mutual proximity graphs for improved reachability in music recommendation, Journal of New Music Research, 47 (2018), pp. 17—28.

[79] J. H. Friedman, J. L. Bentley, and R. A. Finkel, An Algorithm for Finding Best Matches in Logarithmic Expected Time, ACM Transactions on Mathematical Software (TOMS), 3 (1977), pp. 209—226.

[80] O. Virmajoki and P. Fränti, Divide-and-conquer algorithm for creating neighborhood graph for clustering, in Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on, vol. 1, IEEE, 2004, pp. 264—267.

[81] J. Chen, H.-r. Fang, and Y. Saad, Fast approximate k NN graph construction for high dimensional data via recursive Lanczos bisection, The Journal of Machine Learning Research, 10 (2009), pp. 1989—2012.

[82] C. Fu and D. Cai, EFANNA : An Extremely Fast Approximate Nearest Neighbor Search Algorithm Based on kNN Graph, CoRR, abs/1609.07228 (2016).

[83] Y.-M. Zhang, K. Huang, G. Geng, and C.-L. Liu, Fast kNN Graph Construction with Locality Sensitive Hashing, in Machine Learning and Knowledge Discovery in Databases, Springer, 2013, pp. 660—674.

[84] W. Dong, C. Moses, and 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.

[85] G. M. Morton, A computer oriented geodetic data base and a new technique in file sequencing, International Business Machines Company, 1966.

[86] J. A. Orenstein and T. H. Merrett, A class of data structures for associative searching, in Proceedings of the 3rd ACM SIGACT-SIGMOD symposium on Principles of Database Systems, ACM, 1984, pp. 181—190.

[87] H. Tropf and H. Herzog, Multidimensional Range Search in Dynamically Balanced Trees., ANGEWANDTE INFO., (1981), pp. 71—77.

[88] N. Ailon and B. Chazelle, The Fast Johnson-Lindenstrauss Transform and Approximate Nearest Neighbors, SIAM Journal on Computing, 39 (2009), pp. 302—322.

[89] Z. Yang, J. Peltonen, and S. Kaski, Optimization equivalence of divergences improves neighbor embedding, in International Conference on Machine Learning, 2014, pp. 460—468.

[90] N. Galiauskas and J. Žilinskas, On Multidimensional Scaling with City-Block Distances, In International Conference on Learning and Intelligent Optimization, Springer, 2014, pp. 82—87.

[91] J. Wang, Y. Zhang, and 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.

[92] J. Hou and 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.

[93] S. Lloyd, Least squares quantization in PCM, IEEE Transactions on Information Theory, 28 (1982), pp. 129—137.

[94] I. Katsavounidis, C.-C. J. Kuo, and Z. Zhang, A new initialization technique for generalized Lloyd iteration, IEEE Signal Processing Letters, 1 (1994), pp. 144—146.

[95] F. Cao, J. Liang, and L. Bai, A new initialization method for categorical data clustering, Expert Systems with Applications, 36 (2009), pp. 10223—10228.

[96] H. J. Curti and R. S. Wainschenker, FAUM: Fast Autonomous Unsupervised Multidimensional classification, Information Sciences, 462 (2018), pp. 182—203.

PAPER 2

S. Sieranoja and P. Fränti,
“Fast and general density peaks clustering”,
Pattern Recognition Letters,
128, 551-558, December 2019.
https://doi.org/10.1016/j.patrec.2019.10.019

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
0167-8655/© 2019 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license. (http://creativecommons.org/licenses/by/4.0/)

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 $BigBrother(x) \notin \mathrm{kNN}(x)$
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].

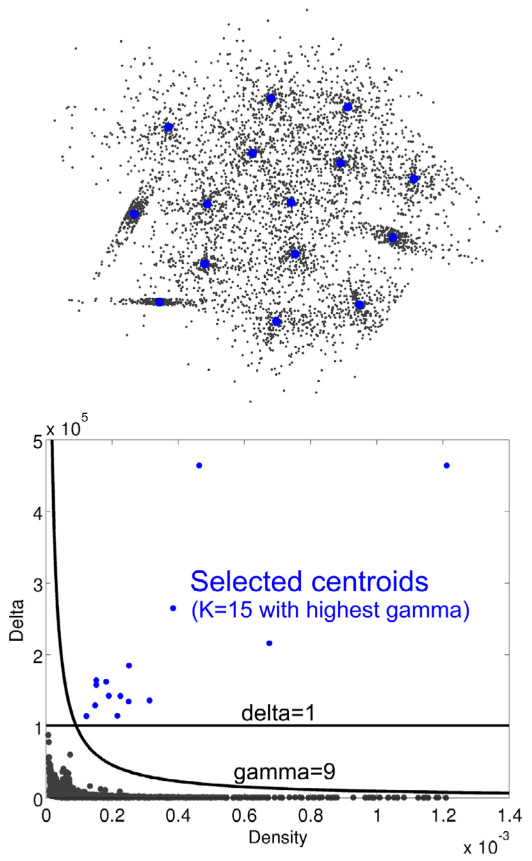


Fig. 1. Different cluster selection strategies based on the density-vs-delta plot for the S4 dataset. Cluster centroids typically have both high density and high distance to higher density point (delta). Therefore, thresholding based on combination of delta and density (gamma) is expected to work better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

