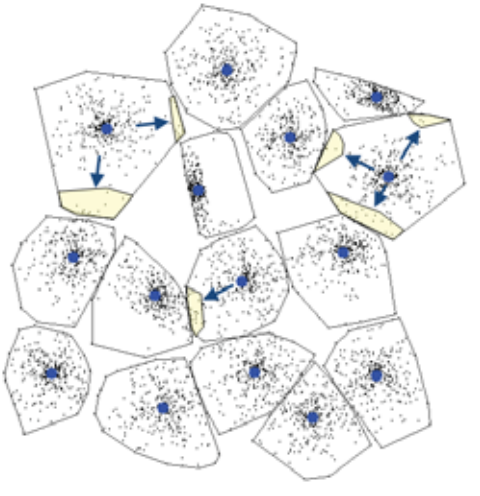
MIKKO MALINEN

New Alternatives for k-Means Clustering



Publications of the University of Eastern Finland
Dissertations in Forestry and Natural Sciences
No 178

Academic Dissertation
To be presented by permission of the Faculty of Science and Forestry for public examination in the Metria M100 Auditorium at the University of Eastern Finland, Joensuu, on June, 25, 2015,
at 12 o'clock noon.

School of Computing

Kopio Niini Oy
Helsinki, 2015
Editors: Research director Pertti Pasanen, Prof. Pekka Kilpeläinen,
Prof. Kai Peiponen, Prof. Matti Vornanen

Distribution:
University of Eastern Finland Library / Sales of publications
julkaisumyynti@uef.fi
http://www.uef.fi/kirjasto

ISBN: 978-952-61-1788-1 (printed)
ISSNL: 1798-5668
ISSN: 1798-5668
ISBN: 978-952-61-1789-8 (pdf)
ISSNL: 1798-5668
ISSN: 1798-5668

Author's address:
University of Eastern Finland
School of Computing
Box 111
FIN-80101 JOENSUU
FINLAND
email: mmali@cs.uef.fi

Supervisor:
Professor Pasi Fränti, Ph.D.
University of Eastern Finland
School of Computing
Box 111
FIN-80101 JOENSUU
FINLAND
email: franti@cs.uef.fi

Reviewers:
Professor Erkki Mäkinen, Ph.D.
University of Tampere
School of Information Sciences
FI-33014 Tampereen yliopisto
FINLAND
email: em@sis.uta.fi

Professor Olli Nevalainen, Ph.D.
University of Turku
Department of Information Technology
FI-20014 TURUN YLIOPISTO
FINLAND
email: olli.nevalainen@utu.fi

Opponent:
Professor Refael Hassin, Ph.D.
Tel-Aviv University
Department of Statistics and Operations Research
Tel-Aviv 69978
ISRAEL
email: hassin@post.tau.ac.il

ABSTRACT

This work contains several theoretical and numerical studies on data clustering. The total squared error (*TSE*) between the data points and the nearest centroids is expressed as an analytic function, the gradient of that function is calculated, and the gradient descent method is used to minimize the *TSE*.

In balance-constrained clustering, we optimize *TSE*, but so that the number of points in clusters are equal. In balance-driven clustering, balance is an aim but is not mandatory. We use a cost function summing all squared pairwise distances and show that it can be expressed as a function which has factors for both balance and *TSE*. In Balanced *k*-Means, we use the Hungarian algorithm to find the minimum *TSE*, subject to the constraint that the clusters are of equal size.

In traditional clustering, one fits the model to the data. We present also a clustering method, that takes an opposite approach. We fit the data to an artificial model and make a gradual inverse transform to move the data its original locations and perform *k*-means at every step.

We apply the divide-and-conquer method for quickly calculate an approximate minimum spanning tree. In the method, we divide the dataset into clusters and calculate a minimum spanning tree of each cluster. To complete the minimum spanning tree, we then combine the clusters.

Universal Decimal Classification: 004.93, 517.547.3, 519.237.8
AMS Mathematics Subject Classification: 30G25, 62H30, 68T10
INSPEC Thesaurus: pattern clustering; classification; functions; gradient methods; mean square error methods; nonlinear programming; optimization; data analysis
Yleinen suomalainen asiasanasto: data; klusterit; järjestäminen; luokitus; analyyttiset funktiot; virheanalyysi; optimointi; algoritmit

Preface

The work presented in this thesis was carried out at the School of Computing, University of Eastern Finland, Finland, during the years 2009–2015.

I want to express my special thanks to my supervisor, Prof. Pasi Fränti. In 2009 he took me on to do research. His numerous comments have had an impact in improving my papers. He is also active in hobbies, and because of him I started ice swimming, in which I have later competed in Finnish and World championships, and orienteering. The chess tournaments organized by him led me to improve my skills in chess.

I wish to thank those colleagues with whom I have worked and talked during these years, especially, Dr. Qinpei Zhao, Dr. Minjie Chen, Dr. Rahim Saeidi, Dr. Tomi Kinnunen, Dr. Ville Hautamäki, Dr. Caiming Zhong, and doctoral students Mohammad Rezaei and Radu Mariescu-Istodor. I thank M.A.(Hons.) Pauliina Malinen Teodoro for language checking of some of my articles.

I am thankful to Prof. Erkki Mäkinen and Prof. Olli Nevalainen, the reviewers of the thesis, for their feedback and comments. I would also thank Prof. Refael Hassin for acting as my opponent.

I also greatly appreciate my family and all of my friends, who have given me strength during these years.

This research has been supported by the School of Computing, University of Eastern Finland, the East Finland Graduate School in Computer Science and Engineering (ECSE) and the MOPSI project.

I have enjoyed every single day!

Joensuu 11th May, 2015 *Mikko Malinen*

LIST OF PUBLICATIONS

This thesis consists of the present review of the author's work in the field of data clustering and graph theory and the following selection of the author's publications:

I M. I. Malinen and P. Fränti, "Clustering by analytic functions," *Information Sciences* **217,** 31–38 (2012).

II M. I. Malinen, R. Mariescu-Istodor and P. Fränti, "*K*-means*: Clustering by gradual data transformation," *Pattern Recognition* **47** (10), 3376–3386 (2014).

III M. I. Malinen and P. Fränti, "All-pairwise squared distances lead to balanced clustering", manuscript (2015).

IV M. I. Malinen and P. Fränti, "Balanced K-means for clustering", *Joint Int. Workshop on Structural, Syntactic, and Statistical Pattern Recognition (S+SSPR 2014),* LNCS 8621, 32–41, Joensuu, Finland, 20–22 August (2014).

V C. Zhong, M. Malinen, D. Miao and P. Fränti, "A fast minimum spanning tree algorithm based on *k*-means", *Information Sciences* **295**, 1–17 (2015).

Throughout the overview, these papers will be referred to by Roman numerals. The papers are included at the end of the thesis by the permission of their copyright holders.

AUTHOR'S CONTRIBUTION

The publications selected in this dissertation are original research papers on data clustering and graph theory. The idea of the Papers **I** – **IV** was originated by the first author Mikko I. Malinen and refined on discussions of the author and the co-authors. The idea of the paper **V** is of the first author of the paper.

In publications **I** – **IV** the author has carried out all numerical computations and the selection of the used methods have been done by him. In publication **V** the author has taken part in the theoretical analysis.

The author has written the manuscript to the papers **I**–**IV**; in the paper **II** the last co-author has written part of the text and the first co-author has made the clustering animator. In the paper **III** the last co-author has written the abstract. In the paper **V** the first author of the paper has written the manuscript.

Contents

1 Introduction

We are living the middle of a digital revolution. ”Digital revolution” means that most of the information content we store and transmit will be coded in digital form, that is, in bits. The digital revolution could be considered to have started, when Shannon introduced the term bit in the 1940's. One term that has become popular the last years, is ”Big data”. This means that datasets are becoming bigger in number and size.

1.1 CLUSTERING IS AN NP-HARD PROBLEM

Clustering is an important tool in data mining and machine learning. It aims at partitioning the objects of a dataset so that similar objects will be put into the same clusters and different objects in different clusters. Sum-of-squares clustering, which is the most commonly used clustering approach, and which this thesis mostly discusses, is an NP-hard problem [1]. This means that an optimal clustering solution cannot be achieved except for very small datasets. When the number of clusters k is constant, Euclidean sum-of-squares clustering can be done in polynomial $O(n^{kd+1})$ time [2], where d is the number of dimensions. This is slow in practice, since the power $kd + 1$ is high, and thus, suboptimal algorithms are used.

1.2 THE AIMS OF CLUSTERING

Clustering aims at assigning similar objects into the same groups and dissimilar objects into different groups. Similarity is typically measured by the distance between the objects. The most typical criterion for the goodness of a clustering is the mean squared error (*MSE*) or total squared error (*TSE*), which are related: they differ only by a constant factor. The goodness of clustering can be also measured by cluster validity indices, but these are typically

not used as a cost function, because of the more complicated optimization entailed. Widely used external validity indices are the Adjusted Rand index [3], the Van Dongen index [4], and the Normalized mutual information index [5]. *MSE* or *TSE* is the most common cost function in clustering. It is often called the *k*-means method, which means the *MSE* cost function.

The time complexity of clustering varies from $O(n)$ in grid-based clustering to $O(n^3)$ in the PNN algorithm [6]. The most common clustering algorithm *k*-means takes time

$$T(n) = O(I \cdot k \cdot n), \tag{1.1}$$

where k is the number of clusters and I is the number of iterations. The *k*-means algorithm is fast in practice, but in worst case, it can be slow when the number of iterations is large. An upper bound for the number of iterations is $O(n^{kd})$ [7].

In *balanced clustering*, we need to balance the clusters in addition to optimize the *MSE*. Sometimes balance is an aim, but not a mandatory requirement, as in the Scut method in paper **III**, where we have both *MSE* and balance affecting the cost function. Sometimes, the balance is a mandatory requirement, and the *MSE* optimization is a secondary criterion, as in paper **IV**.

1.3 DISTANCE MEASURE AND CLUSTERING CRITERION

Clustering requires two choices to be made: how to measure the distance between two points, and how to measure the error of the clustering. One distance measure is the L_1 norm, i. e., the Manhattan distance

$$d_1(\bar{x}, \bar{c}) = \sum_{i=1}^{d} ||x_{\{i\}} - c_{\{i\}}||, \tag{1.2}$$

where $(\bar{x}, \bar{c})$ are vectors

$$\bar{x} = (x_{\{1\}}, x_{\{2\}}, ..., x_{\{d\}}) \text{ and } \bar{c} = (c_{\{1\}}, c_{\{2\}}, ..., c_{\{d\}}) \tag{1.3}$$

and by $x_{\{i\}}$ and $c_{\{i\}}$ we mean the i:th component (feature) of vectors (points) $\bar{x}$ and $\bar{c}$, respectively. Another commonly used distance measure is the L_2 norm, the Euclidean distance

$$d_2(\bar{x}, \bar{c}) = \sqrt{\sum_{i=1}^{d} (x_{\{i\}} - c_{\{i\}})^2}. \tag{1.4}$$

The Minkowski norm L_p can also be used with freely chosen p:

$$d_p(\bar{x}, \bar{c}) = (\sum_{i=1}^{d} (x_{\{i\}} - c_{\{i\}})^p)^{1/p}. \tag{1.5}$$

The L_∞ norm can also be used

$$d_\infty(\bar{x}, \bar{c}) = \max(x_{\{i\}} - c_{\{i\}}). \tag{1.6}$$

In this thesis, we use Euclidean distance, but in paper **I** we also tell how the L_∞-norm could be used in practice.