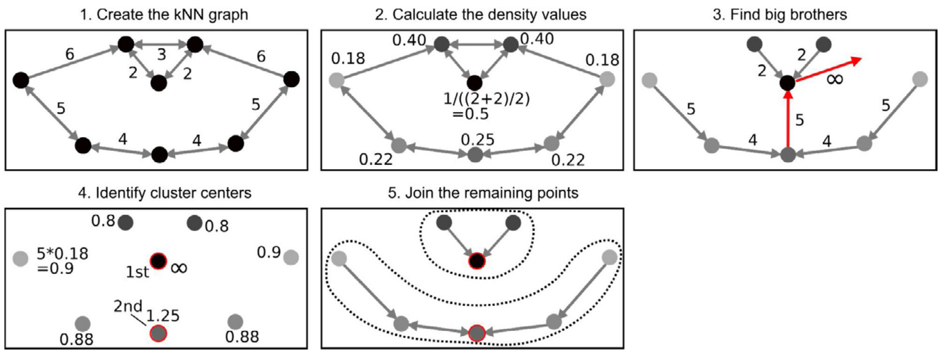


Fig. 2. Illustration of the proposed Fast density peaks algorithm. (1) For a given data set, the kNN graph is constructed. (2) Densities are calculated as inverse of the mean distance to the neighbors. (3) Nearest higher density point (big brother) is (in case of black lines) found in the kNN graph; 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);
```

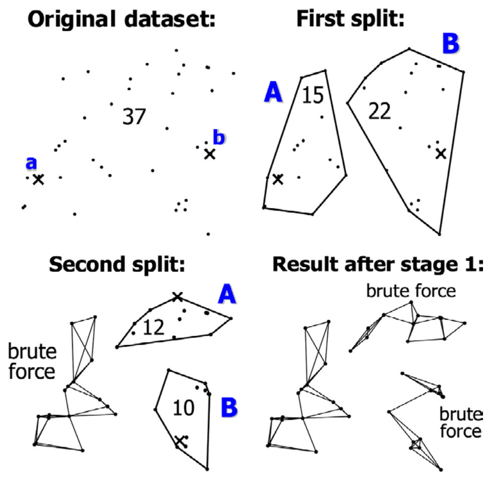


Fig. 3. The RP-div algorithm recursively subdivides the dataset of size $N = 37$ by first choosing two random points (a,b). The dataset is split based on which of the two points is nearer. After the first split, the size of the subset A is smaller than threshold $W = 20$, and is solved by brute force.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

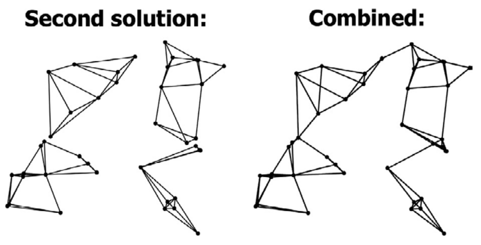


Fig. 4. After repeating the random pair division, a new solution is obtained. This is solution is merged with the previous one to form a new improved kNN graph.

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

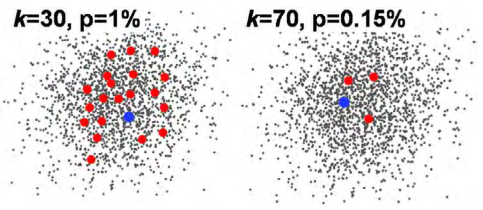


Fig. 5. Distribution of slope points (gray) and local peaks (red) inside an example cluster. One of the local peaks (blue) is the resulting cluster centroid (global peak). The case of $k = 30$ (left) and $k = 70$ (right) are shown. When the number of neighbors k in the kNN graph is increased, the number 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.

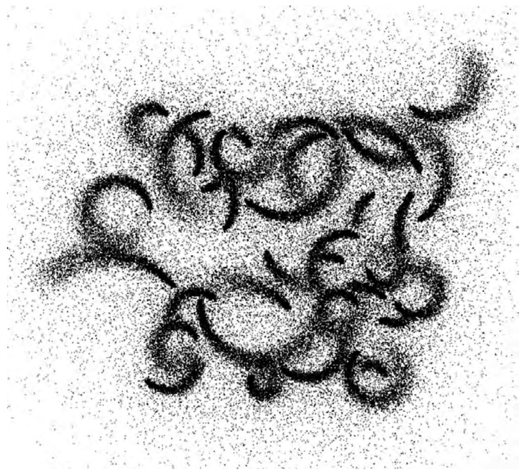


Fig. 6. The Worms2 dataset contains 35 shapes which depict trails of random movement in 2D space.

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:

$$CI(C,G) = \max\left(Orphans(C \to G), Orphans(G \to C)\right) \tag{1}$$

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]:

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

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.

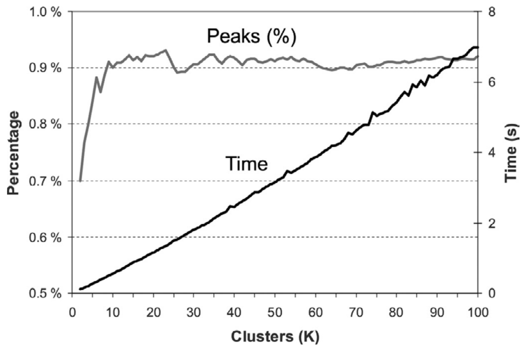


Fig. 7. Dependency of the proportion of local peaks on the number of clusters (K). The corresponding Fast-DensP processing time depends linearly on the data size, which is $N = 1000 \cdot K$.

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/DMYvpOI54i #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.

PAPER 4

P. Fränti and S. Sieranoja,
“How much k-means can be improved by using better initialization and repeats?”,
Pattern Recognition,
93, 95-112, 2019.
https://doi.org/10.1016/j.patcog.2019.04.014

Pattern Recognition 93 (2019) 95–112

Contents lists available at ScienceDirect

Pattern Recognition

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

How much can k-means be improved by using better initialization and repeats?

Pasi Fränti, Sami Sieranoja*

Machine Learning Group, School of Computing, University of Eastern Finland, P.O. Box 111, FIN-80101 Joensuu, Finland

ARTICLE INFO

Article history:
Received 13 August 2018
Revised 11 March 2019
Accepted 13 April 2019
Available online 15 April 2019

Keywords:
Clustering algorithms
K-means
Initialization
Clustering accuracy
Prototype selection

ABSTRACT

In this paper, we study what are the most important factors that deteriorate the performance of the k-means algorithm, and how much this deterioration can be overcome either by using a better initialization technique, or by repeating (restarting) the algorithm. Our main finding is that when the clusters overlap, k-means can be significantly improved using these two tricks. Simple furthest point heuristic (Maxmin) reduces the number of erroneous clusters from 15% to 6%, on average, with our clustering benchmark. Repeating the algorithm 100 times reduces it further down to 1%. This accuracy is more than enough for most pattern recognition applications. However, when the data has well separated clusters, the performance of k-means depends completely on the goodness of the initialization. Therefore, if high clustering accuracy is needed, a better algorithm should be used instead.

1. Introduction

K-means (KM) algorithm [1–3] groups *N* data points into *k* clusters by minimizing the sum of squared distances between every point and its nearest cluster mean (*centroid*). This objective function is called *sum-of-squared errors* (SSE). Although k-means was originally designed for minimizing SSE of numerical data, it has also been applied for other objective functions (even some non-numeric).

Sometimes the term *k-means* is used to refer to the clustering problem of minimizing SSE [4–7]. However, we consider here k-means as an *algorithm*. We study how well it performs as a clustering algorithm to minimize the given objective function. This approach follows the recommendation in [8] to establish a clear distinction between the *clustering method* (objective function) and the *clustering algorithm* (how it is optimized).

In real-life applications, the selection of the objective function is much more important. Clustering results depend primarily on the selected objective function, and only secondarily on the selected algorithm. Wrong choice of the function can easily reverse the benefit of a good algorithm so that a proper objective function with a worse algorithm can provide better clustering than good algorithm with wrong objective function. However, it is an open question how much clustering results are biased because of using an inferior algorithm.

There are other algorithms that are known, in many situations, to provide better clustering results than k-means. However, k-means is popular for good reasons. First, it is simple to implement. Second, people often prefer to use an extensively studied algorithm whose limitations are known rather than a potentially better, but less studied, algorithm that might have unknown or hidden limitations. Third, the local fine-tuning capability of k-means is very effective, and for this reason, it is also used as part of better algorithms such as the genetic algorithm [9,10], random swap [11,12], particle swarm optimization [13], spectral clustering [14], and density clustering [15]. Therefore, our results can also help better understand those more complex algorithms that rely on the use of k-means.

K-means starts by selecting *k* random data points as the initial set of centroids, which is then improved by two subsequent steps. In the *assignment step*, every point is put into the cluster of the nearest centroid. In the *update step*, the centroid of every cluster is recalculated as the mean of all data points assigned to the cluster. Together, these two steps constitute one *iteration* of k-means. These steps fine-tune both the cluster borders and the centroid locations. The algorithm is iterated a fixed number of times, or until convergence (no further improvement is obtained). MacQueen also presented *sequential* variant of k-means [2], where the centroid is updated immediately after every single assignment.

K-means has excellent fine-tuning capabilities. Given a rough allocation of the initial cluster centroids, it can usually optimize

* Corresponding author.
E-mail addresses: pasi.franti@uef.fi (P. Fränti), sami.sieranoja@uef.fi, samisi@cs.uef.fi (S. Sieranoja).

https://doi.org/10.1016/j.patcog.2019.04.014

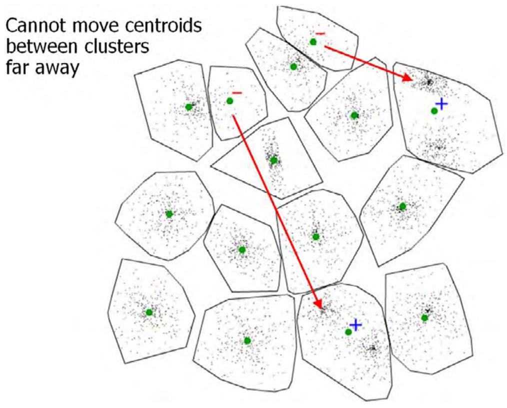


Fig. 1. K-means is excellent in fine-tuning cluster borders locally but fails to relocate the centroids globally. Here a minus sign (−) represents a centroid that is not needed, and a plus sign (+) a cluster where more centroids would be needed. K-means cannot do it because there are stable clusters in between.

their locations locally. However, the main limitation of k-means is that it rarely succeeds in optimizing the centroid locations globally. The reason is that the centroids cannot move between the clusters if their distance is big, or if there are other stable clusters in between preventing the movements, see Fig. 1. The k-means result therefore depends a lot on the initialization. Poor initialization can cause the iterations to get stuck into an inferior local minimum.

Fortunately, finding the exact optimum is not always important. In pattern recognition applications, the goal can be merely to model the distribution of the data, and the clustering result is used as a part in a more complex system. In [16], the quality of the clustering was shown not to be critical for the speaker recognition performance when any reasonable clustering algorithm, including repeated k-means, was used.

However, if the quality of clustering is important then k-means algorithm has problems. For example, if we need to solve the number of clusters, the goodness of the algorithm matters much more. Experiments with three different indexes (WB, DBI, Dunn) have shown that k-means rarely achieves the correct number of clusters whereas random swap succeeded in most cases [17]. Similar observations were made with stability-based approach in [18].

To compensate for the mentioned weaknesses of k-means, two main approaches have been considered: (1) using a better initialization, (2) repeating k-means several times by different initial solution. Numerous initialization techniques have been presented in the literature, including the following:

- Random points
- Furthest point heuristic
- Sorting heuristic
- Density-based
- Projection-based
- Splitting technique

Few comparative studies exists [19–22], but there is no consensus of which technique should be used. A clear state-of-the-art is missing. Pena et al. [19] studied four basic variants: *random centroids* [1] and *MacQueen's variant* of it [2], *random partition* and Kaufman's variant of the *Maxmin heuristic* [23]. Their results show that random partition and Maxmin outperform the random centroid variants with the three datasets (Iris, Ruspini, Glass).

He et al. [20] studied random centroids, random perturbation of the mean [24], greedy technique [25], Maxmin [26], and Kaufman's variant of Maxmin [23]. They observed that the Maxmin variants provide slightly better performance. Their argument is that the Maxmin variants are based on distance optimization, which tends to help k-means provide better cluster separation.

Steinley and Brusco [21] studied 12 variants including complete algorithms like *agglomerative clustering* [27] and *global k-means* [28]. They ended up recommending these two algorithms and Steinley's variant [29] without much reservation. The first two are already complete stand-alone algorithms themselves and not true initialization techniques, whereas the last one is a trivial improvement of the random partition.

Steinley and Brusco also concluded that agglomerative clustering should be used only if the size, dimensionality or the number of clusters is big; and that global k-means (GKM) [28] should be used if not enough memory to store the N^2 pairwise distances. However, these recommendations are not sound. First, agglomerative clustering can be implemented without storing the distance matrix [30]. Second, GKM is extremely slow and not practical for bigger datasets. Both these alternatives are also standalone algorithms and they provide better clustering even without k-means.

Celebi et al. [22] performed the most extensive comparison so far with 8 different initialization techniques on 32 real and 12,228 synthetic datasets. They concluded that random centroids and Maxmin often perform poorly and should not be used, and that there are significantly better alternatives with comparable computational requirements. However, their results do not clearly point out a single technique that would be consistently better than others.

The detailed results in [22] showed that a sub-sampling and repeat strategy [31] performs consistently in *the best group* and k-means++ performs *generally well*. For small datasets Bradley's sub-sampling strategy or greedy variant of k-means++ was recommended. For large data, split-based algorithm was recommended.

The second major improvement, besides the initializations, is to repeat k-means [32]. The idea is simply to restart k-means several times from different initial solution to produce several candidate solutions, and then keeping the best result found as the final solution. This approach requires that the initialization technique produces different starting solutions by involving some randomness in the process. We call this variant *repeated k-means* (RKM). The number of repeats is typically small like $R = 20$ in [33].

Many researchers consider the repeats as an obvious and necessary improvement to the k-means at the cost of increased processing time. Bradley and Fayyad [31] used slightly different variant by combining the repeats and sub-sampling to avoid the increase in the processing time. Besides these papers, it is hard to find any systematic study how the repeats affect on the k-means. For example, none of the review papers investigate the effect of the repeats on the performance.