The clustering criterion determines how the distances affect the error measure. Some error measures are sum-of-squares, that is, the total squared error, mean squared error, infinite norm error and mean absolute error. The total squared error of the clustering is calculated as

$$TSE = \sum_{X_i \in P_j} ||X_i - C_j||^2, \tag{1.7}$$

where X_i is the data point, C_j is the centroid, and P_j is the partition of cluster j. The mean squared error MSE is defined as

$$MSE = TSE/n, \tag{1.8}$$

where n is the number of points in the dataset. MSE is the most widely used criterion, and minimizing MSE leads to the same result as minimizing TSE. Some other criteria are the mean absolute error and the infinite norm error. The mean absolute error leads to a clustering which gives less weight to outliers, which are single points outside the dense regions of the dataset. Outliers often follow from incorrect measurements in data collecting.

2 Solving Clustering

2.1 THE NUMBER OF CLUSTERS

Most clustering algorithms require the user to give the number of clusters as an input to the algorithm. Some algorithms determine the number of clusters at run time. Often the user has no a priori information about the proper number of clusters, and then the calculation of a validity index may be needed to obtain this information.

Two widely used validity indices for this purpose are the Silhouette coefficient [8] and the F-ratio (WB-index) [9]. Also, a way to determine the number of clusters is the minimum description length (MDL) principle [10] by Rissanen. In MDL for clustering one calculates the length of the code needed to describe the data plus code length to describe the model. This sum varies when the number of clusters changes. The first term decreases and the second term increases when the number of clusters increase. The minimum description length is the minimum of this sum. It is one of the few connections between information theory and clustering. The principle is written here formally in its general form [10], which is most useful in a short introduction like this:
Find a model with which the observed data and the model can be encoded with the shortest code length

$$\min_{\theta,k}[\log \frac{1}{f(X;\theta,k)} + L(\theta,k)], \tag{2.1}$$

where f is the maximum likelihood of the model, θ and k are the parameters defining the model, and $L(\theta,k)$ denotes the code length for the parameters defining the model.

2.2 CLUSTERING ALGORITHMS

When the cost function has been defined the clustering problem becomes an algorithmic problem.

2.2.1 k-Means

The k-means algorithm [11] starts by initializing the k centroids. Typically, a random selection among the data points is made, but other techniques are discussed in [12–14]. Then k-means consists of two repeatedly executed steps [15]:

Assignment step: Assign each data point X_i to clusters specified by the nearest centroid:

$$P_j^{(t)} = \{X_i : \|X_i - C_j^{(t)}\| \leq \|X_i - C_{j^*}^{(t)}\| \quad \text{for all} \quad j^* = 1, ..., k\}.$$

Update step: Calculate the mean of each cluster:

$$C_j^{(t+1)} = \frac{1}{|P_j^{(t)}|} \sum_{X_i \in P_j^{(t)}} X_i.$$

These steps are repeated until the centroid locations do not change anymore. The k-means assignment step and update step are optimal with respect to MSE in the sense that the partitioning step minimizes the MSE for a given set of centroids and the update step minimizes MSE for a given partitioning. The solution converges to a local optimum but without a guarantee of global optimality. To get better results than k-means, slower agglomerative algorithms [6, 16, 17] or more complex k-means variants [14, 18–20] are sometimes used. Gaussian mixture models can also be used (Expectation-Maximization algorithm) [21, 22].

2.2.2 Random Swap

To overcome the low accuracy of k-means, the *randomized local search* (RLS) algorithm [18] has been developed. It is often called the

random swap algorithm. Once a clustering result is available, one centroid is randomly swapped to another location and k-means is performed. If the result gets better, it is saved. The swapping is continued until the desired number of iterations is done. With a large enough number of iterations, often 5000, it gives good results, making it one of the best clustering algorithms available. For a pseudocode of random swap see Algorithm 1.

Algorithm 1 Random Swap

```
C ← SelectRandomDataObjects(k)
P ← OptimalPartition(C)
repeat
  C^new ← RandomSwap(C)
  P^new ← LocalRepartition(P, C^new)
  k-Means(P^new, C^new)
  if MSE(P^new, C^new) < MSE(P, C) then
    (P, C) ← (P^new, C^new)
  end if
until T times
```

2.2.3 Other Hierarchical and Partitional Algorithms

The pairwise nearest neighbor (PNN) algorithm [6] gives good accuracy, but with a high time complexity: $T = O(n^3)$. It starts with all points in their own clusters. It finds the point pair which has the lowest merge cost and merges it. This merging is continued until the number of clusters is the desired k. A faster version of PNN [16] runs with a time complexity $O(\tau n^2)$, where τ is a data-dependent variable expressing the size of the neighborhood.

k-Means++ [14], which is based on k-means, emphasizes a good choice of initial centroids, see Algorithm 2. Let $D(X_i)$ denote the distance from a data point X_i to its closest centroid. C_1 is initialized as $X_{\text{rand}(1..n)}$. The variable i is selected by the function

Algorithm 2 k-means++

$C_1 \leftarrow$ RandomInit()
$j \leftarrow 2$
repeat
  $i \leftarrow$ RandomWeightedBySquaredDistance()
  $C_j \leftarrow X_i$
  $j \leftarrow j + 1$
until $j > k$
$C \leftarrow \text{kmeans}(C, k)$
output C

RandomWeightedBySquaredDistance() =

$$\begin{aligned} &\min \quad i \\ &\text{s.t.} \quad \frac{D(X_1)^2 + D(X_2)^2 + ... + D(X_i)^2}{D(X_1)^2 + D(X_2)^2 + ... + D(X_n)^2} > \text{rand}([0, 1[). \end{aligned} \tag{2.2}$$

As a result, new centers are added, most likely to the areas lacking centroids. k-Means++ also has a performance guarantee [14]

$$E[TSE] \leq 8(\ln k + 2)TSE_{OPT}. \tag{2.3}$$

X-means [19] splits clusters as long as the Bayesian information criterion (BIC) gives a lower value for the slit than for the non-slit cluster.

Global k-means [20] tries all points as candidate initial centroid locations, and performs k-means. It gives good results, but with slow speed.

For a comparison of results of several clustering algorithms, see the summary Chapter 9 of this thesis or [17].

3 Clustering by Analytic Functions

Data clustering is a combinatorial optimization problem. The publication **I** shows that clustering is also an optimization problem for an analytic function. The mean squared error, or in this case, the total squared error can be expressed as an analytic function. With an analytic function we benefit from the existence of standard optimization methods: the gradient of this function is calculated and the descent method is used to minimize the function.

The *MSE* and *TSE* values can be calculated when the data points and centroid locations are known. The process involves finding the nearest centroid for each data point. We write c_{ij} for the feature j of the centroid of cluster i. The squared error function can be written as

$$f(\bar{c}) = \sum_{u} \min_{i} \{ \sum_{j} (c_{ij} - x_{uj})^2 \}. \tag{3.1}$$

The min operation forces one to choose the nearest centroid for each data point. This function is not analytic because of the min operations. A question is whether we can express $f(\bar{c})$ as an analytic function which then could be given as input to a gradient-based optimization method. The answer is given in the following section.

3.1 FORMULATION OF THE METHOD

We write the p-norm as

$$\|\bar{x}\|_p = (\sum_{i=1}^{d} |x_i|^p)^{1/p}. \tag{3.2}$$

The maximum value of the x_i's can be expressed as

$$\max(|x_i|) = \lim_{p \to \infty} \|\bar{x}\|_p = \lim_{p \to \infty} (\sum_{i=1}^{n} |x_i|^p)^{1/p}. \tag{3.3}$$

Since we are interested in the *minimum* value, we take the inverses $\frac{1}{x_i}$ and find their maximum. Then another inverse is taken to obtain the minimum of the x_i:

$$\min(|x_i|) = \lim_{p\to\infty}(\sum_{i=1}^{d}\frac{1}{|x_i|^p})^{-1/p}. \tag{3.4}$$

3.2 ESTIMATION OF INFINITE POWER

Although calculations of the infinity norm ($p = \infty$) without comparison operations are not possible, we can estimate the exact value by setting p to a high value. The error of the estimate is

$$\epsilon = (\sum_{i=1}^{d}\frac{1}{|x_i|^p})^{-1/p} - \lim_{p_2\to\infty}(\sum_{i=1}^{d}\frac{1}{|x_i|^{p_2}})^{-1/p_2}. \tag{3.5}$$

The estimation can be made up to any accuracy, the estimation error being

$$|\epsilon| \geq 0.$$

To see how close we can come in practice, a mathematical software package Matlab run was made:

$$1/\text{nthroot}((1/x1)\text{^}p + (1/x2)\text{^}p, p).$$

For example, with the values $x1, x2 = 500$, $p = 100$ we got the result 496.54. When the values of $x1$ and $x2$ are far from each other, we get an accurate estimate, but when the numbers are close to each other, an approximation error is present.

3.3 ANALYTIC FORMULATION OF *TSE*

Combining (3.1) and (3.4) yields

$$f(\bar{c}) = \sum_u[\lim_{p\to\infty}((\sum_i\frac{1}{|\sum_j(c_{ij}-x_{uj})^2|^p})^{-1/p})]. \tag{3.6}$$

Proceeding from (3.6) by removing lim, we can now write $\hat{f}(\bar{c})$ as an estimator for $f(\bar{c})$:

$$\hat{f}(\bar{c}) = \sum_u [(\sum_i (\sum_j (c_{ij} - x_{uj})^2)^{-p})^{-\frac{1}{p}}]. \tag{3.7}$$

This is an analytic estimator, although the exact $f(\bar{c})$ cannot be written as an analytic function when the data points lie in the middle of cluster centroids in a certain way.

The partial derivatives and the gradient can also be calculated. The formula for partial derivatives is calculated using the chain rule:

$$\begin{aligned}\frac{\partial \hat{f}(\bar{c})}{\partial c_{st}} = &\sum_u [-\frac{1}{p} \cdot (\sum_i (\sum_j (c_{ij} - x_{uj})^2)^{-p})^{-\frac{p+1}{p}} \\ &\cdot \sum_i (-p \cdot (\sum_j (c_{ij} - x_{uj})^2)^{-(p+1)}) \cdot 2 \cdot (c_{st} - x_{ut})].\end{aligned} \tag{3.8}$$

3.4 TIME COMPLEXITY

The time complexity for calculating the estimator of the total squared error has been derived in paper **I** as

$$T(\hat{f}(\bar{c})) = O(n \cdot d \cdot k \cdot p). \tag{3.9}$$

The time complexity of calculating $\hat{f}(\bar{c})$ grows linearly with the number of data points n, dimensionality d, number of centroids k, and power p. The time complexity of calculating a partial derivative is

$$T(\text{partial derivative}) = O(n \cdot d \cdot k \cdot p).$$

The time complexity for calculating all partial derivatives, which is the same as the gradient, is

$$T(\text{all partial derivatives}) = O(n \cdot d \cdot k \cdot p).$$

This differs only by the factor p from one iteration time complexity of the k-means $O(k \cdot n \cdot d)$. In these time complexity calculations a result concerning the time complexity of calculation of the nth root is used [23].