To sum up, existing literature provides merely relative comparisons between the selected initialization techniques. They lack clear answers of the significance of the results, and present no analysis on which type of data the techniques work and fail. Many of the studies also use classification datasets, which have limited suitability for studying the clustering performance.

We made a brief survey about how recent research papers apply k-means. Random centroids [5,34,35] seems to be the most popular initialization method, along with k-means++ [6,33,36]. Some papers do not specify how they initialize [37], or it had to be concluded indirectly. For example, Boutsidis [5] used the default method available in MATLAB, which was random centroids in the 2014a version and k-means++ starting from the 2014b version. The method in [38] initializes both the centroids and the partition labels at random. However, as they apply the centroid step first, the random partition is effectively applied.

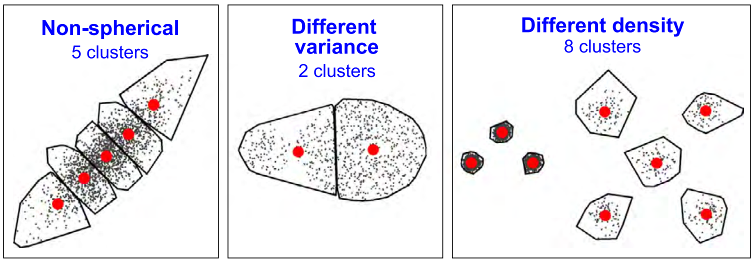


Fig. 2. Three examples of clustering result when using SSE cost function. Gaussian cluster is split into several spherical clusters (left); mismatch of the variance causes the larger cluster to be split (middle); mismatch of the cluster sizes does not matter if the clusters are well-separated.

The number of k-means repeats varies from a relatively small amount of 10–20 [5,33,35] to a relatively high value of 100 [36]. The most extreme example is [34] where 20 h time limit is applied. Although they stop iterating if the running time grows twice as that of their proposed algorithm, it is still quite extensive. Several papers do not repeat k-means at all [6,7,37].

The choice of the initialization and the number of repeats might also vary depending on the motivation. The aim of using k-means can be to have a good clustering result, or to provide merely a point of comparison. In the first case, all the good tricks are used, such as more repeats and better initialization. In the second case, some simpler variant is more likely applied. A counter-example is in [34] where serious efforts seem to be made to ensure all algorithms have the best possible performance.

In this paper we study the most popular initialization heuristics. We aim at answering the following questions. First, to what extent k-means can be improved by a better initialization technique? Second, can the fundamental weakness of k-means be eliminated simply by repeating the algorithm several times? Third, can we predict under which conditions k-means works, and which it fails?

In a recent study [39], it was shown that k-means performs poorly when the clusters are well separated. Here we will answer how much a better initialization or repeats can compensate for this weakness. We will also show that dimensionality does not matter for most variants, and that unbalance of cluster sizes deteriorates the performance of most initializations.

The rest of the paper is organized as follows. In Section 2, we define the methodology and data. We also give brief review of the properties of the standard k-means algorithm. Different initialization techniques are then studied in Section 3. Experimental analysis is performed in Section 4, and conclusions are drawn in Section 5.

2. Performance of k-means

Following the recommendation of Jain [8], we make a clear distinction between the clustering method and algorithm. *Clustering method* refers to the objective function, and *clustering algorithm* to the process optimizing it. Without this distinction, it would be easy to draw wrong conclusions.

For example, k-means has been reported to work poorly with unbalanced cluster sizes [40], and that it can cause large clusters to be wrongly split and smaller clusters wrongly merged [41]. These observations themselves are correct but they miss the root cause, which is the SSE objective function. Even an optimal algorithm minimizing SSE would end up with the same incorrect result. Such observations therefore relate to the objective function, and not to the k-means *algorithm*.

Fig. 2 demonstrates the situation. An algorithm minimizing SSE would find spherical clusters regardless of the data. If the data contain non-spherical clusters, they would be divided into spherical sub-clusters, usually along the direction of the highest variance. Clusters of variable sizes would also cause large clusters to be split, and smaller ones to be merged. In these cases, if natural clusters are wanted, a better clustering result could be achieved by using an objective function based on *Mahalanobis distance* [42] or *Gaussian mixture model* [43] instead of SSE.

2.1. Datasets

In this paper, we focus on the algorithmic performance of k-means rather than the choice of the objective function. We use the *clustering basic benchmark* [39] as all these datasets can be clustered correctly with SSE. Therefore, any clustering errors made by k-means must originate from the properties of the algorithm, and not from the choice of wrong objective function. The datasets are summarized in Table 1. They are designed to vary the following properties as defined in [39]:

- Cluster overlap
- Number of clusters
- Dimensionality
- Unbalance of cluster sizes

2.2. Methodology

To measure the success of the algorithm, the value of the objective function itself is the most obvious measure. Existing literature reviews of k-means use either SSE [19,22], or the deviation of the clusters [20], which is also a variant of SSE. It is calculated as:

$$\mathrm{SSE} = \sum_{i=1}^{N} \left\| x_i - c_j \right\|^2 \quad (1)$$

where x_i is a data point and c_j is its nearest centroid. In [39], SSE is also measured relative to the SSE-value of the ground truth solution (SSE_{opt}):

$$\varepsilon - \mathrm{ratio} = \frac{(SSE - SSE_{opt})}{SSE_{opt}} \quad (2)$$

If the ground truth is known, external indexes such as *adjusted Rand index* (ARI), *Van Dongen* (VD), *variation of information* (VI) or *normalized mutual information* (NMI) can also be used [22]. A comparative study of several suitable indexes can be found in [44]. The number of iterations have also been studied in [19,22], and the time complexities reported in [22].

The problem of SSE, and most of the external indexes, is that the raw value does not tell how significant the result is. We therefore use *Centroid Index* (CI) [45], which indicates how many cluster

Table 1
Basic clustering benchmark [39]. The data is publicly available here: http://cs.uef.fi/sipu/datasets/.

Dataset	Varying	Size	Dimensions	Clusters	Per cluster
A	Number of clusters	3000–7500	2	20–50	150
S	Overlap	5000	2	15	333
Dim	Dimensions	1024	32–1024	16	64
G2	Dimensions + overlap	2048	2–1024	2	1024
Birch	Structure	100,000	2	100	1000
Unbalance	Balance	6500	2	8	100–2000

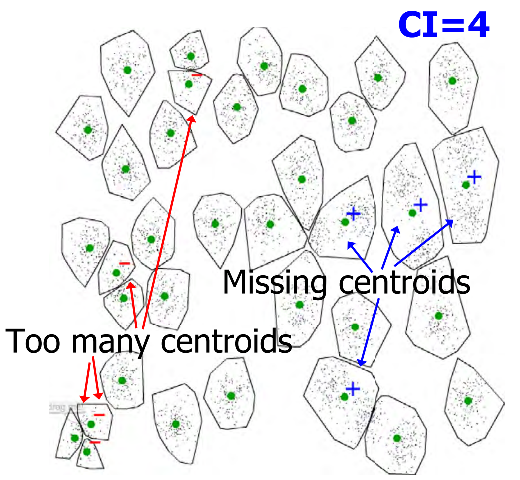


Fig. 3. Performance of k-means with the A2 dataset: CI = 4, SSE = 3.08 $(\cdot 10^{10})$, $\varepsilon = 0.52$.

centroids are wrongly located. Specifically, the value CI = 0 implies that the clustering structure is correct with respect to the ground truth.

An example is shown in Fig. 3, where k-means provides SSE = 3.08×10^{10}, which is 52% higher than that of the ground truth. But what do these numbers really mean? How significant is the difference? On the other hand, the value CI = 4 tells that exactly four real clusters are missing a centroid.

Based on CI, a *success rate* (%) was also defined in [39] to measure the probability of finding the correct clustering. For example, when running k-means 5000 times with dataset A2 (Fig. 3), CI = 0 was never reached, and thus, its success rate is 0%. Another example with dataset S2 (Fig. 4) results in success rate of 1/6 = 17%.

The success rate has an important implication. Any value higher than 0% indicates that the correct clustering can be found simply by repeating k-means. For a success rate p, the expected number of repeats is $1/p$. For instance, $p = 50\%$ indicates that expected number of repeats is 2; and $p = 1\%$ indicates 100 repeats. Even with as low value as $p = 0.1\%$ the correct solution is expected to be found in 1000 repeats. This is time consuming, but feasible. However, for some of our datasets the success rate is so low that the number repeats would be unreasonably high. For example, even 200,000 repeats produces 0% success rate in our experiments with some datasets.

2.3. Properties of k-means

We next briefly summarize the main properties of the k-means algorithm. Generally the clustering problem is the easier the more the clusters are separated. However, in [39] it was found that for k-means it is just the opposite; the less overlap the worse the clustering performance, see Fig. 5. This is a fundamental weakness of the k-means algorithm.

In [39], it was also found that the number of errors has linear dependency on the number of clusters (k). For example, the CI-values for the A sets with k = 20, 35, 50 clusters were measured as CI = 2.5, 4.5, 6.5, respectively. The relative CI-values (CI/k) correspond to a constant of 13% of centroids being wrongly located. Results with the subsets of Birch2 (varying k from 1 to 100) converge to about 16% when k approaches to 100, see Fig. 6.

Two series of datasets are used to study the dimensionality: DIM and G2. The DIM sets have 16 well separated clusters in high-dimensional space with dimensionality varying from $D = 32$ to 1024. Because of clear cluster separation, these datasets should be easy for any good clustering algorithm to reach CI = 0 and 100% success rate. However, k-means again performs poorly; it obtains the values CI = 3.6 and 0% success rate regardless of the dimensionality. The reason for the poor performance is again the lack of cluster overlap, and not the dimensionality.

The results with the G2 sets confirmed the dependency between the dimensionality and the success rate. We allocated four centroids with 3:1 unbalance so that the first cluster had three centroids and the second only one. We then ran k-means and checked whether it found the expected 2:2 allocation by moving one of the three centroids to the second group. The results in Fig. 7 show that the overlap is the mediating factor for the success rate: the more overlap, the lower the success rate of k-means.

The cluster size unbalance was also shown in [39] to result in poor performance. The main reason for this was the random initialization, which cannot pick the initial centroids in a balanced way. Another reason was the k-means iterations which fail to improve the initial solution due to lack of cluster overlap.

The effect of the different properties of data on k-means can be summarized as follows:

Property:	**Effect:**
Cluster overlap	Overlap is good
Number of clusters	Linear dependency
Dimension	No direct effect
Unbalance	Bad

3. K-means initialization techniques

Next we study how much these problems of k-means can be solved by the following two improvements:

- Better initialization
- Repeating k-means

K-means is a good algorithm for local fine-tuning but it has serious limitation to relocate the centroids when the clusters do not overlap. It is therefore unrealistic to expect the clustering problem to be solved simply by inventing a better *initialization* for k-means. The question is merely, how much a better initialization can compensate for the weakness of k-means.

Any clustering algorithm could be used as an initialization technique for k-means. However, solving the location of initial centroids is not significantly easier than the original clustering

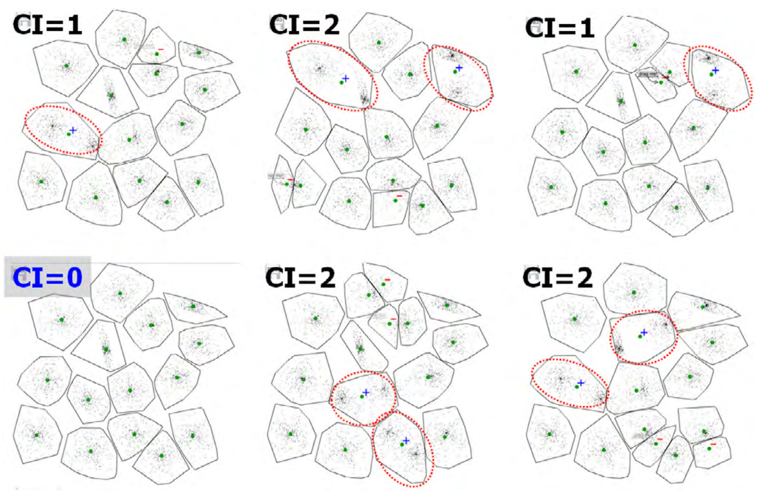


Fig. 4. Centroid index measures how many real clusters are missing a centroid (+), or how many centroids are allocated to wrong cluster (−). Six examples are shown for S2 dataset.

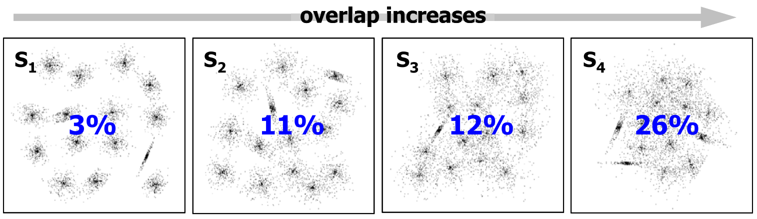


Fig. 5. Success rate (%) of k-means, measured as the probability of finding correct clustering, improves when the cluster overlap increases.

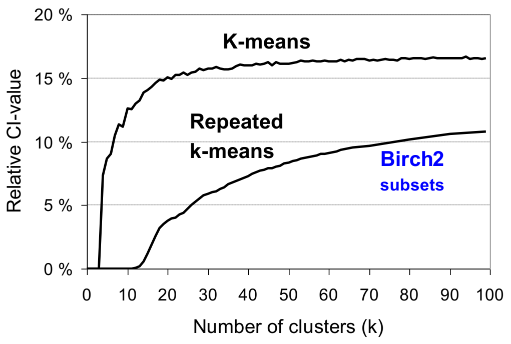


Fig. 6. CI-value of k-means increases linearly with k, and relative CI converges to 16% with the Birch2 subsets.

problem itself. Therefore, for an algorithm to be considered as *initialization* technique for k-means, in contrast to being a standalone algorithm, we set the following requirements:

1. Simple to implement
2. Lower (or equal) time complexity than k-means
3. No additional parameters

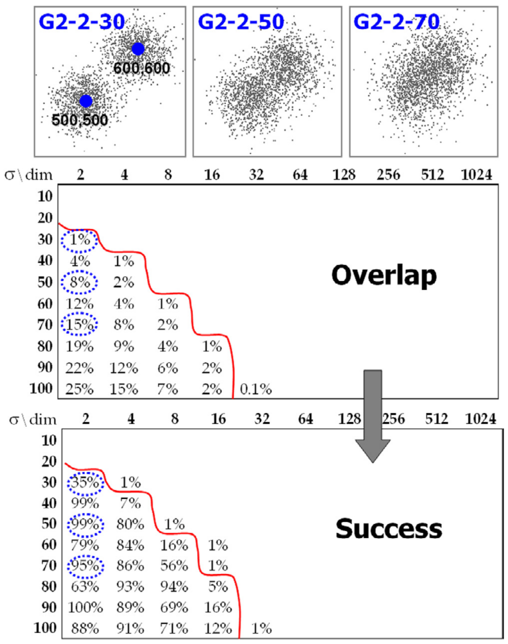
First, the algorithm should be trivial, or at least very easy to implement. Measuring implementation complexity can be subjective. The number of functions and the lines of code were used in [16]. Repeated k-means was counted to have 5 functions and 162 lines of C-code. In comparison, random swap [11,12], fast agglomerative clustering variant [30], and sophisticated splitting algorithm [46] had 7, 12 and 22 functions, and 226, 317 and 947 lines of codes, respectively. Random initialization had 2 functions and 26 lines of code.