3.5 ANALYTIC OPTIMIZATION OF *TSE*

Since we can calculate the values of $\hat{f}(\bar{c})$ and the gradient, we can find a (local) minimum of $\hat{f}(\bar{c})$ by the gradient descent method. In the gradient descent method, the solution points converge iteratively to a minimum:

$$\bar{c}_{i+1} = \bar{c}_i - \nabla \hat{f}(\bar{c}_i) \cdot l, \tag{3.10}$$

where l is the step length. The value of l can be calculated at every iteration, starting from some l_{max} and halving it recursively until $\hat{f}(\bar{c}_{i+1}) < \hat{f}(\bar{c}_i)$.

Equation (3.8) for the partial derivatives depends on p. For any $p \geq 0$, either a local or the global minimum of (3.7) is found. Setting p large enough, we get a satisfactory estimator $\hat{f}(\bar{c})$, although there is often some bias in this estimator and a p that is too small may lead to a different clustering result.

The analytic clustering method presented here corresponds to the k-means algorithm [11]. It can be used to obtain a local minimum of the squared error function similarly to k-means, or to simulate the random swap algorithm [18] by changing one cluster centroid randomly. In the random swap algorithm, a centroid and a datapoint are chosen randomly, and a trial movement of this centroid to this datapoint is made. If the k-means with the new centroid provide better results than the earlier solution, the centroid remains swapped. Such trial swaps are then repeated for a fixed number of times.

Analytic clustering and k-means work in the same way, although their implementations differ. Their step length is different. The difference in the clustering result also originates from the approximation of the ∞-norm by the p-norm.

We have used an approximation to the infinity norm to find the nearest centroids for the datapoints, and used the sum-of-squares for the distance metric. The infinity norm, on the other hand, could be used to cluster with the infinity norm distance metric. The Euclidean norm ($p = 2$) is normally used in the literature, but experiments with other norms are also published. For example, $p = 1$ gives the k-medians clustering, e.g. [24], and $p \to 0$ gives the categorical k-modes clustering. Papers on the k-midrange clustering (e.g. [25,26]) employ the infinity norm ($p = \infty$) in finding the range of a cluster. In [27] a $p = \infty$ formulation has been given for the more general fuzzy case. A description and comparison of different formulations has been given in [28]. With the infinity norm distance metric, the distance of a data point from a centroid is calculated by taking the dominant feature of the difference vector between the data point and the centroid. Our contribution in this regard is that we can form an analytic estimator for the cost function even if the distance metric were the infinity norm. This would make the formula for $\hat{f}(\bar{c})$ and the formula for the partial derivatives a somewhat more complicated but nevertheless possible.

The experimental results are illustrated in Table 3.1 and show that analytic clustering and k-means clustering provide comparable results.

Table 3.1: Averages of TSE *values of 30 runs of analytic and traditional methods. The TSE values are divided by* 10^{13} *or* 10^{6} *(wine set) or* 10^{4} *(breast set) or 1 (yeast set). Processing times in seconds for different datasets and methods.*

Dataset	Total squared error				Processing time			
	K-means		Random swap		K-means		Random swap	
	Anal.	Trad.	Anal.	Trad.	Anal.	Trad.	Anal.	Trad.
s1	1.93	1.91	1.37	1.39	4.73	0.04	52.46	0.36
s2	2.04	2.03	1.52	1.62	6.97	0.08	51.55	0.61
s3	1.89	1.91	1.76	1.78	4.59	0.06	59.03	0.58
s4	1.70	1.68	1.58	1.60	5.43	0.23	49.12	1.13
iris	22.22	22.22	22.22	22.22	0.12	0.01	0.48	0.03
thyroid	74.86	74.80	73.91	73.91	0.22	0.02	0.72	0.04
wine	2.41	2.43	2.37	2.37	0.44	0.02	4.39	0.04
breast	1.97	1.97	1.97	1.97	0.15	0.02	1.07	0.04
yeast	48.87	48.79	45.83	46.06	5.15	0.12	50.00	0.91

4 Clustering by Gradual Data Transformation

The traditional approach to clustering is to fit a model (partition or prototypes) to the given data. In publication **II** we propose a completely opposite approach: fitting the data to a given clustering model that is optimal for similar pathological (not normal) data of equal size and dimensions. We then perform an inverse transform from this pathological data back to the original data while refining the optimal clustering structure during the process. The key idea is that we do not need to find an optimal global allocation of the prototypes. Instead, we only need to perform local fine-tuning of the clustering prototypes during the transformation in order to preserve the already optimal clustering structure.

We first generate an artificial data X^* of the same size (n) and dimension (d) as the input data, so that the data vectors are divided into k perfectly separated clusters without any variation. We then perform a one-to-one bijective mapping of the input data to the artificial data ($X \rightarrow X^*$).

The key point is that we already have a clustering that is optimal for the artificial data, but not for the real data. In the next step, we perform an inverse transform of the artificial data back to the original data by a sequence of gradual changes. While doing this, the clustering model is updated after each change by k-means. If the changes are small, the data vectors will gradually move to their original position without breaking the clustering structure. The details of the algorithm including the pseudocode are given in Section 4.1. An online animator demonstrating the progress of the algorithm is available at `http://cs.uef.fi/sipu/clustering/animator/`. The animation starts when "Gradual k-means" is chosen from the menu.

The main design problems of this approach are to find a suitable artificial data structure, how to perform the mapping, and how to control the inverse transformation. We will demonstrate next that the proposed approach works with simple design choices, and overcomes the locality problem of *k*-means. It cannot be proven to provide optimal results every time, as there are bad cases where it fails to find the optimal solution. Nevertheless, we show by experiments that the method is significantly better than *k*-means and *k*-means++, and competes equally with repeated *k*-means. Also, it is rare that it ends up with a bad solution as is typical to *k*-means.

Experiments will show that only a few transformation steps are needed to obtain a good quality clustering.

4.1 DATA INITIALIZATION

In the following subsections, we will go through the phases of the algorithm. For the pseudocode, see Algorithm 3. We call this algorithm *k*-means*, because of the repeated use of *k*-means. However, instead of applying *k*-means to the original data points, we create another artificial data set which is prearranged into *k* clearly separated zero-variance clusters.

The algorithm starts by choosing the artificial clustering structure and then dividing the artificial data points among these equally. We do this by creating a new dataset X_2 and by assigning each data point in the original dataset X_1 to a corresponding data point in X_2. We consider seven different structures for the initialization:

- line
- diagonal
- random
- random with optimal partition
- initialization used in *k*-means++
- line with uneven clusters
- point.

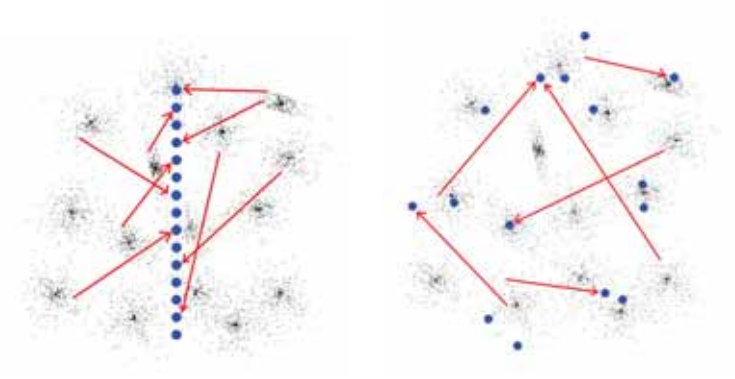


Figure 4.1: Original dataset and line init (left) or random init (right) with sample mappings shown by arrows.

In the *line structure*, the clusters are arranged along a line. The k locations are set as the middle value of the range in each dimension, except the last dimension where the k clusters are distributed uniformly along the line, see Figure 4.1 (left) and the animator `http://cs.uef.fi/sipu/clustering/animator/`. The range of 10% nearest to the borders are left without clusters.

In the *diagonal structure*, the k locations are set uniformly to the diagonal of the range of the dataset.

In the *random structure*, the initial clusters are selected randomly from among the data point locations in the original dataset, see Figure 4.1 (right). In these structuring strategies, data point locations are initialized randomly to these cluster locations. Even distribution among the clusters is a natural choice. To further justify this, lower cardinality clusters could more easily become empty later, which was an undesirable situation.

The fourth structure is *random locations* but using *optimal partitions* for the mapping. This means assigning the data points to the nearest clusters.

The fifth structure corresponds to the initialization strategy used in *k-means++* [14].

The sixth structure is the line with *uneven clusters*, in which we

place twice as many points at the most centrally located half of the cluster locations than at the other locations.

The seventh structure is the *point*. It is like the line structure but we put the clusters in a very short line, which, in a larger scale, looks like a single point. In this way, the dataset "explodes" from a single point during the inverse transform. This structure is useful mainly for the visualization purposes in the web-animator.

The *k*-means++-style structure with evenly distributed data points is the recommended structure because it works best in practice, and therefore we use it inthe further experiments. In choosing the structure, good results are achieved when there is a notable separation between the clusters and evenly distributed data points in the clusters.

Once the initial structure has been chosen, each data point in the original data set is assigned to a corresponding data point in the initial structure. The data points in this manually created data set are randomly but evenly located.

4.2 INVERSE TRANSFORMATION STEPS

The algorithm proceeds by executing a given number (> 1) of inverse transformation *steps* given as a user-set integer parameter. The default value for *steps* is 20. At each step, all data points are transformed towards their original location by the amount

$$\frac{1}{steps} \cdot (X_{1,i} - X_{2,i}), \tag{4.1}$$

where $X_{1,i}$ is the location of the ith datapoint in the original data and $X_{2,i}$ is its location in the artificial structure. After every transform, *k*-means is executed given the previous centroids along with the modified dataset as input. After all the steps have been completed, the resulting set of centroids C is output.

It is possible that two points that belong to the same cluster in the final dataset will be put into different clusters in the artificially created dataset. Then they smoothly move to their final locations during the inverse transform.

Table 4.1: Time complexity of the k-means algorithm.*

	Theoretical			
	k free	$k = O(n)$	$k = O(\sqrt{n})$	$k = O(1)$
Initialization	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Data set transform	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Empty clusters removal	$O(kn)$	$O(n^2)$	$O(n^{1.5})$	$O(n)$
k-means	$O(kn^{kd+1})$	$O(n^{O(n)\cdot d+2})$	$O(n^{O(\sqrt{n}d+\frac{3}{2})})$	$O(n^{kd+1})$
Algorithm total	$O(kn^{kd+1})$	$O(n^{O(n)\cdot d+2})$	$O(n^{O(\sqrt{n}d+\frac{3}{2})})$	$O(n^{kd+1})$
	Fixed k-means			
	k free	$k = O(n)$	$k = O(\sqrt{n})$	$k = O(1)$
Initialization	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Data set transform	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Empty clusters removal	$O(kn)$	$O(n^2)$	$O(n^{1.5})$	$O(n)$
k-means	$O(kn)$	$O(n^2)$	$O(n^{1.5})$	$O(n)$
Algorithm total	$O(kn)$	$O(n^2)$	$O(n^{1.5})$	$O(n)$

4.3 TIME COMPLEXITY

The worst case complexities of the phases are listed in Table 4.1. The overall time complexity is not more than for the k-means, see Table 4.1.

4.4 EXPERIMENTAL RESULTS

We ran the algorithm with different values of steps and for several data sets. For the *MSE* calculation we use the formula

$$MSE = \frac{\sum_{j=1}^{k} \sum_{X_i \in C_j} || X_i - C_j ||^2}{n \cdot d},$$

where MSE is normalized by the number of features in the data. All the datasets can be found on the SIPU web page [29].