Second, the algorithm should have lower or equal time complexity compared to k-means. Celebi et al. [22] categorizes the algorithms to linear, log-linear and quadratic based on their time complexities. Spending quadratic time cannot be justified as the fastest agglomerative algorithms are already working in close to quadratic time [30]. A faster $O(N \log N)$ time variant also exists [47] but it is significantly more complex to implement and requires to calculate *k-near neighbors* (KNN). K-means requires $O(gkN)$ time, where g is the number of iterations and typically varies from 20 to 50.

The third requirement is that the algorithm should be free of parameters; others than k. For instance, there are algorithms [25,48] that select the first centroid using some simple rule, and the rest greedily by cluster growing, based on whether the point is within a given distance. Density-connectivity criterion was also used in [49]. Nevertheless, this approach requires one or more threshold parameters.

Table 2
Summary of the initialization techniques compared in this paper. *Time* refers to the average processing time with the A3 dataset ($N = 7500$, $k = 50$). *Randomized* refers to whether the technique include randomness naturally. Randomness will be needed for the repeated k-means variant later.

Technique	Ref.	Complexity	Time	Randomized	Parameters
Random partitions	[3]	$O(N)$	10 ms	Yes	–
Random centroids	[1,2]	$O(N)$	13 ms	Yes	–
Maxmin	[54]	$O(kN)$	16 ms	Modified	–
kmeans++	[59]	$O(kN)$	19 ms	Yes	–
Bradley	[31]	$O(kN+Rk^2)$	41 ms	Yes	$R = 10$, $s = 10\%$
Sorting heuristic	[62]	$O(N \log N)$	13 ms	Modified	–
Projection-based	[72]	$O(N \log N)$	14 ms	Yes	–
Luxburg	[50]	$O(kN \log k)$	29 ms	Yes	–
Split	[46,68]	$O(N \log N)$	67 ms	Yes	$k = 2$



σ\dim	2	4	8	16	32	64	128	256	512	1024
10										
20										
30	1%									
40	4%	1%								
50	8%	2%								
60	12%	4%	1%							
70	15%	8%	2%							
80	19%	9%	4%	1%						
90	22%	12%	6%	2%						
100	25%	15%	7%	2%	0.1%					

σ\dim	2	4	8	16	32	64	128	256	512	1024
10										
20										
30	35%	1%								
40	99%	7%								
50	99%	80%	1%							
60	79%	84%	16%	1%						
70	95%	86%	56%	1%						
80	63%	93%	94%	5%						
90	100%	89%	69%	16%						
100	88%	91%	71%	12%	1%					

Fig. 7. The effect of overlap for the success of k-means with the G2 datasets. The numbers circled are for the three sample datasets shown above. The dataset names are coded as G2-DIM-SD, where DIM refers to the dimensions and SD to the standard deviation; the higher the SD, the more the two clusters overlap.

The most common heuristics are summarized in Table 2. We categorize them roughly into *random, furthest point, sorting*, and *projection-based* heuristics. Two standalone algorithms are also considered: *Luxburg* [50] and *Split* algorithm. For a good review of several others we refer to [51].

3.1. Random centroids

By far the most common technique is to select k random data objects as the set of initial centroids [1,2]. It guarantees that every cluster includes at least one point. We use *shuffling method* by swapping the position of every data point with another randomly chosen point. This takes $O(N)$ time. After that, we take the first k points from the array. This guarantees that we do not select the same point twice, and that the selection is independent on the order of the data. For the random number generator we use the method in [52]. We refer to this initialization method as *random centroids*.

Slightly different variant in [2] selects simply the first k data points. This is the default option in the Quick Cluster in IBM SPSS Statistics [53]. If the data is in random order the result is effectively the same as random centroids, except that it always provides the same selection.

We note that the randomness is actually a required property for the *repeated* k-means variant. This is because we must be able to produce different solutions at every repeat. Some practitioners might not like the randomness and prefer deterministic algorithms always producing the same result. However, both of these goals can actually be achieved if so wanted. We simply use pseudo-random number generator with the same *seed number*. In this way, single runs of k-means will produce different result but the overall algorithm still produces always the same result for the same input.

3.2. Random partitions

An alternative to random centroids is to generate random partitions. Every point is put into a randomly chosen cluster and their centroids are then calculated. The positive effect is that it avoids selecting outliers from the border areas. The negative effect is that the resulting centroids are concentrated in the central area of the data due to the averaging. According to our observations, the technique works well when the clusters are highly overlapped but performs poorly otherwise, see Fig. 8.

According to [19], the random partition avoids the worst case behavior more often than the random centroids. According to our experiments, this is indeed the case but only when the clusters have high overlap. The behavior of the random partition is also more deterministic than that of random centroids. This is because the centroids are practically always near the center of the data. Unfortunately, this also reduces the benefits of the repeated k-means because there is very little variation in the initial solutions, and therefore, also the final solutions often become identical.

Steinley [29] repeats the initialization 5000 times and selects the one with the smallest SSE. However, repeating only the initialization does not fix the problem. Instead, it merely slows down the initialization because it takes $5000 \cdot N$ steps, which is typically much more than $O(kN)$.

Thiesson et al. [24] calculate the mean point of the data set and then add random vectors to it. This effectively creates initial centroids like a cloud around the center of the data, with very similar effect as the random partition. The size of this cloud is a parameter. If it is set up high enough, the variant becomes similar to the random centroids technique, with the exception that it can select points also from empty areas.

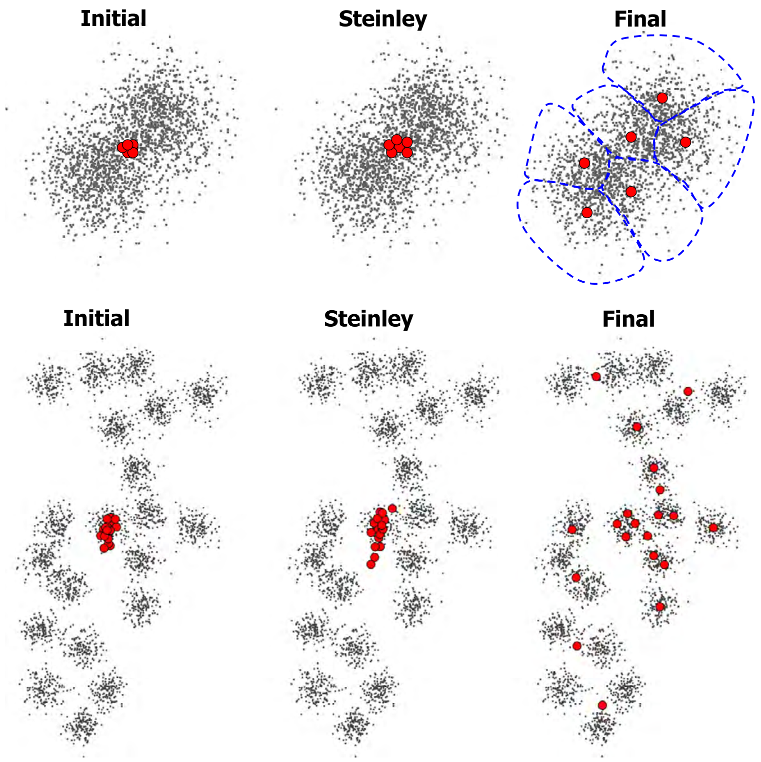


Fig. 8. Initial centroids created by random partition (left), by Steinley's variant (middle), and the final result after the k-means iterations (right).

Fig. 8 shows the effect of the random partition and Steinley's variant. Both variants locate the initial centroids near the center of the data. If the clusters have low overlap, k-means cannot provide enough movement and many of the far away clusters will lack centroids in the final solution.

3.3. Furthest point heuristic (Maxmin)

Another popular technique is the furthest point heuristic [54]. It was originally presented as standalone 2-approximate clustering algorithm but has been widely used to initialize k-means. It selects an arbitrary point as the first centroid and then adds new centroids one by one. At each step, the next centroid is the point that is furthest (max) from its nearest (min) existing centroid. This is also known as *Maxmin* [19,21,22,55].

Straightforward implementation requires O(k^2N) time but it can be easily reduced to O(kN) as follows. For each point, we maintain pointer to its nearest centroid. When adding a new centroid, we calculate the distance of every point to this new centroid. If the new distance is smaller than to the previous nearest, then it is updated. This requires N distance calculations. The process is repeated k times, and the time complexity is therefore O(kN) in total, which is the same as one iteration of k-means. Further speedup can be achived by searching for the furthest point in just a subset of the data [56].

There are several alternative ways to choose the first centroid. In the original variant the selection is arbitrary [54]. In [55], the furthest pair of points are chosen as the first two centroids. Another variant selects the one with maximum distance to the origin [57] because it is likely to be located far from the center. Maximum density has also been used [51,58].

K-means++ [59] is a randomized variant of the furthest point heuristic. It chooses the first centroid randomly and the next ones using a weighted probability $p_i = cost_i/\mathrm{SUM}(cost_i)$, where $cost_i$ is the squared distance of the data point x_i to its nearest centroids. This algorithm is an O(log k)-approximation to the problem. We also implement k-means++ for our tests because of its popularity.

Chiang and Mirkin [55] recalculate all the centroids after updating the partitions, and the next centroid is selected as the farthest from the recently added centroid. Slightly more complex variant [23] selects the point that decreases the objective function most. It requires calculation of all distances between every pair of points, which takes O(N^2) time. Thus, it does not qualify our criteria for k-means initialization. With the same amount of computation we can already run implement agglomerative clustering algorithm.

Other authors also weight the distances by the density of the point [51,58]. This reduces the probability that outliers are selected. Erisoglu et al. [60] use cumulative distance to all previous centroids instead of the maxmin criterion. However, this performs worse because it can easily choose two nearby points provided that they have large cumulative distance to all other centroids [61].

We use here a variant that selects the first point randomly [54,59]. This adds randomness to the process as required by the repeated k-means variant. The next centroids we select using the original maxmin criterion, i.e. choosing the point with biggest distance to its nearest centroid.

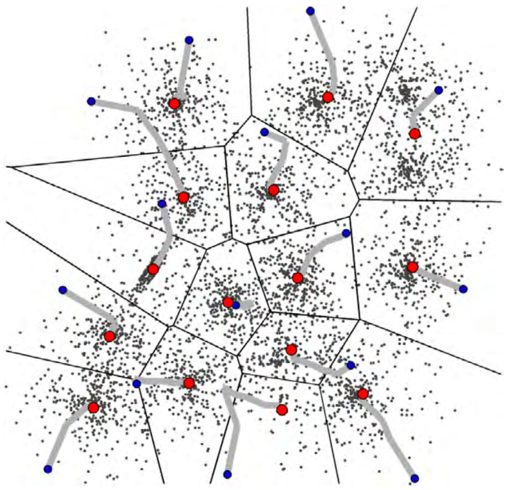


Fig. 9. Example of the maxmin heuristic for *S3* dataset. The blue dots are the initial and the red dots the final centroids. The trajectories show their movement during the k-means iterations. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Maxmin technique helps to avoid worst case behavior of the random centroids, especially when the cluster sizes have serious unbalance. It also has tendency to pick up outlier points from the border areas, which leads to slightly inferior performance in the case of datasets with high overlap (S3 and S4). However, k-means usually works better with such datasets [39], which compensates for the weakness of Maxmin. Fig. 9 demonstrates the performance of the Maxmin technique.

3.4. Sorting heuristics

Another popular technique is to sort the data points according to some criterion. Sorting requires O(N log N) time, which is less than that of one k-means iteration, O(kN), assuming that log$N \le k$. After sorting, k points are selected from the sorted list using one of the following heuristics:

- First k points.
- First k points while disallowing points closer than ε to already chosen centroids.
- Every (N/k)th point (uniform partition)

For the sorting, at least the following criteria have been considered:

- Distance to center point [62]
- Density [21,63]
- Centrality [64]
- Attribute with the greatest variance [65]

Hartigan and Wong [62] sort the data points according to their distance to the center of the data. The centroids are then selected as every N/kth point in this order. We include this variant in our tests. To have randomness, we choose a random data point as a reference point instead of the center. This heuristic fulfills our requirements: it is fast, simple, and requires no additional parameters.

Astrahan [63] calculates *density* as the number of other points within a distance d_1. First centroid is the point with the highest density, and the remaining k-1 centroids are chosen at a decreasing order, with the condition that they are not closer than distance d_2 from an already chosen centroid. Steinley and Brusco [21] recommends using the average pairwise distance (pd) both for d_1 and d_2. This makes the technique free from parameters but it is still slow, O(N^2) time, for calculating the pairwise distances.

It would be possible to simplify this technique further and use random sampling: select N pairs of points, and use this subsample to estimate the value of pd. However, the calculation of the densities is still the bottleneck, which prevents this approach from meeting the requirements for k-means initialization as such.

Cao et al. [64] proposed a similar approach. They use a primary criterion (*cohesion*) to estimate how central a point is (how far from boundary). Secondary threshold criterion (*coupling*) is used to prevent centroids from being neighbors.

Al-Daoud [65] sorts the data points according to the dimension with the largest variance. The points are then partitioned into k equal size clusters. Median of each cluster is selected instead of the mean. This approach belongs to a more general class of projection-based techniques where the objects are mapped to some linear axis such as diagonal or principal axis.