Algorithm 3 *k*-means*
Input: data set X_1, number of clusters k, *steps*,
Output: Codebook C.

$n \leftarrow size(X_1)$
$[X_2, C] \leftarrow Initialize()$
for repeats = 1 to steps **do**
  for i = 1 to n **do**
    $X_{3,i} \leftarrow X_{2,i} + (repeats/steps) * (X_{1,i} - X_{2,i})$
  end for
  $C \leftarrow kmeans(X_3, k, C)$
end for
output C

The sets s1, s2, s3 and s4 are artificial datasets consisting of Gaussian clusters with same variance but increasing overlap. Given 15 seeds, data points are randomly generated around them. In a1 and DIM sets, the clusters are clearly separated, whereas in s1-s4 they are overlap more. These sets are chosen because they are still easy enough for a good algorithm to find the clusters correctly but hard enough for a bad algorithm to fail. The results for the number of steps 2-20 are plotted in Figure 4.2.

We observe that 20 steps is enough for k-means* (Figure 4.2). Many clustering results of these data sets stabilize at around 6 steps. More steps give only a marginal additional benefit, but at the cost of a longer execution time. For some of the data sets, even just one step gives the best result. In these cases, initial positions for centroids just happened to be good.

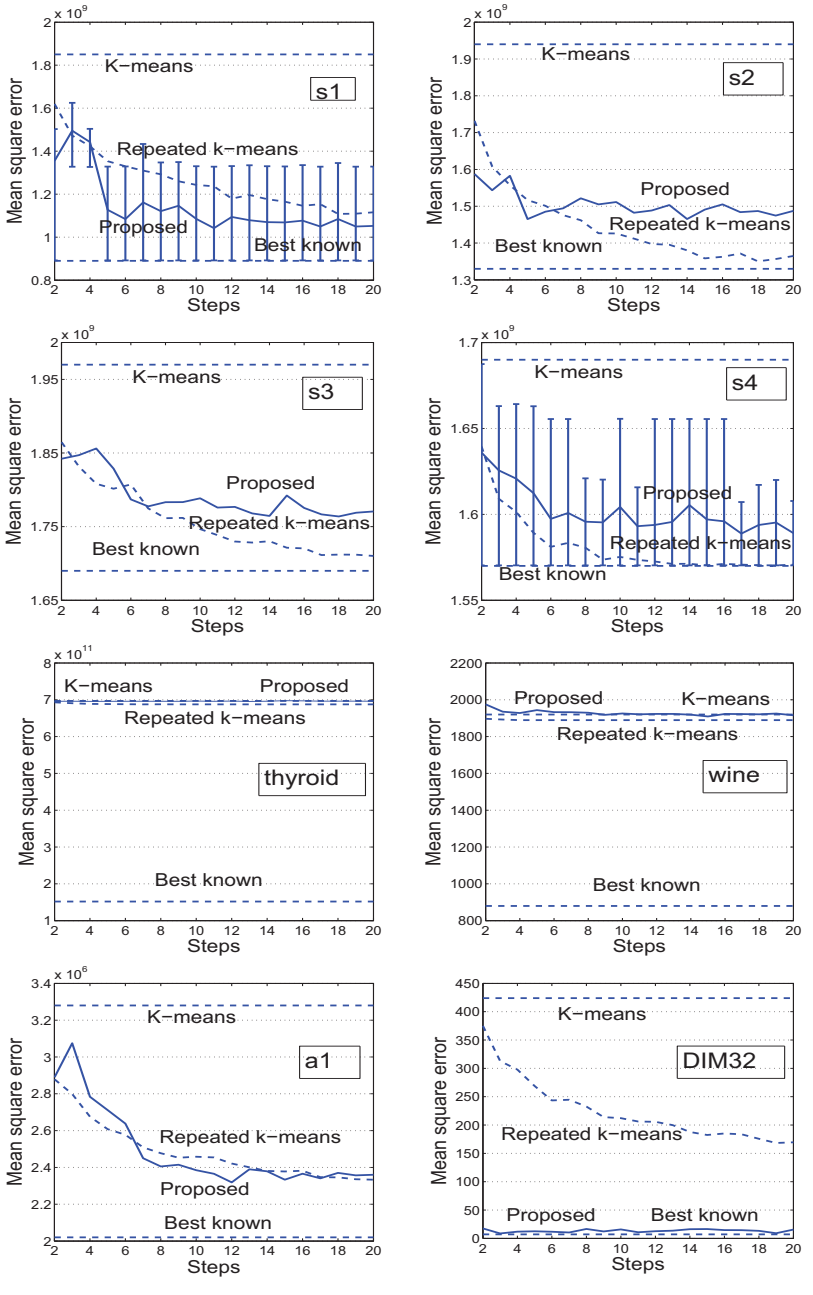


Figure 4.2: Results of k-means (average over 200 runs) for datasets s1, s2, s3, s4, thyroid, wine, a1 and DIM32 with different numbers of steps. For repeated k-means there are an equal number of repeats as there are steps in the proposed algorithm. For s1 and s4, the 75% error bounds are also shown. We observe that 20 steps is enough for this algorithm.*

5 All-Pairwise Squared Distances as Cost

All-pairwise squared distances has been used as a cost function in clustering [30, 31]. In publication **III**, we showed that it leads to more balanced clustering than centroid-based distance functions as in k-means. Clustering by all-pairwise squared distances is formulated as a cut-based method, and it is closely related to the MAX k-CUT method. We introduce two algorithms for the problem, both of which are faster than the existing one based on l_2^2-Stirling approximation. The first algorithm uses semidefinite programming as in MAX k-CUT. The second algorithm is an on-line variant of classical k-means. We show by experiments that the proposed approach provides better overall joint optimization of the mean squared error and cluster balance than the compared methods.

5.1 BALANCED CLUSTERING

A balanced clustering is defined as a clustering where the points are evenly distributed into the clusters. In other words, every cluster includes either $\lfloor n/k \rfloor$ or $\lceil n/k \rceil$ points. We define balanced clustering as a problem which aims at maximizing the balance and minimizing some other cost function, such as *MSE*. Balanced clustering is desirable in workload-balancing algorithms. For example, one algorithm for the multiple traveling salesman problem [32] clusters the cities so that each cluster is solved by one salesman. It is desirable that each salesman has an equal workload.

Balanced clustering, in general, is a 2-objective optimization problem, in which two aims contradict each other: to minimize a cost function such as *MSE*, and to balance cluster sizes at the same time. Traditional clustering aims at minimizing *MSE* completely without considering cluster size balance. Balancing, on the

Table 5.1: Classification of some balanced clustering algorithms.

Balance-constrained	Type
Balanced *k*-means (publication **IV**)	*k*-means
Constrained *k*-means [33]	*k*-means
Size constrained [34]	integer linear programming
Balance-driven	**Type**
Scut (publication **III**)	on-line *k*-means
FSCL [35]	assignment
FSCL additive bias [36]	assignment
Cluster sampled data [37]	*k*-means
Ratio cut [38]	divisive
Ncut [39]	divisive
Mcut [40]	divisive
SRcut [41]	divisive
Submodular fractional programming [42]	submodular fractional programming

other hand, would be trivial if we did not care about *MSE*: Then we would simply divide the vectors into equal size clusters randomly. For optimizing both, there are two approaches: *balance-constrained* and *balance-driven* clustering.

In balance-constrained clustering, cluster size balance is a mandatory requirement that must be met, and minimizing *MSE* is a secondary criterion. In balance-driven clustering, balanced clustering is an aim, but it is not mandatory. It is a compromise between the two goals: balance and the *MSE*. The solution is a weighted cost function between *MSE* and the balance, or it is a heuristic, that aims at minimizing *MSE* but indirectly creates a more balanced result than optimizing *MSE* alone.

Existing algorithms for balanced clustering are grouped into these two classes in Table 5.1. As more application-specific approaches, networking uses balanced clustering to obtain some desirable goals [43, 44].

5.2 CUT-BASED METHODS

Cut-based clustering is a process where the dataset is cut into smaller parts based on the similarity $S(X_l, X_s)$ or the cost $d(X_l, X_s)$ between pairs of points. By cut(A, B) one means partitioning a dataset into two parts A and B, and the value of cut(A, B) is the total weight between all pairs of points between the sets A and B:

$$\text{cut}(A, B) = \sum_{X_l \in A, X_s \in B} w_{ls}. \tag{5.1}$$

The weights w can be defined either as distances or similarities between the two points. Unless otherwise noted, we use (squared) Euclidean distances in publication **III**. The cut(A, B) equals the total pairwise weights of $A \cup B$ subtracted by the pairwise weights within the parts A and B:

$$\text{cut}(A, B) = W - W(A) - W(B), \tag{5.2}$$

where

$$W = \sum_{l=1}^{n} \sum_{s=1}^{n} w_{ls}, \tag{5.3}$$

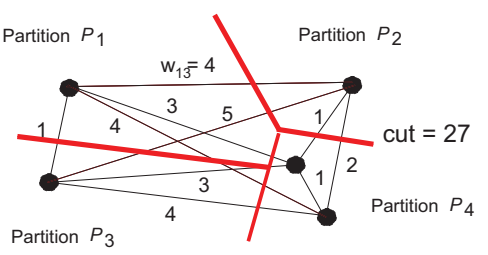
and

$$W(A) = \sum_{X_l \in A, X_s \in A} w_{ls}, \tag{5.4}$$

and $W(B)$ is defined respectively. In cut-based clustering, two common objective functions are *Ratio cut* [38] and *Normalized cut* (Ncut, for short) [39]. Both of these methods favor balanced clustering [45]. In practice, one approximates these problems by relaxation, i.e., solving a nearby easier problem. Relaxing Ncut leads to normalised spectral clustering, while relaxing RatioCut leads to unnormalised spectral clustering [45]. There exists also a semidefinite-programming based relaxation for Ncut [46].

5.3 MAX K-CUT METHOD

In the weighted MAX k-CUT problem [47], one partitions a graph into k subgraphs so that the sum of the weights of the edges between the subgraphs is maximised. The weights are distances.



$$\begin{aligned}
&\text{cut}(P_1, \bar{P}_1) = 12\\
&\text{cut}(P_2, \bar{P}_2) = 12\\
&\text{cut}(P_3, \bar{P}_3) = 13\\
&\text{cut}(P_4, \bar{P}_4) = 17\\
&\textstyle\sum = 54.\\
&1/2 \cdot 54 = 27
\end{aligned}$$

Figure 5.1: An example of MAX k-CUT, when $k = 4$.

MAX k-CUT aims at partitioning the data into k clusters $P_1, ..., P_k$. Following the notation of Section 5.2 and inserting a factor $1/2$ in order to avoid summing the weights twice, the MAX k-CUT problem is defined as

$$\max_{P_j, 1 \le j \le k} \frac{1}{2} \sum_{j=1}^{k} \text{cut}(P_j, \bar{P}_j). \tag{5.5}$$

There is an example of MAX k-CUT in Figure 5.1. MAX k-CUT is an NP-hard problem [48] for general weights.

If we use Euclidean distance for the weights of the edges between every pair of points, then taking optimal weighted MAX k-CUT results in the minimum intra-cluster pairwise distances among any k-CUT. If we use *squared* distances as weights of the edges, we end up with minimum intra-cluster pairwise squared distances. If we use squared Euclidean distances as weights, the problem is expected to remain NP-hard.