The sorting heuristic would work if the clusters were well separated, and all have different criterion value (such as the distance from center point). This actually happens with the very high dimensional DIM datasets in our benchmark. However, with most other datasets the clusters tend to be randomly located in respect to the center point, and it is unlikely that all the clusters would have different criterion values. What happens in practice, is that the selected centroids are just random data points in the space, see Fig. 10.

3.5. Projection-based heuristics

Sorting heuristics can also be seen as a projection of the points into a one-dimensional (non-linear) curve in the space. Most criteria would just produce an arbitrary curve connecting the points randomly, and lacking convexity or any sensible shape. However, several linear projection-based techniques have been considered in the literature:

- Diagonal axis [65]
- Single axis [66,67]
- Principal axis [46,67–71]
- Two random points [72]
- Furthest points [72]

After the projection is performed, the points are partitioned into k equal size clusters similarly as with the sorting-based heuristics.

Yedla et al. [66] sort the points according to their distance to origin, and then select every N/kth point. If the origin is the center of data, this is essentially the same technique as in [62]. If the attributes are non-negative, then this is essentially the same as projecting the data to the diagonal axis. Such projection is trivial to implement by calculating the average of the attribute values. It has also been used for speeding-up nearest neighbor searches in clustering in [73].

Al-Daoud [65] sorts the points according to the dimension with the largest variance. The points are then partitioned into k equal size clusters. Median of each cluster is selected instead of the mean. This adapts to the data slightly better than just using the diagonal.

A more common approach is to use *principal axis*, which is the axis of projection that maximizes variance. It has been used effectively in divisive clustering algorithms [46,67–71]. Calculation of the principal axis takes O(DN)-O(D^2N) depending on the variant

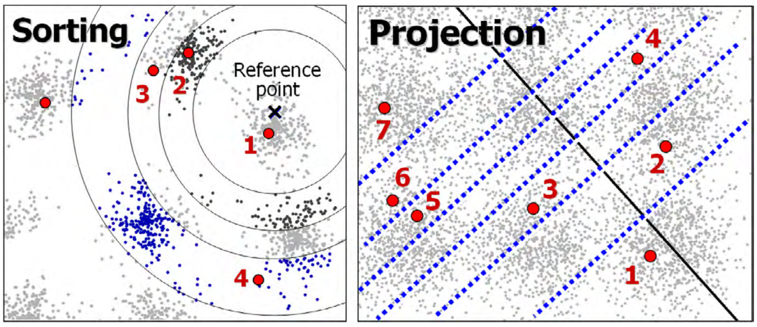


Fig. 10. Examples of sorting and projection-based techniques.

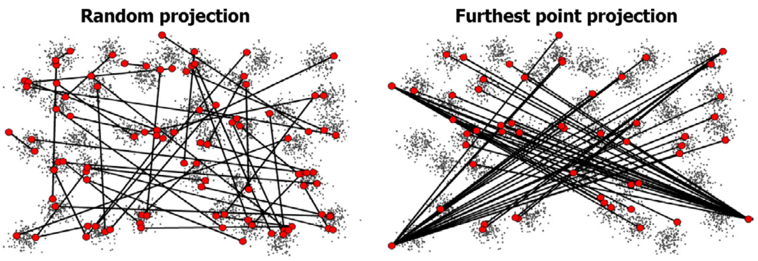


Fig. 11. Examples of the two projection-based heuristics for A2 dataset: random points (left), and the furthest point projections (right) [72].

[46]. A more complex *principal curve* has also been used for clustering [74].

We consider two simple variants: *random* and *two furthest points* projection as studied in [72]. The first heuristic takes two random data points and projects to the line passing by these two reference points. The key idea is the randomness; single selection may provide poor initialization but when repeating several times, the chances to find one good initialization increases, see Fig. 11. We include this technique into our experiments and refer to it as *Projection*.

The second heuristic is slightly more deterministic but still random. We start by selecting a random point, and calculate its furthest point. The projection axis is the line passing by these two reference points. We again rely on randomness, but now the choices are expected to be more sensible, potentially providing better results using fewer trials. However, according to [72] this variant does not perform any better than the simpler random heuristic.

Projection works well if the data has one-dimensional structure. In [72], *projective value* is calculated to estimate how well a given projection axis models the data. From our data, Birch2 and G2 have high projective values and suitable for projection-based technique. However, with all other datasets, the projection does not make much more sense than the naïve sorting heuristics, see Fig. 10.

We also note that projection-based techniques also generalize to segmentation-based clustering, where k-1 dividing planes are searched simultaneously using dynamic programming [74,75]. These clustering results usually require fine-tuning by k-means at the final step, but nevertheless, they are standalone algorithms.

3.6. Density-based heuristics

Density was already used both with the furthest point and the sorting heuristics, but the concept deserves a little bit further discussion. The idea of using density itself is appealing but it is not trivial how to calculate the density, and how to use it in clustering. Especially since the initialization technique should be fast and simple.

The main bottleneck of the algorithms is how to calculate the density is estimated for the points. There are three common approaches for this:

- Buckets
- ε-radius circle
- k-nearest neighbors (KNN)

The first approach divides the space by a regular grid, and counts the frequency of the points in every bucket [76]. The density of a point is then inherited from the bucket it is in. This approach is feasible in low-dimensional space but would become impractical in higher-dimensional spaces. In [61], the problem is addressed by processing the dimensions independently in a heuristic manner. Other authors have used *kd-tree* [51,57] or space-filling curve [77] to partition the space into buckets containing roughly the same number of points. In [51,57], the number of buckets is $10 \cdot k$.

The other two approaches calculate the density for every point individually. The traditional one is to define a neighborhood using a cutoff threshold (ε-radius), and then counting the number of other points within this neighborhood [21,63,64,78]. The third approach finds the *k-nearest neighbors* of a point [79], and then calculates the average distance to the points within this neighborhood. Lemke and Keller calculate the density between every pair of points [49].

The bottleneck of the last two approaches is that we need to find the points that are within the neighborhood. This requires O(N^2) distance calculations in both cases. Several speed-up techniques and approximate variants exist [80,81] but none that is both fast and simple to implement. Calculating density values only for a subset of size SQRT(N) would reduce the complexity to O($N^{1.5}$)

depending whether the distances are calculated to all points or only within the subset. In [82], density is calculated in each dimension separately, and then final approximation is obtained by summing up the individual densities. This allows rapid O(DN) time estimation with more accurate estimation than the sub-sampling approach.

Once calculated, the density can be used jointly with the furthest point heuristic, with the sorting heuristic, or some of their combination. For example, in [51] the furthest point heuristic was modified by weighting the distance by its density so that outliers are less likely chosen. The *density peaks* algorithm in [78] finds for every point its nearest neighbor with higher density. It then applies sorting heuristic based on one of the two features: density and the distance to its neighbor. The method works as a standalone algorithm and does not require k-means at all.

Luxburg [50] first selects k*SQRT(k) preliminary clusters using k-means and then eliminates the smallest ones. After this, the furthest point heuristic is used to select the $k$clusters from the preliminary set of clusters. When minimizing SSE, the size of the clusters correlates to their density. Thus, Luxburg's technique indirectly implements a density-based approach which favors clusters of high density. We include this technique in our experiments although it does not satisfy our simplicity criterion.

We also note that there are several standalone clustering algorithms based on density [49,78,83,84]. However, they do not fit to our requirements for speed and simplicity. If combined with the faster density estimation in [82], some of these techniques could be made competitive also in speed.

3.7. Splitting algorithm

Split algorithm puts all points into a single cluster, and then iteratively splits one cluster at a time until k clusters are reached. This approach is seemingly simple and tempting to consider for initializing k-means. However, there are two non-trivial design choices to make: which cluster to split, and how to split it. We therefore consider split mainly as a standalone algorithm, but discuss briefly some existing techniques that have been used within k-means.

Linde et al. [85] uses binary split for initialization of their *LBG algorithm* in the vector quantization context. Every cluster is split by replacing the original centroid c by $c+\varepsilon$ and c-ε, where ε refers to a random vector. Splitting every cluster avoids the question of which cluster to split but it does not have any real speed benefit. In [46], ε was calculated as the standard deviation of the points in the cluster, in each dimension separately.

Projection-based approaches are also suitable for the splitting algorithm. The idea is to divide a chosen cluster according to a hyperplane perpendicular to the projection axis. It is possible to find the optimal choice of the cluster to be split, and the optimal location of the hyperplane in O(N) time [46,68]. This results in a fast, O(N·log$N$·log$k$) time algorithm, but the implementation is quite complex. It requires 22 functions and 947 lines of codes, compared to 5 functions and 162 lines in repeated k-means [16].

There is also a split-kmeans variant that applies k-means iteration after every split in [46], later popularized under the name *Bisecting k-means* in document clustering [86]. However, this would increase the time complexity to O(k^2N), which equals to O(N^2) if $k \approx$ SQRT(N). Tri-level k-means [87] performs the clustering in two stages. It first creates less clusters than k, and then splits the clusters with highest variation before applying the traditional k-means. All these variants are definitely standalone algorithms, and do not qualify as an initialization technique here.

In this paper, we therefore implement a simpler variant. We always select the biggest cluster to be split. The split is done by selecting two random points in the cluster. K-means is then applied but only within the cluster that was split as done in [68]. The main difference to the bisecting k-means [86] and its original split+kmeans variant in [46], is that the time complexity sums up to only O(N·log$N$); a proof can be easily derived from the one in [46].

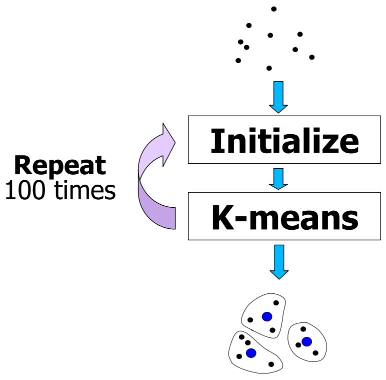


Fig. 12. General principle of repeated k-means (RKM). The key idea is that the initialization includes randomness to produce different solutions at every repeat.

3.8. Repeated k-means

Repeated k-means performs k-means multiple times starting with different initialization, and then keeping the result with lowest SSE-value. This is sometimes referred as *multi-start k-means*. The basic idea of the repeats is to increase the probability of success. Repeated k-means can be formulated as a probabilistic algorithm as follows. If we know that k-means with a certain initialization technique will succeed with a probability of p, the expected number of repeats (R) to find the correct clustering would be:

$$R = 1/p$$

In other words, it is enough that k-means succeeds even sometimes ($p > 0$). It is then merely a question of how many repeats are needed. Only if $p \approx 0$ the number of repeats can be unrealistically high. For example, standard k-means with random centroids succeeds 6–26% of the time with the S1-S4 datasets. These corresponds to $R = 7$ to 14 repeats, on average.

If the initialization technique is deterministic (no randomness), then it either succeeds ($p = 100\%$) or fails ($p = 0\%$) every time. To justify the repeats, a basic requirement is that there is some randomness in the initialization so that the different runs produce different results. Most techniques have the randomness implicitly. The rest of the techniques we modify as follows:

• Rand-P	Already included
• Rand-C	Already included
• Maxmin	First centroid randomly
• Kmeans++	Already included
• Bradley	Already included
• Sorting	Reference point randomly
• Projection	Reference points randomly
• Luxburg	Already included
• Split	Split centroids randomly

Repeats add one new parameter R. Since p is not known in practice, we cannot derive value for R automatically. In this paper, we use $R = 100$ unless otherwise noted. Fig. 12 shows the overall scheme of the repeated k-means.

Repeating k-means also multiplies the processing time by a factor of R. It is possible to compensate for this by dividing the data into random subsets. For instance, if we divide the data into R subsets of size N/R, the total processing time would be roughly the same as that of a single run.