5.4 SQUARED CUT (SCUT)

Publication **III** deals with the *Squared cut, Scut* method, which uses all pairwise squared distances as the cost function. This cost function has been presented in [49], where it is called l_2^2 k-clustering. However, we formulate it by using the TSE's of the clusters and show that the method leads to a more balanced clustering problem than TSE itself. It is formulated as a cut-based method and it resembles the MAX k-CUT method [30]. We present two algo-

rithms for the problem; both more practical than the exhaustive search proposed in [31] for l_2^2 k-clustering. The first algorithm is based on *semidefinite programming*, similar to MAX k-CUT, and the second one is an *on-line k-means* algorithm directly optimizing the cost function.

A general k-clustering problem by Sahni and Gonzales [30] defines the cost by calculating all pairwise distances within the clusters for any arbitrary weighted graphs. Guttmann-Beck and Hassin [50] studies the problem when the distances satisfy the triangle inequality. Schulman [49] gives probabilistic algorithms for l_2^2 k-clustering [30]. The running time is linear if the dimension d is of the order $o(\log n / \log\log n)$ but, otherwise, it is $n^{O(\log\log n)}$. De la Vega et al. [31] improved and extended Schulman's result, giving a true polynomial time approximation algorithm for arbitrary dimension. However, even their algorithm is slow in practice. We therefore present faster algorithms for the Scut method.

In Scut, we form the graph by assigning squared Euclidean distances as the weights of the edges between every pair of points. In a single cluster j, the intra-cluster pairwise squared distances are of the form $n_j \cdot TSE_j$, where n_j is the number of points in cluster j [51], p. 52. The generalisation of this to all clusters is known as *Huygens's theorem*, which states that the total squared error (TSE) equals the sum over all clusters, over all squared distances between pairs of entities within that cluster divided by its cardinality:

$$W(A_j) = n_{A_j} \cdot TSE(A_j) \quad \text{for all } j.$$

Huygens's theorem is crucial for our method, because it relates the pairwise distances to the intra-cluster TSE, and thus, to the Scut cost function:

$$\text{Scut} = n_1 \cdot TSE_1 + n_2 \cdot TSE_2 + ... + n_k \cdot TSE_k, \tag{5.6}$$

where n_j is the number of points and TSE_j is the total squared error of the jth cluster. Based on (1.8), this may also be written as

$$\text{Scut} = n_1^2 \cdot MSE_1 + n_2^2 \cdot MSE_2 + ... + n_k^2 \cdot MSE_k, \tag{5.7}$$

Algorithm 4 Scut
Input: dataset X, number of clusters k
Output: partitioning of points P

for each edge of the graph **do**
  Weight of edge $w_{ij} \leftarrow$ Euclidean_distance$(X_i, X_j)^2$
end for
Approximate MAX k-CUT.
Output partitioning of points P.

Figure 5.2: Two different sized clusters with the same MSE.

where MSE_j is the mean squared error of the jth cluster. In cut-notation the cost function is total pairwise weights minus the value of MAX k-CUT:

$$\text{Scut} = W - \max_{P_j, 1 \le j \le k} \frac{1}{2} \sum_{i=1}^{k} \text{cut}(P_j, \bar{P}_j). \tag{5.8}$$

From this we conclude that using squared distances and optimizing MAX k-CUT results in the optimization of the Scut cost function (5.6). For approximating Scut, the Algorithm 4 can be used. Our cut-based method has an MSE-based cost function and it tends to balance the clusters because of the n_j^2 factors in (5.7). This can be seen by the following simple example where two clusters have the same squared error: $MSE_1 = MSE_2 = MSE$ (Figure 5.2). The total errors of these are $2^2 \cdot MSE_1 = 4 \cdot MSE$, and $10^2 \cdot MSE_2 = 100 \cdot MSE$. Adding one more point would increase the error by

$(n+1)^2 \cdot MSE - n^2 \cdot MSE = (2n+1) \cdot MSE$. In the example in Figure 5.2, the cost would increase by $5 \cdot MSE$ (cluster 1) and $21 \cdot MSE$ (cluster 2). The cost function therefore always favors putting points into a smaller cluster, and therefore, it tends to make more balanced clusters. Figure 5.3 demonstrates the calculation of the cost.

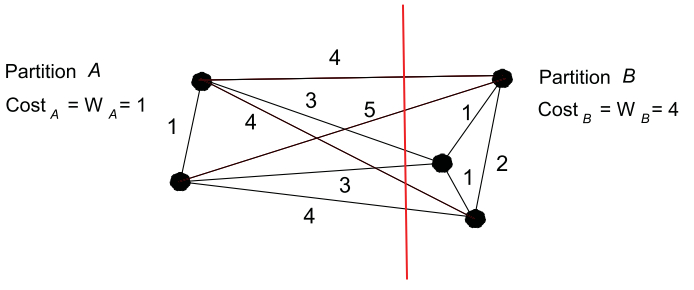


Figure 5.3: Calculation of the cost. Edge weights are squared Euclidean distances.

5.5 APPROXIMATING SCUT

5.5.1 Approximation algorithms

Weighted MAX k-CUT is an NP-hard problem but it can be solved by an approximation algorithm based on *semidefinite programming* (SDP) in polynomial time [47]. Although polynomial, the algorithm is slow. According to our experiments, it can only be used for datasets with just over 150 points. A faster approximation algorithm has been presented by Zhu and Guo [48]. It begins with an arbitrary partitioning of the points, and moves a point from one subset to another if the sum of the weights of edges across different subsets decreases. The algorithm stops when no further improvements can be attained. In subection 5.5.2, we will propose an even faster algorithm, which instead of maximising MAX k-CUT minimizes the Scut cost function (5.6). Nevertheless, the result will be the same.

Algorithm 5 Fast approximation algorithm (on-line *k*-means) for Scut
Input: dataset X, number of clusters *k*, number of points *n*
Output: partitioning of points P

```
Create some initial partitioning P.
changed ← TRUE
while changed do
  changed ← FALSE
  for i = 1 to n do
    for l = 1 to k do
      if ΔScut < 0 then
        move point i to the cluster l
        update centroids and TSE's of previous cluster and cluster l
        changed ← TRUE
      end if
    end for
  end for
end while
Output partitioning of points P.
```

5.5.2 Fast Approximation Algorithm for Scut

We next define an on-line *k*-means variant of the Scut method. In the algorithm, the points are repeatedly re-partitioned to the cluster which provides the lowest value for the Scut cost function. The partition of the points is done one-by-one, and a change of cluster will cause an immediate update of the two clusters affected (their centroid and size). We use the fact that calculating the pairwise total squared distance within clusters is the same as calculating the Scut cost function in *TSE* form (5.6). We next derive a fast $O(1)$ update formula which calculates the cost function change when a point is moved from one cluster to another. We keep on moving points to other clusters as long as the cost function decreases, see Algorithm 5. The approximation ratio derived in publication **III**, is

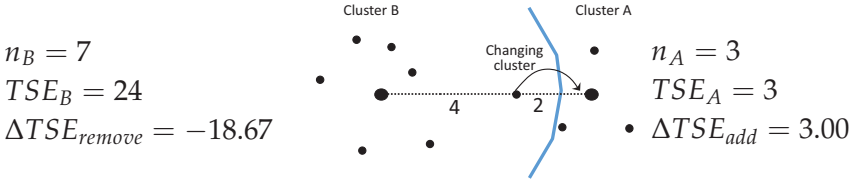


Figure 5.4: Changing point from cluster B to A decreasing cost by 121.02.

$$\begin{aligned}\epsilon_k &= \frac{W - w(\mathrm{P}(k))}{W - w(\mathrm{P}(k)^*)} \\ &= \frac{W - w(\mathrm{P}(k))}{\max(0, W - \frac{1}{\alpha_k} \cdot w(\mathrm{P}(k)))} \quad , \end{aligned} \tag{5.9}$$

where W is all pairwise weights, $w(\mathrm{P}(k))$ is cut by the approximation algorithm, $w(\mathrm{P}(k)^*)$ is optimal cut and $\alpha_k > 1 - k^{-1}$. The update formula follows the merge cost in the agglomerative clustering algorithm [6]. It includes the change in TSE when adding a point, the change in TSE when removing a point, and the overall cost in terms of the cost function (5.6). The costs are obtained as follows:
Addition:

$$\Delta TSE_{add} = \frac{n_A}{n_A + 1} \cdot ||C_A - X_i||^2. \tag{5.10}$$

Removal:

$$\begin{aligned}\Delta TSE_{remove} &= -\frac{n_B - 1}{n_B} \cdot ||\frac{n_B}{n_B - 1} \cdot C_B - \frac{1}{n_B - 1} \cdot X_i - X_i||^2 \\ &= -\frac{n_B - 1}{n_B} ||\frac{n_B}{n_B - 1} \cdot C_B - \frac{n_B}{n_B - 1} \cdot X_i||^2 \\ &= -\frac{n_B}{n_B - 1} \cdot ||C_B - X_i||^2. \end{aligned} \tag{5.11}$$

The total cost of clusters A and B before the move is

$$\mathrm{Scut}_{before} = n_A \cdot TSE_A + n_B \cdot TSE_B, \tag{5.12}$$

where n_A and n_B are the number of points in the clusters A and B before the operation, C_A and C_B are the centroid locations before

the tentative move operation and X_i is the data point involved in the operation. The total cost after the move is

$$\begin{aligned}\text{Scut}_{after} &= (n_A + 1) \cdot (TSE_A + \Delta TSE_{add}) \\ &\quad + (n_B - 1) \cdot (TSE_B + \Delta TSE_{remove}).\end{aligned} \tag{5.13}$$

From these we get the change in cost

$$\Delta\text{Scut} = \text{Scut}_{after} - \text{Scut}_{before} \tag{5.14}$$

$$= TSE_A - TSE_B + (n_A + 1) \cdot \Delta TSE_{add} + (n_B - 1) \cdot \Delta TSE_{remove}, \tag{5.15}$$

$$= TSE_A - TSE_B + (n_A + 1) \cdot \frac{n_A}{n_A + 1} \cdot ||C_A - X_i||^2 \tag{5.16}$$

$$+ (n_B - 1) \cdot -\frac{n_B}{n_B - 1} \cdot ||C_B - X_i||^2. \tag{5.17}$$

See an example of a point changing its cluster in Figure 5.4, where the changes in the TSEs are the following: $\Delta TSE_{add} = 3/4 \cdot 2^2 = 3.00$ and $\Delta TSE_{remove} = -7/6 \cdot 4^2 = -18.67$. In Figure 5.4, the change in cost function is $\Delta\text{Scut} = 3 - 24 + (3 + 1) \cdot 3 + (7 - 1) \cdot -18.67 = -121.02$.

5.6 EXPERIMENTS

To solve the semidefinite program instances, we use the SeDuMi solver [52] and the Yalmip modelling language [53]. We use datasets from SIPU [29]. To compare how close the obtained clustering is to balance-constrained clustering (an equal distribution of sizes $\lceil n/k \rceil$), we measure the balance by calculating the difference in the cluster sizes and a balanced n/k distribution, calculated by

$$2 \cdot \sum_j \max(n_j - \lceil \frac{n}{k} \rceil, 0). \tag{5.18}$$

. We first compare Scut with the SDP algorithm against repeated k-means. The best results of 100 repeats (lowest distances) are chosen. In the SDP algorithm we repeat only the point assignment phase.