For example, Bradley and Fayyad [31] apply k-means for a sub-sample of size N/R, where $R = 10$ was recommended. Each sample is clustered by k-means starting with random centroids. However, instead of taking the best clustering of the repeats, a new dataset is created from the $R{\cdot}k$ centroids. This new dataset is then clustered by repeated k-means (R repeats). The total time complexity is $R{\cdot}k{\cdot}(N/R) + R{\cdot}k^2 = kN + Rk^2$, where the first part comes from clustering the sub-samples, and the second part from clustering the combined set. If $k = \text{SQRT}(N)$, then this would be $N^{1.5} + RN$. Overall, the algorithm is fast and satisfies the criteria for initialization technique.

Bahmani et al. [88] have a similar approach. They repeat k-means++ $R = O(\log N)$ times to obtain $R{\cdot}k$ preliminary centroids, which are then used as a new dataset for clustering by standard k-means. They reported that $R = 5$ would be sufficient for the number of repeats. In our experiments, we consider the Bradley and Fayyad [31] as an initialization, and use $R = 100$ repeats as with all techniques.

4. Experimental results

We study next the overall performance of different initialization techniques, and how the results depend on the following factors:

- Overlap of clusters
- Number of clusters
- Dimensions
- Unbalance of cluster sizes

The overall results (CI-values and success rates) are summarized in Table 3. We also record (as *fails*) how many datasets provide success rate $p = 0\%$. This means that the algorithm cannot find the correct clustering even with 5000 repeats. We test the following methods:

- Rand-P
- Rand-C
- Maxmin
- kmeans++
- Bradley
- Sorting
- Projection
- Luxburg
- Split

4.1. Overall results

CI-values: Random partition works clearly worse (CI = 12.4) than the random centroids (CI = 4.5). Bradley and sorting heuristics are slightly better (CI = 3.1 and 3.3), but the maxmin heuristics (Maxmin and kmeans++) are the best among the true initialization techniques (CI = 2.2 and 2.3). The standalone algorithms (Luxburg and Split) are better (CI = 1.2 and 1.2), but even they provide the correct result (CI = 0) only for the easiest dataset: DIM32.

Success rates: The results show that Maxmin is a reasonable heuristic. Its average success rate is 22% compared to 5% of random centroids. It also fails (success rate = 0%) only in case of three datasets; the datasets with a high number of clusters (A3, Birch1, Birch2). Random partition works with S2, S3 and S4 but fails with all the other 8 datasets. The standalone algorithms (Luxburg and Split) provide 40% success rates, on average, and fail only with Birch1 and Unbalance.

Effect of iterations: From the initial results we can see that Luxburg and Bradley are already standalone algorithms for which k-means brings only little improvement. The average CI-value of Luxburg improves only from 1.7 to 1.2 (~30%), and Bradley from 3.4 to 3.1 (~10%). The latter is more understandable as k-means is already involved in the iterations. Split heuristic, although a standalone algorithm, leaves more space for k-means to improve (61%).

Number of iterations: The main observation is that the easier the dataset, and the better the initialization, the fewer the iterations needed. The differences between the initialization vary from 20 (Luxburg) to 36 (Rand-C); with the exception of random partition (Rand-P), which takes 65 iterations.

4.2. Cluster overlap

The results with the S1–S4 datasets (Table 3) demonstrate the effect of the overlap in general: the less overlap, the worse the k-means' performance. Some initialization techniques can compensate for this weakness. For example, the maxmin variants and the standalone algorithms reduce this phenomenon but do not remove it completely. They provide better initial solution with S1 (less overlap) than with S4 (more overlap), but the final result after the k-means iterations is still not much different. An extreme case is DIM32, for which all these better techniques provide correct solution. However, they do it even without k-means iterations!

Further tests with G2 confirm the observation, see Fig. 13. When overlap is less than 2%, the k-means iterations do not help much and the result depends mostly on the initialization. If the correct clustering is found, it is found without k-means. Thus, the clustering is solved by a better algorithm, not by better k-means initialization. In case of high overlap, k-means reaches almost the same result (about 88% success rate) regardless of how it was initialized.

4.3. Number of clusters

The results with the A1–A3 datasets (Table 3) show that the more there are clusters the higher the CI-value and the lower the success rate. This phenomenon holds for all initialization techniques and it is not specific to k-means algorithm only. If an algorithm provides correct clustering with success rate p for a dataset of size k, then p is expected to decrease when k increases. Fig. 14 confirms this dependency with the Birch2 subsets. Projection heuristic is the only technique that manages to capture the hidden 1-dimensional structure in this data. The success rate of all other true initialization techniques eventually decreases to 0%.

Fig. 15 shows that the CI-value has a near linear dependency on the number of clusters. In most cases, the *relative* CI-value converges to a constant when k approaches its maximum ($k = 100$). An exception is Luxburg, which is less sensitive to the increase of k; providing values CI = (0.82, 1.25, 1.42, 1.54) for $k = (25, 50, 75, 100)$. Besides this exception, we conclude that the performance has linear dependency on k regardless of the initialization technique.

4.4. Dimensions

We tested the effect of dimensions using the DIM and G2 datasets. Two variants (Maxmin, Split) solve the DIM sets almost every time (99–100%), whereas Kmeans++ and Luxburg solve them most of the times (≈95%), see Fig. 16. Interestingly, they find the correct result by the initialization and no k-means iterations are needed. In general, if the initialization technique is able to solve the clustering, it does it regardless of the dimensionality.

The sorting and projection heuristics are exceptions in this sense; their performance actually improves with the highest dimensions. The reason is that when the dimensions increase, the clusters eventually become so clearly separated that even such

Table 3
Average CI-values before and after k-means iterations, success rates, and the number of iterations performed. The results are averages of 5000 runs. *Fail* records for how many datasets the correct solution was never found (success rate = 0%). From DIM datasets we report only DIM32; the results for the others are practically the same. Note: The values for Impr. and Aver. columns are calculated from precise values and not from the shown rounded values. (For interpretation of the references to color in the Table the reader is referred to the web version of this article.)

CI-values (initial)

Method	s1	s2	s3	s4	a1	a2	a3	unb	b1	b2	dim32	Aver.
Rand-P	12.5	14.0	12.8	14.0	19.0	32.9	48.1	7.0	96.0	96.6	13.1	**33.3**
Rand-C	5.3	5.5	5.4	5.4	7.3	12.7	18.2	4.6	36.6	36.6	5.8	**13.0**
Maxmin	1.3	2.9	6.1	6.8	2.1	4.1	5.0	0.9	21.4	9.6	0.0	**5.5**
kmeans++	1.7	2.3	3.2	3.3	3.1	5.6	7.9	0.8	21.3	10.4	0.1	**5.4**
Bradley	1.0	0.7	0.6	0.5	1.5	3.4	5.3	3.3	5.7	13.6	1.7	**3.4**
Sorting	3.3	3.7	4.1	4.4	4.9	10.4	15.6	4.0	34.1	7.2	1.7	**8.5**
Projection	3.0	3.4	3.9	4.2	4.5	9.8	15.2	4.0	33.7	1.0	1.1	**7.6**
Luxburg	0.8	0.8	1.1	1.3	0.9	1.1	1.2	4.2	5.6	1.7	0.0	**1.7**
Split	0.5	0.8	1.4	1.4	1.3	2.4	3.5	4.5	12.0	2.7	0.0	**2.8**

CI-values (final)

Method	s1	s2	s3	s4	a1	a2	a3	unb	b1	b2	dim32	Aver.	Impr.
Rand-P	3.3	0.6	1.2	0.4	6.0	10.7	17.9	4.0	11.3	75.6	5.3	**12.4**	**63%**
Rand-C	1.8	1.4	1.3	0.9	2.5	4.5	6.6	3.9	6.6	16.6	3.6	**4.5**	**65%**
Maxmin	0.7	1.0	0.7	1.0	1.0	2.6	2.9	0.9	5.5	7.3	0.0	**2.2**	**62%**
kmeans++	1.0	0.9	1.0	0.8	1.5	2.9	4.2	0.5	4.9	7.2	0.1	**2.3**	**57%**
Bradley	0.9	0.6	0.5	0.4	1.3	3.0	4.8	3.5	4.6	12.5	1.6	**3.1**	**11%**
Sorting	1.3	1.1	1.0	0.7	1.5	3.6	5.5	4.0	5.7	4.3	1.4	**2.7**	**69%**
Projection	1.2	0.9	0.8	0.6	1.2	3.3	5.2	4.0	5.3	0.2	0.9	**2.2**	**71%**
Luxburg	0.5	0.4	0.6	0.4	0.6	0.9	1.0	4.0	2.7	1.6	0.0	**1.2**	**29%**
Split	0.2	0.3	0.4	0.4	0.5	1.1	1.8	4.0	2.8	1.6	0.0	**1.2**	**61%**

Success-%

Method	s1	s2	s3	s4	a1	a2	a3	unb	b1	b2	dim32	Aver.	Fails
Rand-P	0%	47%	5%	63%	0%	0%	0%	0%	0%	0%	0%	**10%**	**8**
Rand-C	3%	11%	12%	26%	1%	0%	0%	0%	0%	0%	0%	**5%**	**6**
Maxmin	37%	16%	36%	9%	15%	1%	0%	22%	0%	0%	100%	**22%**	**3**
kmeans++	21%	24%	18%	30%	7%	0%	0%	51%	0%	0%	88%	**22%**	**4**
Bradley	21%	46%	49%	64%	7%	0%	0%	0%	0%	0%	2%	**17%**	**5**
Sorting	12%	20%	22%	36%	10%	0%	0%	0%	0%	12%	15%	**12%**	**4**
Projection	16%	29%	30%	42%	18%	0%	0%	0%	0%	92%	34%	**24%**	**4**
Luxburg	52%	60%	45%	61%	45%	33%	31%	0%	0%	17%	95%	**40%**	**2**
Split	78%	75%	62%	64%	51%	17%	5%	0%	0%	10%	99%	**42%**	**2**

Number of iterations

Method	s1	s2	s3	s4	a1	a2	a3	unb	b1	b2	dim32	Aver.
Rand-P	32	37	37	39	43	58	76	36	228	130	3	**65**
Rand-C	20	24	27	40	22	26	27	33	117	48	5	**36**
Maxmin	13	19	24	37	20	18	20	4	92	43	2	**26**
kmeans++	14	19	24	35	17	20	22	13	89	43	2	**27**
Bradley	13	12	13	17	12	17	19	24	77	45	2	**23**
Sorting	17	21	25	37	19	24	26	38	104	33	3	**32**
Projection	15	20	25	35	17	24	25	36	99	6	3	**28**
Luxburg	9	12	17	27	11	12	12	33	62	23	2	**20**
Split	7	11	19	27	12	16	18	35	65	27	2	**22**

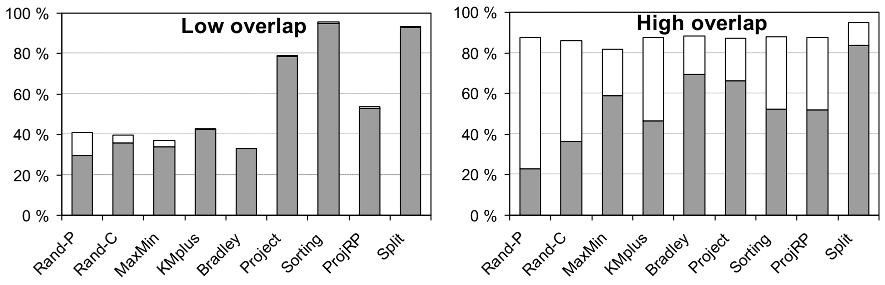


Fig. 13. Average success rates for all G2 datasets before (gray) and after k-means (white). The datasets were divided into two categories: those with low overlap <2% (left), and those with high overlap ≥2% (right).

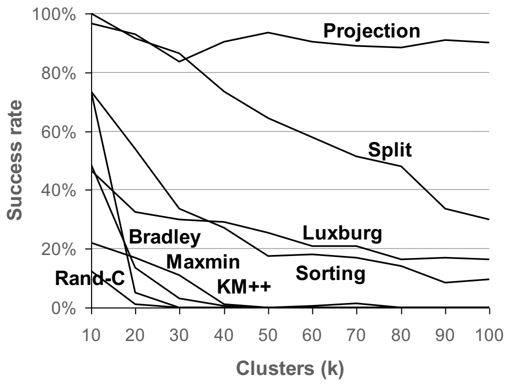


Fig. 14. Dependency of the success rate and the number of clusters when using the subsets of Birch2 (*B2-sub*).

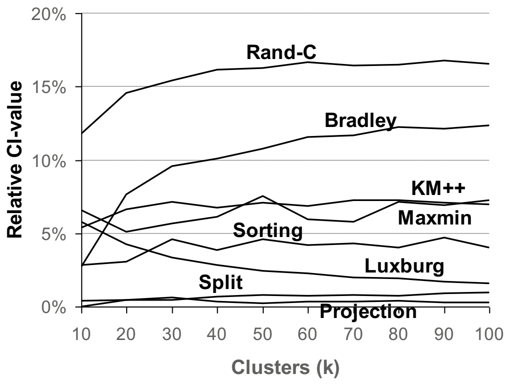


Fig. 15. Dependency of the *relative* CI-values (CI/k) and the number of clusters when using the subsets of Birch2 (*B2-sub*).

naïve heuristics will be able to cluster the data. In general, the reason for success or failure is not the dimensionality but the cluster separation.

The results with G2 confirm the above observation, see Fig. 16. With the lowest dimensions, k-means iterations work because some cluster overlap exists. However, for higher dimensions the overlap eventually disappears and the performance starts to depend mainly on the initialization. We also calculated how much the success rate correlates with the dimensions and the overlap. The results in Table 4 show that the final result correlates much stronger with the overlap than with the dimensionality.

Since there is causality between dimensions and overlap, it is unclear whether the dimensionality has any role at all. To test this further, we generated additional datasets with $D = 2$–16 and compared only those with overlap = 2%, 4%, 8%. The results showed that success of the k-means iterations do not depend on the dimensions even when the clusters overlap.

To sum up, our conclusion is that k-means iterations cannot solve the problem when the clusters are well separated. All techniques that solve these datasets, do it already by the initialization technique without any help of k-means. When there is overlap, k-means works better. But even then, the performance does not depend on the dimensionality.

4.5. Unbalance

Unbalance dataset shows one weakness of k-means. The problem is not the different densities as such, but the unbalance of cluster sizes together with the separation of the clusters. If no centroids are selected from the sparse area, k-means iterations manage to move only one centroid into this area, and all other centroids will remain in the dense area, see Fig. 17. The probability that a single random centroid would be selected from the sparse area is $p = 500/6500 = 7\%$. To pick all required five centroids from the sparse area would happen with probability of 0.01%,[1] i.e. only once every 8430 runs.

Besides Rand-C and Rand-P, sorting and projection heuristics, Luxburg and Split algorithms all fail with this data by allocating most centroids to the dense area. Bradley works only slightly better and often allocates two centroids to the sparse area. Maxmin heuristics work best because they rely more on distances than on frequencies. K-means++ typically misses one centroid whereas Maxmin does the opposite and allocates one too many centroids in the sparse area. They provide success rates of 22% (Maxmin) and 51% (KM++), in contrast to the other techniques that result in 0% success.

To sum up, success depends mainly on the goodness of the initialization; k-means iterations can do very little with this kind of data. If the correct clustering is found, it is found mainly without k-means.

4.6. Repeats

We next investigate to what extent the k-means performance can be improved by repeating the algorithm several times. Table 5 summarizes the results. We can see that significant improvement is achieved with all initialization techniques. When the success rate of a single run of k-means is 2% or higher, CI = 0 can always be reached thanks to the repeats. However, none of the variants can solve all datasets. Overall performance of the different initialization techniques can be summarized as follows:

- Random partition is almost hopeless and the repeats do not help much. It only works when the clusters have strong overlap. But even then, k-means works relatively well anyway regardless of the initialization.
- Random centroids is improved from CI = 4.5 to 2.1, on average, but still it can solve only three datasets (S2, S3, S4). Two other datasets (S1, A1) could be solved with significantly more repeats, but not the rest.
- Maxmin variants are the best among the simple initialization techniques providing CI = 0.7, on average, compared to 2.1 of Rand-C. They still fail with four datasets. K-means++ is not significantly better than the simpler Maxmin.
- The standalone algorithms (Luxburg and Split) are the best. They provide average value of CI = 1.2 without the repeats, and CI = 0.4 with 100 repeats. They fail only with the Unbalance datasets.

The improvement from the repeats is achieved at the cost of increased processing time. We used the fast k-means variant [89] that utilizes the activity of the centroids. For the smaller data sets the results are close to real-time, but with the largest dataset (Birch1, $N = 100{,}000$), the 100 repeats can take from 10–30 min.

We extended the tests and ran 200,000 repeats for A3 and Unbalance datasets. The results in Table 6 show that Maxmin would need 216 repeats to reach CI = 0 with A3, on average, whereas k-means++ would require 8696 repeats even though it finds CI = 1

[1] $\binom{8}{5} p^5 (1-p)^3$.

Table 4
Correlation of success rate with increasing overlap (left) and dimensions (right) with the G2 datasets (3:3 centroid allocation test). Red>0.60, Yellow = 0.30–0.53.

	Overlap		Dimension	
	Init	Final	Init	Final
Rand-P	-0.34	0.68	0.11	-0.46
Rand-C	0.08	0.82	0.13	-0.35
Maxmin	0.61	0.73	-0.23	-0.32
kmeans++	0.63	0.80	-0.39	-0.41
Bradley	0.67	0.71	-0.48	-0.47
Sorting	0.46	0.69	-0.53	-0.51
Projection	0.02	0.45	0.01	-0.27
Luxburg	0.12	0.80	-0.32	-0.38
Split	-0.61	0.06	-0.39	-0.79

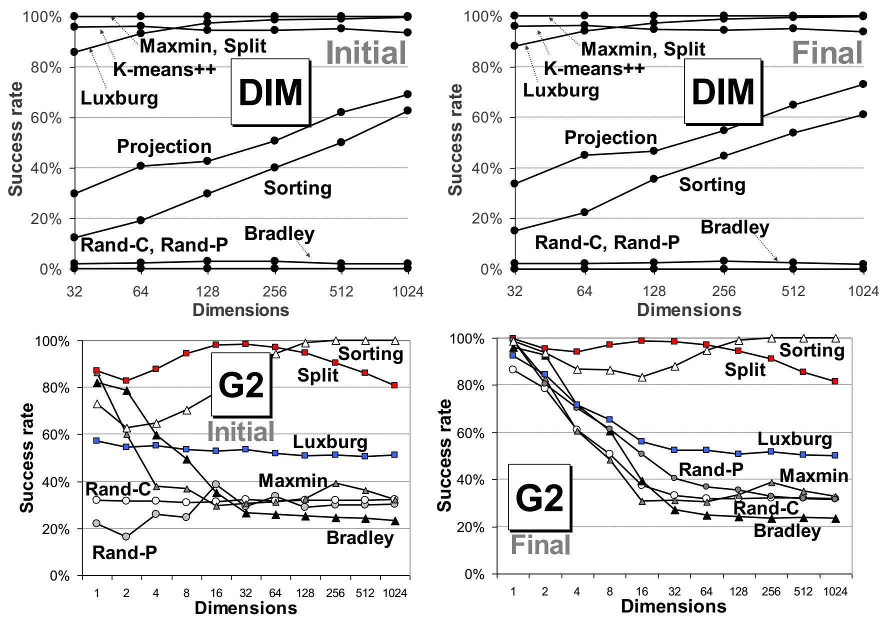


Fig. 16. Dependency of success rate on the dimensions when no overlap (DIM sets), and with overlap (G2 datasets). The results of G2 are average success rates for all sd = 10–100 (G2-D-sd) with a given dimension D, before and after k-means.

already after 138 repeats. The results also show that Unbalance dataset is difficult for almost all initialization techniques but the maxmin heuristics are most suitable for this type of data.

4.7. Summary

We make the following observations:

- Random partition provides an initial solution of similar quality regardless of overlap, but the errors in initial solution can be better fixed by k-means iterations when clusters have high overlap. In this case it can even outperform random centroids. However, repeats do not improve the results much, especially with sets having many clusters (A3, Birch2).
- Cluster overlap is the biggest factor. If there is high overlap, k-means iterations work well regardless of the initialization. If there is no overlap, then the success depends completely on the initialization technique: if it fails, k-means will also fail.
- Practically all initialization techniques perform worse when the number of clusters increases. Success of the k-means depends linearly on the number of clusters. The more clusters, the more errors there are, before and after the iterations.
- Dimensionality does not have a direct effect. It has a slight effect on some initialization techniques but k-means iterations are basically independent on the dimensions.
- Unbalance of cluster sizes can be problematic especially for the random initializations but also for the other techniques. Only

Table 5
Performance of the repeated k-means (100 repeats). The last two columns show the average results of all datasets without repeats (KM) and with repeats (RKM). (For interpretation of the references to color in the Table the reader is referred to the web version of this article.)

CI-values

Method	s1	s2	s3	s4	a1	a2	a3	unb	b1	b2	dim32	KM	RKM
Rand-P	1.4	0.0	0.0	0.0	4.9	8.8	16.7	3.6	8.5	74.0	2.6	12.4	11.0
Rand-C	0.1	0.0	0.0	0.0	0.3	1.8	2.9	2.9	2.8	10.9	1.1	4.5	2.1
Maxmin	0.0	0.0	0.0	0.0	0.0	0.5	0.6	0.0	2.8	3.9	0.0	2.2	0.7
kmeans++	0.0	0.0	0.0	0.0	0.0	0.8	1.6	0.0	1.7	3.4	0.0	2.3	0.7
Bradley	0.0	0.0	0.0	0.0	0.0	0.9	2.1	1.2	2.0	8.5	0.0	3.1	1.3
Sorting	0.0	0.0	0.0	0.0	0.0	0.8	2.2	4.0	2.2	0.0	0.0	2.7	0.8
Projection	0.0	0.0	0.0	0.0	0.0	0.9	2.0	3.9	1.9	0.0	0.0	2.2	0.4
Luxburg	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3.7	0.6	0.0	0.0	1.2	0.4
Split	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.0	0.6	0.0	0.0	1.2	0.4

Success rate (%)

Method	s1	s2	s3	s4	a1	a2	a3	unb	b1	b2	dim32	Aver.	Fails
Rand-P	0%	100%	100%	100%	0%	0%	0%	0%	0%	0%	0%	**27%**	**10**
Rand-C	96%	100%	100%	100%	56%	2%	0%	0%	0%	0%	2%	**41%**	**10**
Maxmin	100%	100%	100%	100%	100%	58%	36%	100%	0%	0%	100%	**72%**	**4**
kmeans++	100%	100%	100%	100%	98%	20%	0%	100%	0%	0%	100%	**65%**	**4**
Bradley	100%	100%	100%	100%	100%	4%	4%	4%	0%	0%	84%	**54%**	**6**
Sorting	100%	100%	100%	100%	100%	24%	0%	0%	2%	100%	100%	**66%**	**4**
Projection	100%	100%	100%	100%	100%	18%	0%	0%	0%	100%	100%	**65%**	**4**
Luxburg	100%	100%	100%	100%	100%	100%	100%	0%	46%	100%	100%	**86%**	**2**
Split	100%	100%	100%	100%	100%	100%	100%	0%	36%	100%	100%	**85%**	**2**

Running time (s)

Method	s1	s2	s3	s4	a1	a2	a3	unb	b1	b2	dim32
Rand-P	3.2	4.4	4.7	6.6	3.1	8.9	15	5.0	1657	1037	0.3
Rand-C	2.6	3.2	4.1	8.2	2.0	4.4	7.5	4.8	882	172	0.4
Maxmin	1.9	2.4	3.7	5.6	1.8	3.2	5.6	2.5	596	146	0.3
kmeans++	1.9	2.8	3.7	6.6	1.9	3.7	6.4	4.1	604	143	0.4
Bradley	2.5	2.7	3.5	5.6	1.9	4.5	7.8	4.7	605	195	1.0
Sorting	2.3	3.1	3.8	7.6	1.9	4.4	6.8	4.9	815	148	0.3
Projection	2.1	3.0	3.7	7.1	1.7	4.1	6.6	5.0	768	84	0.3
Luxburg	1.9	2.4	3.2	6.1	1.7	3.5	5.5	4.9	431	126	0.4
Split	3.5	3.9	5.3	6.9	2.3	5.8	10	11	1072	988	1.2

the maxmin variants with 100 repeats can overcome this problem.

Table 7 summarizes how the four factors affect the different initialization techniques and the k-means iterations.

5. Conclusions

On average, k-means caused errors with about 15% of the clusters (CI = 4.5). By repeating k-means 100 times this errors was reduced to 6% (CI = 2.0). Using a better initialization technique (Maxmin), the corresponding numbers were 6% (CI = 2.1) with k-means as such, and 1% (CI = 0.7) with 100 repeats. For most pattern recognition applications this accuracy is more than enough when clustering is just one component within a complex system.

The most important factor is the cluster overlap. In general, well separated clusters make the clustering problem easier but for k-means it is just the opposite. When the clusters overlap, k-means iterations work reasonably well regardless of the initialization. This is the expected situation in most pattern recognition applications.

The number of errors have a linear dependency on the number of clusters (k): the more clusters, the more errors k-means makes, but the percentage remains constant. Unbalance of cluster sizes is more problematic. Most initialization techniques fail, and only the maxmin heuristics worked in this case. The clustering result then depends merely on the goodness of the initialization technique.

Dimensionality itself is not a factor. It merely matters how the dimensions affect the cluster overlap. With our data, the clusters became more separated when the dimensions were increased,

Table 6
Number of repeats in RKM to reach certain CI-level. Missing values (–) indicate that this CI-level was never reached during the 200,000 repeats.

A3 CI-value							
Initialization	6	5	4	3	2	1	0
Rand-P	–	–	–	–	–	–	–
Rand-C	2	4	11	54	428	11,111	–
Maxmin				1	3	14	216
Kmeans++		1	2	3	14	138	8696
Bradley		1	2	8	58	1058	33,333
Sorting	1	2	4	13	73	1143	–
Projection	1	2	3	9	46	581	18,182
Luxburg						1	3
Split					1	2	9
Unbalance CI-value							
Initialization	6	5	4	3	2	1	0
Rand-P			1	97	8333	–	–
Rand-C			1	16	69	1695	100k
Maxmin						1	4
Kmeans++						1	2
Bradley			1	3	6	70	1471
Sorting			1	–	–	–	–
Projection			1	935	16,667	–	–
Luxburg			1	59	16,667	–	–
Split			1	9524	–	–	–

Table 7
How the four factors have effect on the performance of the initialization and on the k-means iterations.

Method	Overlap	Clusters	Dimension	Unbalance
Rand-P	No effect	Constant	No effect	Very bad
Rand-C	No effect	Constant	No effect	Very bad
Maxmin	Bad	Constant	No effect	A bit worse
kmeans++	A bit worse	Constant	No effect	A bit worse
Bradley	Good	Constant	No effect	Bad
Sorting	A bit worse	Constant	No effect	Very bad
Projection	A bit worse	Constant	No effect	Very bad
Luxburg	A bit worse	Minor effect	No effect	Very bad
Split	A bit worse	Constant	No effect	Very bad
KM iterations	**Good**	**Constant**	**No effect**	**No effect**

which in turn worsened the k-means performance. Besides this indirect effect, the dimensions did not matter much.

With real data the effect might be just the opposite. If the features (attributes) are added in the order of their clustering capability, it is expected that the clusters would become more overlapping when adding more features. As a result, k-means would start to work better but the data itself would become more difficult to cluster, possibly losing the clustering structure. And vice versa, if good feature selection is applied, the clusters can be more separated, which has the danger that k-means would start to perform worse.

Based on these observations, choosing an initialization technique like Maxmin can compensate for the weaknesses of k-means. With unbalanced cluster sizes it might work best overall. However, it is preferable to repeat the k-means 10–100 times; each time taking a random point as the first centroids and selecting the rest using the Maxmin heuristic. This will keep the number of errors relatively small.

However, the fundamental problem of k-means still remains when the clusters are well separated. From all the tested combinations, none was able to solve all the benchmark datasets despite them being seemingly simple. With 100 repeats, Maxmin and k-means++ solved 7 datasets (out of the 11), thus being the best initialization techniques. The better standalone algorithms (Luxburg and Split) managed to solve 9.

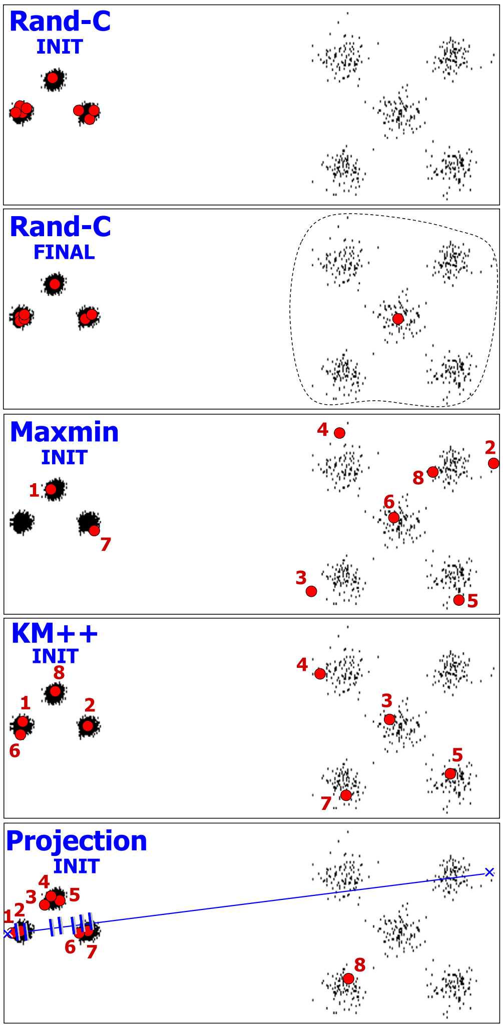


Fig. 17. Examples of the initialization technique on the Unbalance dataset. The only techniques that do not badly fail are the maxmin heuristics. The numbers indicate the order in which the centroids are selected.

To sum up, if the clusters overlap, the choice of initialization technique does not matter much, and repeated k-means is usually good enough for the application. However, if the data has well-separated clusters, the result of k-means depends merely on the initialization algorithm.

In general, the problem of initialization is not any easier than solving the clustering problem itself. Therefore, if the accuracy of clustering is important, then a better algorithm should be used. Using the same computing time spent for repeating k-means, a simple alternative called random swap (RS) [12] solves all the benchmark datasets. Other standalone algorithms that we have found able to solve all the benchmark sets include genetic algorithm (GA) [10], the split algorithm [46], split k-means [46], and density peaks [78]. Agglomerative clustering [30] solves 10 out of 11.

References

[1] E. Forgy, Cluster analysis of multivariate data: efficiency vs. interpretability of classification, Biometrics 21 (1965) 768–780.
[2] J. MacQueen, Some methods for classification and analysis of multivariate observations, in: Berkeley Symposium on Mathematical Statistics and Probability, 1, Statistics University of California Press, Berkeley, Calif., 1967, pp. 281–297.
[3] S.P. Lloyd, Least squares quantization in PCM, IEEE Trans. Inf. Theory 28 (2) (1982) 129–137.
[4] L. Wang, C. Pan, Robust level set image segmentation via a local correntropy--based k-means clustering, Pattern Recognit. 47 (2014) 1917–1925.
[5] C. Boutsidis, A. Zouzias, M.W. Mahoney, P. Drineas, Randomized dimensionality reduction for k-means clustering, IEEE Trans. Inf. Theory 61 (2, February) (2015) 1045–1062.
[6] M. Capo, Perez A, J.A. Lozano, An efficient approximation to the k-means clustering for massive data, Knowl.-Based Syst. 117 (2017) 56–69.
[7] Z. Huang, N. Li, K. Rao, C. Liu, Y. Huang, M. Ma, Z. Wang, Development of a data-processing method based on Bayesian k-means clustering to discriminate aneugens and clastogens in a high-content micronucleus assay, Hum. Exp. Toxicol. 37 (3) (2018) 285–294.
[8] A.K. Jain, Data clustering: 50 years beyond K-means, Pattern Recognit. Lett. 31 (2010) 651–666.
[9] K. Krishna, Murty M.N, Genetic k-means algorithm, IEEE Trans. Syst. Man Cybern. Part B 29 (3) (1999) 433–439.