Table 5.2: Balances and execution times of the proposed Scut method with the SDP algorithm and k-means clustering. 100 repeats, in the SDP algorithm only the point assignment phase is repeated.

Dataset	n	k	balance		time	
			repeated Scut	repeated k-means	repeated Scut	repeated k-means
iris	150	3	2	6	8h 25min	0.50s
SUBSAMPLES:						
s1	150	15	42	30	9h 35min	0.70s
s1	50	3	2	6	34s	0.44s
s1	50	2	0	8	28s	0.34s
s2	150	15	48	24	6h 50min	0.76s
s2	50	3	2	4	27s	0.40s
s2	50	2	0	4	32s	0.38s
s3	150	15	44	28	7h 46min	0.89s
s3	50	3	2	6	31s	0.43s
s3	50	2	0	2	26s	0.41s
s4	150	15	40	30	7h 01min	0.93s
s4	50	3	0	6	28s	0.42s
s4	50	2	0	0	30s	0.36s
a1	50	20	4	4	11s	0.45s
DIM32	50	16	0	6	8s	0.46s
iris	50	3	0	10	33s	0.44s
thyroid	50	2	0	28	28s	0.38s
wine	50	3	2	6	30s	0.40s
breast	50	2	2	34	18s	0.35s
yeast_times100	50	10	8	8	10s	0.48s
glass	50	7	6	6	9s	0.44s
wdbc	50	2	0	20	11s	0.28s
best			**14 times**	**4 times**		

Table 5.3: Best balances and total execution times of the proposed Scut with the fast approximation algorithm and k-means clustering for 100 runs.

Dataset	n	k	balance		time	
			Scut-fast	repeated *k*-means	Scut-fast	repeated *k*-means
s1	5000	15	180	184	4min	2.3s
s2	5000	15	160	172	4min	4.0s
s3	5000	15	260	338	5min	3.6s
s4	5000	15	392	458	6min	7.0s
a1	3000	20	36	40	5min	3.2s
DIM32	1024	16	0	0	42s	2.6s
iris	150	3	4	6	0.9s	0.4s
thyroid	215	2	126	168	1.0s	0.3s
wine	178	3	22	22	0.8s	0.3s
breast	699	2	216	230	1.3s	0.3s
yeast_times100	1484	10	298	362	1min 21s	4.2s
glass	214	7	110	106	4.6s	1.1s
wdbc	569	2	546	546	0.9s	0.4s

The results in Table 5.2 show that 64% of the clustering results are more balanced with the proposed method than with the repeated *k*-means method. They were equally balanced in 18% of the cases, and in the remaining 18% of the cases a *k*-means result was more balanced. Optimization works well with small datasets (systematically better than *k*-means) but with bigger datasets the benefit is smaller. The time complexity is polynomial, but the computing time increases quickly when the number of points increases. With 50 points, the computing time is approximately 20 s, but with 150 points it is approximately 7 hours. The memory requirement for 150 points is 4.4 GB. The results in Table 5.3 are for the fast online *k*-means algorithm, for which we can use bigger datasets. In 9 cases the repeated Scut gave better result than repeated *k*-means, in 3 cases it was equal and in 1 case it was worse.

6 Balance-constrained Clustering

Table 5.1 lists some balance-constrained clustering algorithms. We review them here.

Bradley et al. [33] and Demiriz et al. [54] present a *constrained k-means algorithm*, which is like k-means, but the assignment step is implemented as a linear program, in which the minimum number of points τ_h of clusters can be set as parameters. Setting $\tau_h = \lfloor n/k \rfloor$ gives balance-constrained clustering. The constrained k-means clustering algorithm works as follows:

Given m points in $\mathbb{R}^n$, minimum cluster membership values $\tau_h \geq 0, h = 1, ..., k$ and cluster centers $C_1^{(t)}, C_2^{(t)}, ..., C_k^{(t)}$ at iteration t, compute $C_1^{(t+1)}, C_2^{(t+1)}, ..., C_k^{(t+1)}$ at iteration $t+1$ using the following two steps:

Cluster Assignment. Let $T_{i,h}^t$ be a solution to the following linear program with $C_h^{(t)}$ fixed:

$$\text{minimize}_T \sum_{i=1}^{m}\sum_{h=1}^{k} T_{i,h} \cdot (\frac{1}{2}||X_i - C_h^{(t)}||_2^2) \tag{6.1}$$

$$\text{subject to} \sum_{i=1}^{m} T_{i,h} \geq \tau_h, h = 1, ..., k \tag{6.2}$$

$$\sum_{h=1}^{k} T_{i,h} = 1, i = 1, ..., m \tag{6.3}$$

$$T_{i,h} \geq 0, i = 1, ..., m, h = 1, ..., k. \tag{6.4}$$

Cluster Update.

$$C_h^{(t+1)} = \begin{cases} \frac{\sum_{i=1}^{m} T_{i,h}^{(t)} X_i}{\sum_{i=1}^{m} T_{i,h}^{(t)}} & \text{if} \quad \sum_{i=1}^{m} T_{i,h}^{(t)} > 0, \\ C_h^{(t)} & \text{otherwise.} \end{cases}$$

These steps are repeated until $C_h^{(t+1)} = C_h^{(t)}$, for all $h = 1, ..., k$.

The algorithm terminates in a finite number of iterations at a partitioning that is locally optimal [33]. At each iteration, the cluster assignment step cannot increase the objective function of constrained k-means (3) in [33]. The cluster update step either strictly decreases the value of the objective function or the algorithm terminates. Since there are a finite number of ways to assign m points to k clusters such that cluster h has at least τ_h points, constrained k-means algorithm does not permit repeated assignments, and the objective of constrained k-means (3) in [33] is strictly nonincreasing and bounded below by zero, the algorithm must terminate at some cluster assignment that is locally optimal.

Zhu et al. [34] try to find a partition close to the given partition, but such that the cluster size constraints are fulfilled.

In publication **IV**, we formulate balanced k-means algorithm; it belongs to the balance-constrained clustering category. It is otherwise the same as standard k-means but it guarantees balanced cluster sizes. It is also a special case of constrained k-means, where

cluster sizes are set equal. However, instead of using linear programming in the assignment phase, we formulate the partitioning as a pairing problem [55], which can be solved optimally by the Hungarian algorithm in $O(n^3)$ time.

6.1 BALANCED *K*-MEANS

To describe the balanced k-means algorithm, we need to define what is an assignment problem. The formal definition of an assignment problem (or linear assignment problem) is as follows. Suppose given two sets (A and S), of equal size, and a weight $w_{a,i}, a \in A, i \in S$, the goal is to find a bijection $f : A \to S$ so that the cost function

$$\text{Cost} = \sum_{a \in A} w_{a,f(a)}$$

is minimized. In the proposed algorithm, A corresponds to the cluster slots and S to the data points, see Figure 6.1.

In balanced k-means, we proceed as in the common k-means, but the assignment phase is different: instead of selecting the nearest centroids, we have n pre-allocated slots (n/k slots per cluster), and datapoints can be assigned only to these slots, see Figure 6.1. This will force all clusters to be of same size, assuming that $\lceil n/k \rceil = \lfloor n/k \rfloor = n/k$. Otherwise, there will be ($n \mod k$) clusters of size $\lceil n/k \rceil$, and $k - (n \mod k)$ clusters of size $\lfloor n/k \rfloor$.

To find an assignment that minimizes the *MSE*, we use the Hungarian algorithm [55]. First we construct a bipartite graph consisting of n datapoints and n cluster slots, see Figure 6.2. We then partition the cluster slots into clusters of as even number of slots as possible.

We generate centroid locations to the partitioned cluster slots, one centroid to each cluster. The initial centroid locations can be drawn randomly from all data points. The edge weight is the squared distance from the point to the cluster centroid it is assigned to. Unlike the standard assignment problem with fixed weights, here the weights dynamically change after each k-means iteration

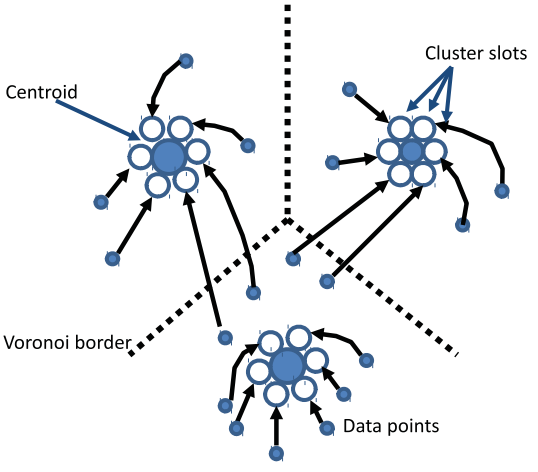


Figure 6.1: Assigning points to centroids via cluster slots.

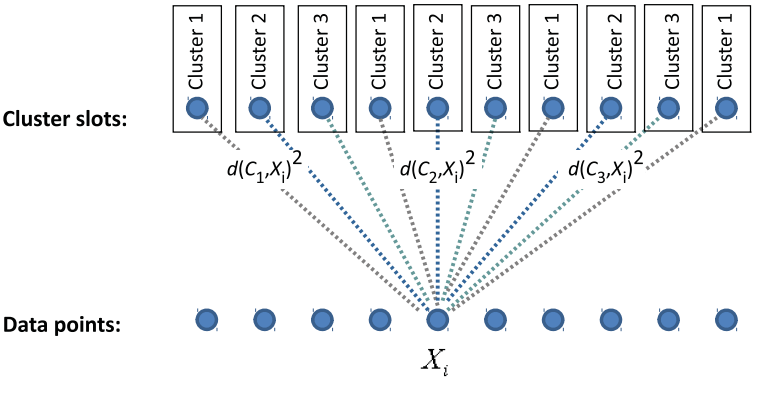


Figure 6.2: Minimum MSE calculation with balanced clusters. Modeling with bipartite graph.

according to the newly calculated centroids. After this, we perform the Hungarian algorithm to get the minimal weight pairing. The squared distances are stored in an $n \times n$ matrix, for the needs of the Hungarian algorithm. The update step is similar to that of k-means, where the new centroids are calculated as the means of the data points assigned to each cluster:

$$C_j^{(t+1)} = \frac{1}{n_j} \cdot \sum_{X_i \in C_j^{(t)}} X_i. \tag{6.5}$$

The weights of the edges are updated immediately after the update step. The pseudocode is in Algorithm 6. In the calculation of the edge weights, the index of the cluster slot is denoted by a and mod is used to calculate to which cluster a slot belongs (index = a mod k). The edge weights are calculated by

$$w_{a,i} = dist(X_i, C^t_{(a \bmod k)+1})^2, \tag{6.6}$$

for each cluster slot a and point i. The resulting partition of points X_i, $i \in [1, n]$, is

$$X_{f(a)} \in P_{(a \bmod k)+1}. \tag{6.7}$$

Algorithm 6 Balanced k-means
Input: dataset X, number of clusters k
Output: partitioning of dataset.

Initialize centroid locations C^0.
$t \leftarrow 0$
repeat
  Assignment step:
    Calculate edge weights.
    Solve an assignment problem.
  Update step:
    Calculate new centroid locations C^{t+1}
  $t \leftarrow t + 1$
until centroid locations do not change.
Output partitioning.

The convergence result for the constrained k-means in the beginning of this chapter applies to balanced k-means as well, since the linear programming in constrained k-means and the pairing in balanced k-means do essentially the same thing when the parameters are suitably set. We can express the convergence result principle as follows.

1. The result never gets worse
2. The algorithm ends when the result does not get better.

We consider the assignment step to be optimal with respect to MSE because of pairing and the update step to be optimal, because MSE is clusterwise minimized as is in k-means.

6.2 TIME COMPLEXITY

The time complexity of the assignment step in k-means is $O(k \cdot n)$. Constrained k-means involves linear programming. It takes $O(v^{3.5})$ time, where v is the number of variables, by Karmarkar's projec-

tive algorithm [56,57], which is the fastest interior point algorithm known to the authors. Since $v = k \cdot n$, the time complexity is $O(k^{3.5}n^{3.5})$. The assignment step of the proposed balanced k-means algorithm can be solved in $O(n^3)$ time with the Hungarian algorithm, because the number of points and cluster slots ($k \cdot (n/k)$) is equal to n. This makes it much faster than in the constrained k-means, and therefore allows therefore significantly bigger datasets to be clustered.

Table 6.1: MSE*, and time/run of 100 runs of Balanced k-means and Constrained k-means.*

Dataset	n	k	Algorithm	Best	Mean	Time
s2	5000	15	Bal. k-means	**2.86**	(one run)	**1h 40min**
			Constr. k-means	—	—	**-**
s1	1000	15	Bal. k-means	2.89	(one run)	**47s**
subset			Constr. k-means	**2.61**	(one run)	26min
s1	500	15	Bal. k-means	3.48	3.73	**8s**
subset			Constr. k-means	**3.34**	**3.36**	30s
			k-means	2.54	4.21	0.01s
s1	500	7	Bal. k-means	14.2	15.7	10s
subset			Constr. k-means	**14.1**	**15.6**	**8s**
s2	500	15	Bal. k-means	3.60	3.77	**8s**
subset			Constr. k-means	**3.42**	**3.43**	29s
s3	500	15	Bal. k-means	3.60	3.69	**9s**
subset			Constr. k-means	**3.55**	**3.57**	35s
s4	500	15	Bal. k-means	3.46	3.61	**12s**
subset			Constr. k-means	**3.42**	**3.53**	45s
thyroid	215	2	Bal. k-means	**4.00**	**4.00**	2.5s
			Constr. k-means	**4.00**	**4.00**	**0.25s**
wine	178	3	Bal. k-means	**3.31**	3.33	0.36s
			Constr. k-means	**3.31**	**3.31**	**0.12s**
iris	150	3	Bal. k-means	**9.35**	9.39	0.34s
			Constr. k-means	**9.35**	**9.35**	**0.14s**

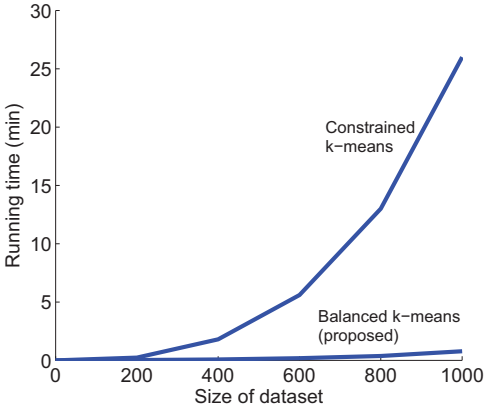


Figure 6.3: Running time with different-sized subsets of s1 dataset.

6.3 EXPERIMENTS

In the experiments we use artificial datasets s1–s4, which have Gaussian clusters with increasing overlap, and the real-world datasets thyroid, wine and iris. The source of the datasets is [29]. As a platform, Intel Core i5-3470 3.20GHz processor was used. We have been able to cluster datasets of 5000 points. A comparison of the *MSE* values of the constrained k-means with that of the balanced k-means is shown in Table 6.1, and the corresponding running times in Figure 6.3. The results indicate that constrained k-means gives slightly better MSE in many cases, but that balanced k-means is significantly faster when the size of the dataset increases. For a dataset with a size of 5000, constrained k-means could no longer provide the result within one day. The difference in the MSE is most likely due to the fact that balanced k-means strictly forces balance within ± 1 points, but constrained k-means does not. It may happen that constrained k-means has many clusters of size $\lfloor n/k \rfloor$, but some smaller amount of clusters of size bigger than $\lceil n/k \rceil$.

7 Clustering Based on Minimum Spanning Trees

Constructing a minimum spanning tree (MST) is needed in some clustering algorithms. We review here path-based clustering, for which constructing a minimum spanning tree quickly is beneficial. Path-based clustering is used when the shapes of the clusters are expected to be non-spherical, as manifolds.

7.1 CLUSTERING ALGORITHM

Path-based clustering employs the minimax distance to measure the dissimilarities of the data points [58, 59]. For a pair of data points X_i, X_j, the minimax distance D_{ij} is defined as:

$$D_{ij} = \min_{\mathcal{P}_{ij}^k} \{ \max_{(X_p, X_{p+1}) \in \mathcal{P}_{ij}^k} d(X_p, X_{p+1}) \} \tag{7.1}$$

where $\mathcal{P}_{ij}^k$ denotes all possible paths between X_i and X_j, k is an index that enumerates the paths, and $d(X_p, X_{p+1})$ is the Euclidean distance between two neighboring points X_p and X_{p+1}.

The minimax distance can be computed by an all-pair shortest path algorithm, such as the Floyd Warshall algorithm. However, this algorithm runs in time $O(n^3)$. An MST is used to compute the minimax distance more efficiently by Kim and Choi [60]. To make the path-based clustering robust to outliers, Chang and Yeung [61] improved the minimax distance and incorporated it into spectral clustering.

7.2 FAST APPROXIMATE MINIMUM SPANNING TREE

The paper **V** presents the fast minimum spanning tree (FMST) algorithm. It divides the dataset into clusters by k-means and calculates

the MSTs of the individual clusters by an exact algorithm. Then it combines these sub-MSTs. Figure 7.1 shows the phases of the construction.

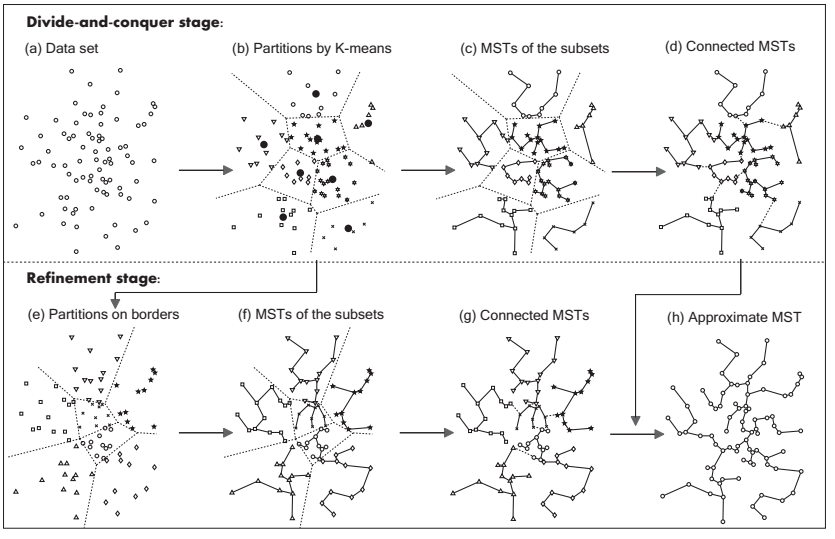


Figure 7.1: Phases of FMST algorithm.

7.3 ACCURACY AND TIME COMPLEXITY

The MST of a dataset can be constructed in $O(n^2)$ time with Prim's algorithm (we deal with complete graph). The exponent is too high for big datasets, so a faster variant of the algorithm is needed. We propose the FMST algorithm, which theoretically can achieve a time complexity of $O(n^{1.5})$. To get an estimate on its time complexity in practice, runs were made with different sizes of subsets of data and curves $T = aN^b$ were fitted to that data to find the exponent b. The running time in practice was found to be near $an^{1.5}$, see Table 7.1. The difference between theoretical and practical time complexity is due to the fact that the theoretical analysis makes the assumption that the cluster sizes are equal. This binds the publication **V** to balanced clustering.

Table 7.1: The exponent b obtained by fitting $T = aN^b$. FMST denotes the proposed method.

	b			
	t4.8k	MNIST	ConfLongDemo	MiniBooNE
n	8000	10000	164,860	130,065
d	2	748	3	50
FMST	1.57	1.62	1.54	1.44
Prim's Alg.	1.88	2.01	1.99	2.00

The resulting MST is not necessarily correct, but there may be some erroneous edges, the error rate being circa 2%–17% of the edges according to experiments.

The accuracy of the algorithm was tested on a clustering application. We tested the FMST within the path-based method on three synthetic datasets (Pathbased, Compound and S1) [29].

For computing the minimax distances, Prim's algorithm and FMST are used. In Fig. 7.2, one can see that the clustering results on the three datasets are almost equal. Quantitative measures are given in Table 7.2, which contains two validity indexes [3]. They indicate that the results of using Prim's algorithm on the first dataset are slightly better than the FMST, but the difference is insignificant.

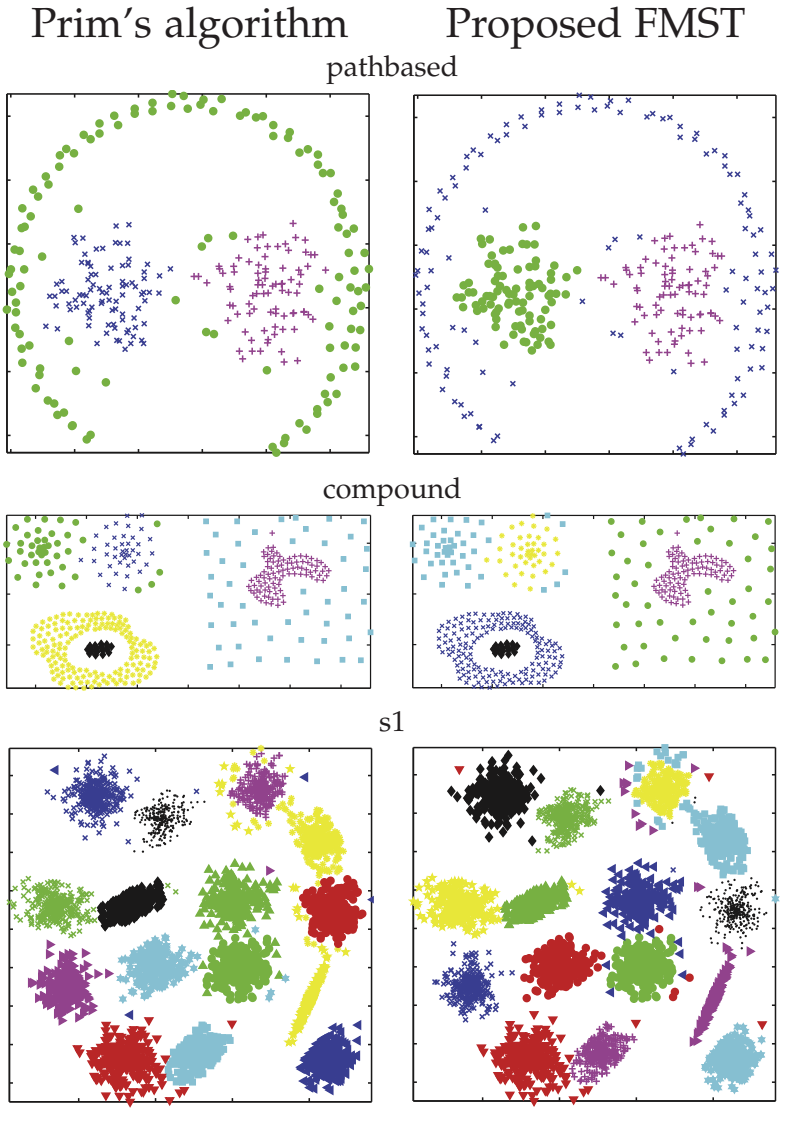


Figure 7.2: Prim's algorithm (left) and the proposed FMST based (right) clustering results for datasets pathbased (top), compound (middle) and s1 (bottom).