[10] P. Fränti, Genetic algorithm with deterministic crossover for vector quantization, Pattern Recognit. Lett. 21 (1) (2000) 61–68.
[11] P. Fränti, J. Kivijärvi, Randomized local search algorithm for the clustering problem, Pattern Anal. Appl. 3 (4) (2000) 358–369.
[12] P. Fränti, Efficiency of random swap clustering, J. Big Data 5 (13) (2018) 1–29.
[13] S. Kalyani, K.S. Swarup, Particle swarm optimization based K-means clustering approach for security assessment in power systems, Expert Syst. Appl. 32 (9) (2011) 10839–10846.
[14] D. Yan, L. Huang, M.I. Jordan, Fast approximate spectral clustering, ACM SIGKDD Int. Conf. Knowl. Discov. Data Min. (2009) 907–916.
[15] L. Bai, X. Cheng, J. Liang, H. Shen, Y. Guo, Fast density clustering strategies based on the k-means algorithm, Pattern Recognit. 71 (2017) 375–386.
[16] T. Kinnunen, I. Sidoroff, M. Tuononen, P. Fränti, Comparison of clustering methods: a case study of text-independent speaker modeling, Pattern Recognit. Lett. 32 (13, October) (2011) 1604–1617.
[17] Q. Zhao, P. Fränti, WB-index: a sum-of-squares based index for cluster validity, Data Knowl. Eng. 92 (July) (2014) 77–89.
[18] M. Rezaei and P. Fränti Can the number of clusters be solved by external index? manuscript. (submitted)
[19] J.M Peña, J.A. Lozano, P. Larrañaga, An empirical comparison of four initialization methods for the k-means algorithm, Pattern Recognit. Lett. 20 (10, October) (1999) 1027–1040.
[20] J. He, M. Lan, C-L Tan, S-Y Sung, H-B Low, Initialization of Cluster Refinement Algorithms: a review and comparative study, IEEE Int. Joint Conf. Neural Netw. (2004).
[21] D. Steinley, M.J. Brusco, Initializing k-means batch clustering: a critical evaluation of several techniques, J. Classification 24 (2007) 99–121.
[22] M.E. Celebi, H.A. Kingravi, P.A. Vela, A comparative study of efficient initialization methods for the k-means clustering algorithm, Expert Syst. Appl. 40 (2013) 200–210.
[23] L. Kaufman, P. Rousseeuw, Finding Groups in data: An introduction to Cluster Analysis, Wiley Interscience, 1990.
[24] B. Thiesson, C. Meek, D.M. Chickering, and D. Heckerman, Learning mixtures of Bayesian networks, Technical Report MSR-TR-97-30 Cooper & Moral, 1997.
[25] J.T. Tou, R.C. Gonzales, Pattern Recognition Principles, Addison-Wesley, 1974.
[26] T.F. Gonzalez, Clustering to minimize the maximum intercluster distance, Theor. Comput. Sci. 38 (2–3) (1985) 293–306.
[27] J.H. Ward, Hierarchical grouping to optimize an objective function, J. Am. Stat. Assoc. 58 (301) (1963) 236–244.
[28] A. Likas, N. Vlassis, J. Verbeek, The global k-means clustering algorithm, Pattern Recognit. 36 (2003) 451–461.
[29] D. Steinley, Local optima in k-means clustering: what you don't know may hurt you, Psychol. Methods 8 (2003) 294–304.
[30] P. Fränti, T. Kaukoranta, D.-F. Shen, K.-S. Chang, Fast and memory efficient implementation of the exact PNN, IEEE Trans. Image Process. 9 (5, May) (2000) 773–777.
[31] P. Bradley, U. Fayyad, Refining initial points for k-means clustering, in: International Conference on Machine Learning, San Francisco, 1998, pp. 91–99.
[32] R.O. Duda, P.E. Hart, Pattern Classification and Scene Analysis, John Wiley and Sons, New York, 1973.
[33] M. Bicego, M.A.T. Figueiredo, Clustering via binary embedding, Pattern Recognit. 83 (2018) 52–63.
[34] N. Karmitsa, A.M. Bagirov, S. Taheri, Clustering in large data sets with the limited memory bundle method, Pattern Recognit. 83 (2018) 245–259.
[35] Y. Zhu, K.M. Ting, M.J. Carman, Grouping points by shared subspaces for effective subspace clustering, Pattern Recognit. 83 (2018) 230–244.
[36] P.B. Frandsen, B. Calcott, C. Mayer, R. Lanfear, Automatic selection of partitioning schemes for phylogenetic analyses using iterative k-means clustering of site rates, BMC Evol. Biol. 15 (13) (2015).
[37] D.G. Márquez, A. Otero, P. Félix, C.A. García, A novel and simple strategy for evolving prototype based clustering, Pattern Recognit. 82 (2018) 16–30.
[38] L. Huang, H.-Y. Chao, C.-D. Wang, Multi-view intact space clustering, Pattern Recognit. 86 (2019) 344–353.
[39] P. Fränti, S. Sieranoja, K-means properties on six clustering benchmark datasets, Appl. Intel. 48 (12) (2018) 4743–4759.
[40] L. Morissette, S. Chartier, The k-means clustering technique: general considerations and implementation in Mathematica, Tutor. Quant. Methods Psychol. 9 (1) (2013) 15–24.
[41] J. Liang, L. Bai, C. Dang F. Cao, The k-means-type algorithms versus imbalanced data distributions, IEEE Trans. Fuzzy Syst. 20 (4, August) (2012) 728–745.
[42] I. Melnykov, V. Melnykov, On k-means algorithm with the use of Mahalanobis distances, Stat. Probab. Lett. 84 (January) (2014) 88–95.
[43] V. Melnykov, S. Michael, I. Melnykov, Recent developments in model-based clustering with applications, in: M. Celebi (Ed.), Partitional Clustering Algorithms, Springer, Cham, 2015.
[44] M. Rezaei, P. Fränti, Set-matching methods for external cluster validity, IEEE Trans. Knowl. Data Eng. 28 (8, August) (2016) 2173–2186.
[45] P. Fränti, M. Rezaei, Q. Zhao, Centroid index: cluster level similarity measure, Pattern Recognit. 47 (9) (2014) 3034–3045.
[46] P. Fränti, T. Kaukoranta, O. Nevalainen, On the splitting method for VQ codebook generation, Opt. Eng. 36 (11, November) (1997) 3043–3051.
[47] P. Fränti, O. Virmajoki, V. Hautamäki, Fast agglomerative clustering using a k-nearest neighbor graph, IEEE Trans. Pattern Anal. Mach. Intel. 28 (11, November) (2006) 1875–1881.
[48] G.H. Ball, D.J. Hall, A clustering technique for summarizing multivariate data, Syst. Res. Behav. Sci. 12 (2, March) (1967) 153–155.
[49] O. Lemke, B. Keller, Common nearest neighbor clustering: a benchmark, Algorithms 11 (2) (2018) 19.
[50] U.V. Luxburg, Clustering stability: an overview, Found. Trends Mach. Learn. 2 (3) (2010) 235–274.
[51] S.J. Redmond, C. Heneghan, A method for initialising the K-means clustering algorithm using kd-trees, Pattern Recognit. Lett. 28 (8) (2007) 965–973.
[52] S. Tezuka, P.L Equyer, Efficient portable combined Tausworthe random number generators, ACM Trans. Model. Comput. Simul. 1 (1991) 99–112.
[53] M.J. Norušis, IBM SPSS Statistics 19 Guide to Data Analysis, Prentice Hall, Upper Saddle River, New Jersey, 2011.
[54] T. Gonzalez, Clustering to minimize the maximum intercluster distance, Theor. Comput. Sci. 38 (2–3) (1985) 293–306.
[55] M.M.-T. Chiang, B. Mirkin, Intelligent choice of the number of clusters in k-means clustering: an experimental study with different cluster spreads, J. Classification 27 (2010) 3–40.
[56] J. Hämäläinen, T. Kärkkäinen, Initialization of big data clustering using distributionally balanced folding, Proceedings of the European Symposium on Artificial Neural Networks, Comput. Intel. Mach. Learn.-ESANN (2016).
[57] I. Katsavounidis, C.C.J. Kuo, Z. Zhang, A new initialization technique for generalized Lloyd iteration, IEEE Signal Process Lett. 1 (10) (1994) 144–146.
[58] F. Cao, J. Liang, L. Bai, A new initialization method for categorical data clustering, Expert Syst. Appl. 36 (7) (2009) 10223–10228.
[59] D. Arthur, S. Vassilvitskii, K-means++: the advantages of careful seeding, ACM-SIAM Symp. on Discrete Algorithms (SODA'07), January 2007.
[60] M. Erisoglu, N. Calis, S. Sakallioglu, A new algorithm for initial cluster centers in k-means algorithm, Pattern Recognit. Lett. 32 (14) (2011) 1701–1705.
[61] C. Gingles, M. Celebi, Histogram-based method for effective initialization of the k-means clustering algorithm, Florida Artificial Intelligence Research Society Conference, May 2014.
[62] J.A. Hartigan, M.A. Wong, Algorithm AS 136: a k-means clustering algorithm, J. R. Stat. Soc. C 28 (1) (1979) 100–108.
[63] M.M. Astrahan, Speech Analysis by Clustering, Or the Hyperphome Method, Stanford Artificial Intelligence Project Memorandum AIM-124, Stanford University, Stanford, CA, 1970.
[64] F. Cao, J. Liang, G. Jiang, An initialization method for the k-means algorithm using neighborhood model, Comput. Math. Appl. 58 (2009) 474–483.
[65] M. Al-Daoud, A new algorithm for cluster initialization, in: World Enformatika Conference, 2005, pp. 74–76.
[66] M. Yedla, S.R. Pathakota, T.M. Srinivasa, Enhancing k-means clustering algorithm with improved initial center, Int. J. Comput. Sci. Inf. Technol. 1 (2) (2010) 121–125.
[67] T. Su, J.G. Dy, In search of deterministic methods for initializing k-means and gaussian mixture clustering, Intel. Data Anal. 11 (4) (2007) 319–338.
[68] X. Wu, K. Zhang, A better tree-structured vector quantizer, in: IEEE Data Compression Conference, Snowbird, UT, 1991, pp. 392–401.
[69] C.-M. Huang, R.W. Harris, A comparison of several vector quantization codebook generation approaches, IEEE Trans. Image Process. 2 (1) (1993) 108–112.
[70] D. Boley, Principal direction divisive partitioning, Data Min. Knowl. Discov. 2 (4) (1998) 325–344.
[71] M.E. Celebi, H.A. Kingravi, Deterministic initialization of the k-means algorithm using hierarchical clustering, Int. J. Pattern Recognit Artif Intell. 26 (07) (2012) 1250018.
[72] S. Sieranoja, P. Fränti, Random projection for k-means clustering, in: Int. Conf. Artificial Intelligence and Soft Computing (ICAISC), Zakopane, Poland, June 2018, pp. 680–689.

[73] S.-W. Ra, J.-K. Kim, A fast mean-distance-ordered partial codebook search algorithm for image vector quantization, IEEE Trans. Circuits Syst. 40 (September) (1993) 576–579.
[74] I. Cleju, P. Fränti, X. Wu, Clustering based on principal curve, in: Scandinavian Conf. On Image Analysis, LNCS, vol. 3540, Springer, Heidelberg, 2005, pp. 872–881.
[75] X. Wu, Optimal quantization by matrix searching, J. Algorithms 12 (4) (1991) 663–673.
[76] M.B. Al-Daoud, S.A. Roberts, New methods for the initialisation of clusters, Pattern Recognit. Lett. 17 (5) (1996) 451–455.
[77] P. Gourgaris, C. Makris, A Density Based K-Means Initialization Scheme, EANN workshops, Rhodes Island, Greece, 2015.
[78] A. Rodriquez, A. Laio, Clustering by fast search and find of density peaks, Science 344 (6191) (2014) 1492–1496.
[79] P. Mitra, C. Murthy, S.K. Pal, Density-based multiscale data condensation, IEEE Trans. Pattern Anal. Mach. Intel. 24 (6) (2002) 734–747.
[80] S. Sieranoja, P. Fränti, Constructing a high-dimensional kNN-graph using a Z-order curve, ACM J. Exp. Algorithmics 23 (1, October) (2018) 1–21 1.9:.
[81] W. Dong, C. Moses, K. Li, Efficient k-nearest neighbor graph construction for generic similarity measures, in: Proceedings of the ACM International Conference on World wide web, ACM, 2011, pp. 577–586.
[82] P. Fränti, S. Sieranoja, Dimensionally distributed density estimation, in: Int. Conf. Artificial Intelligence and Soft Computing (ICAISC), Zakopane , Poland, June 2018, pp. 343–353.
[83] H.J. Curti, R.S. Wainschenker, FAUM: fast Autonomous Unsupervised Multidimensional classification, Inf. Sci. 462 (2018) 182–203.
[84] J. Xie, Z.Y. Xiong, Y.F. Zhang, Y. Feng, J. Ma, Density core-based clustering algorithm with dynamic scanning radius, Knowl.-Based Syst. 142 (2018) 68–70.
[85] Y. Linde, A. Buzo, R.M. Gray, An algorithm for vector quantizer design, IEEE Trans. Commun. 28 (1, January) (1980) 84–95.
[86] M. Steinbach, G. Karypis, V. Kumar, A comparison of document clustering techniques, in: KDD workshop on text mining, vol. 400, Boston, 2000, pp. 525–526.
[87] S-S. Yu, S-W. Chu, C-M. Wang, Y-K. Chan, T-C. Chang, Two improved k-means algorithms, Appl. Soft Comput. 68 (2018) 747–755.
[88] B. Bahmani, B. Mosley, A. Vattani, R. Kumar, S. Vassilvitski, Scalable k-means++, Proc. VLDB Endow. 5 (7) (2012) 622–633.
[89] T. Kaukoranta, P. Fränti, O. Nevalainen, A fast exact GLA based on code vector activity detection, IEEE Trans. Image Process. 9 (8, August) (2000) 1337–1342.

Pasi Fränti received his MSc and PhD degrees from the University of Turku, 1991 and 1994 in Science. Since 2000, he has been a professor of Computer Science at the University of Eastern Finland (UEF). He has published 81 journals and 167 peer review conference papers, including 14 IEEE transaction papers. His main research interests are in machine learning, data mining, pattern recognition including clustering algorithms and intelligent location-aware systems. Significant contributions have also been made in image compression, image analysis, vector quantization and speech technology.

Sami Sieranoja received the B.Sc. and M.Sc. degrees in University of Eastern Finland, 2014 and 2015. Currently he is a doctoral student at the University of Eastern Finland. His research interests include neighborhood graphs and data clustering.

SAMI SIERANOJA

Clustering algorithms can find patterns in data by separating it into groups consisting of similar objects. In this thesis we present new methods for data clustering and analyze the performance of existing methods. The results allow to apply clustering based data analysis on big datasets and provide important information on the limitations of existing methods.

**PUBLICATIONS OF
THE UNIVERSITY OF EASTERN FINLAND**
Dissertations in Forestry and Natural Sciences

ISBN 978-952-61-3644-8
ISSN 1798-5668