Table 7.2: The quantitative measures of clustering results (Rand and Adjusted Rand indices). FMST denotes the proposed method.

Datasets	Rand		AR	
	Prim	FMST	Prim	FMST
Pathbased	0.94	0.94	0.87	0.86
Compound	0.99	0.99	0.98	0.98
S1	0.995	0.995	0.96	0.96

8 Summary of Contributions

In this chapter we summarize the contributions of the original publications **I–V**. The publications **I–IV** introduce new clustering algorithms and the publication **V** introduces a heuristic minimum spanning tree calculation.

I: Data clustering is a combinatorial optimization problem. This publication shows that the clustering problem can be also considered as an optimization problem for an analytic function. The mean squared error can be written approximately as an analytic function. The gradient of this analytic function can be calculated and standard descent methods can be used to minimize this function. This analytic function formulation is a novel finding.

II: A model in clustering means the representatives of clusters. Traditionally, clustering works by fitting a model to the data. In this publication, we use the opposite starting point: we fit the data to an existing cluster model. We then gradually move the data points towards the original dataset, refining the centroid locations by *k*-means at every step. This is a novel approach and the quality of the clustering competes with the repeated *k*-means algorithm, where we set the number of repeats to be the same as the number of steps in our algorithm.

III: In this publication, we show that a clustering method where the total squared errors of the individual clusters are weighted by the number of points in the clusters, provides more balanced clustering than the unweighted *TSE* criterion. We also present a fast on-line algorithm for this problem. Balanced clustering is needed in some applications of workload balancing.

IV: This publication introduces a new balance-contrained clustering algorithm. In balance-constrained clustering, the sizes of the clusters are equal (+/- one point). The algorithm is based on *k*-means, but it differs in the assignment step, which is defined as a pairing problem and solved by the Hungarian algorithm. This

makes the algorithm significantly faster than constrained *k*-means, and allows datasets of over 5000 points to be clustered.

V: We apply a divide-and-conquer technique to the calculation of an approximate minimum spanning tree. We do the divide step with the *k*-means algorithm. The theoretical analysis is based on the assumption that the clusters are balanced after the divide step, which binds this publication to balanced clustering. A minimum spanning tree can be part of a clustering algorithm. This makes the quick computation of the minimum spanning tree desirable.

9 Summary of Results

We show the details of the datasets used throughout this thesis in Table 9.1. The main results for all the proposed algorithms are shown in Tables 9.2 and 9.3. The methods of MSE vs. balance plots are listed in Table 9.4 and the MSE vs. balance plots are in Figures 9.1 and 9.2. In these plots the datsets s1 150 and s4 150 are subsets of 150 points of datasets s1 and s4.

In Figures 9.1 and 9.2 we see that constrained k-means and balanced k-means have perfect balance (values 0), and Scut performs well with regard to both *MSE* and balance.

Table 9.1: Details of the used datasets.

dataset	type	n	k	d	used in publication				
					I	**II**	**III**	**IV**	**V**
s1	synthetic	5000	15	2	x	x	x		x
s2	synthetic	5000	15	2	x	x	x	x	
s3	synthetic	5000	15	2	x	x	x		
s4	synthetic	5000	15	2	x	x	x		
a1	synthetic	3000	20	2		x	x		
DIM32	high-dim.	1024	16	32		x	x		
DIM64	high-dim.	1024	16	64		x			
DIM128	high-dim.	1024	16	128		x			
DIM256	high-dim.	1024	16	256		x			
Bridge	image	4096	256	16		x			
Missa	image	6480	256	16		x			
House	image	34112	256	3		x			
Glass	real	214	7	9		x	x		
Wdbc	real	569	2	32		x	x		
Yeast	real	1484	10	8	x	x	x		
Wine	real	178	3	13	x	x	x	x	
Thyroid	real	215	2	5	x	x	x	x	
Iris	real	150	3	4	x	x	x	x	
Breast	real	699	2	9	x	x	x		
Pathbased	shape	300	3	2					x
Compound	shape	399	6	2					x

Table 9.2: Averages of MSE/d *values of 10–200 runs of methods. *) Best known values are among all the methods or by 2 hours run of random swap algorithm [18] or by GA [17].*

Mean square error												Dataset
Best known (*	$k-$ means++	Repeated $k-$means	*k*-means* paper **II**	*k*-means	Path-based paper **V**	Scut paper **III**	Fast Gl. *k*-means	Global *k*-means	Analytic, paper **I**	Constr. *k*-means	Bal. *k*-m. paper **IV**	
0.89	1.28	1.07	1.05	1.91	1.71	1.11	0.89	-	1.93	-	-	s1
1.33	1.55	1.35	1.47	2.03	5.26	1.53	1.33	1.33	2.04	-	1.43	s2
1.69	1.95	1.71	1.78	1.91	11.1	1.80	1.69	-	1.89	-	-	s3
1.57	1.70	1.57	1.59	1.68	13.3	1.61	1.57	-	1.73	-	-	s4
2.02	2.66	2.32	2.38	3.28	-	2.72	2.02	-	3.25	-	-	a1
4.91	7.18	159	7.10	424	7.28	7.09	7.10	-	4.91	-	7.09	DIM32
3.31	3.39	181	3.31	498	3.34	3.31	3.39	-	-	-	3.30	DIM64
2.10	2.17	276	2.10	615	2.15	2.10	2.17	-	-	-	2.10	DIM128
0.92	0.99	296	0.92	671	9.41	0.92	0.99	-	-	-	0.92	DIM256
161	177	166	165	168	8.13	-	164	-	-	-	-	Bridge
5.11	5.62	5.28	5.19	5.33	9.56	-	5.34	-	-	-	-	Missa
5.86	6.38	9.63	9.49	9.88	22.9	-	5.97	-	-	-	-	House
0.15	0.28	0.15	0.22	0.16	14.4	0.20	0.16	0.15	1.82	2.63	2.67	Glass
1.28	1.28	2.61	2.62	2.62	1.44	2.50	2.62	2.62	2.53	-	4.59	Wdbc
38	610	38	410	49	12.75	45	39	-	40	-	45	Yeast
0.88	0.89	1.89	1.93	2.43	2.86	2.04	0.88	1.89	1.04	2.55	2.56	Wine

Table 9.3: Processing time in seconds for different datasets and methods.

Dataset		s1	s2	s3	s4	a1	DIM32	DIM64	DIM128	DIM256	Bridge	Missa	House	Glass	Wdbc	Yeast	Wine
Processing time	Bal. *k*-m. paper **IV**	-	1h40min	-	-	-	18	17	18	24	-	-	-	1.86	19	6min	0.36
	Constr. *k*-means	-	-	-	-	-	-	-	-	-	-	-	-	1.74	-	-	0.12
	Analytic, paper **I**	4.73	6.97	4.59	5.43	4.01	47.9	-	-	-	-	-	-	0.67	0.27	5.15	0.44
	Global *k*-means	-	6min	-	-	-	-	-	-	-	-	-	-	1.24	0.51	-	0.33
	Fast Gl. *k*-means	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	Scut paper **III**	2.16	3.10	3.22	3.07	2.56	0.43	0.45	0.64	1.18	-	-	-	0.05	0.01	0.83	0.01
	Path-based paper **V**	51	46	50	82	-	2.48	2.33	2.69	3.10	180	999	24	0.15	1.36	13.0	0.08
	k-means	0.04	0.08	0.06	0.23	-	-	-	-	-	-	-	-	-	-	0.12	0.02
	k-means* paper **II**	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	Repeated *k*−means	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	k−means++	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	Best known (*	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table 9.4: Methods compared in Figures 9.1–9.2.

Method	Reference	Abbreviation
Analytic clustering	publication **I**	Analyt
k-means*	publication **II**	*k*-means*
Scut	publication **III**	Scut
Balanced *k*-means	publication **IV**	Bal km
Constrained *k*-means	[33,54], publication **IV**	Constr
k-means	[11]	*k*-means
Genetic algorithm	[62]	GA
Ncut	[39,63]	Ncut

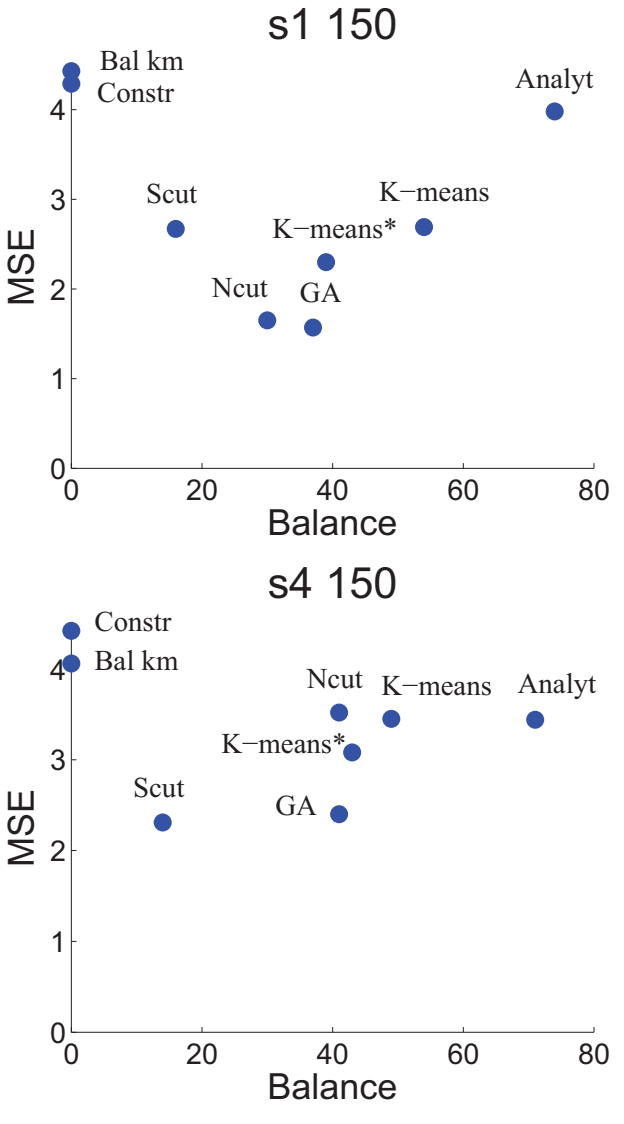


Figure 9.1: MSE *vs. balance for different methods. Means of 100 runs.*

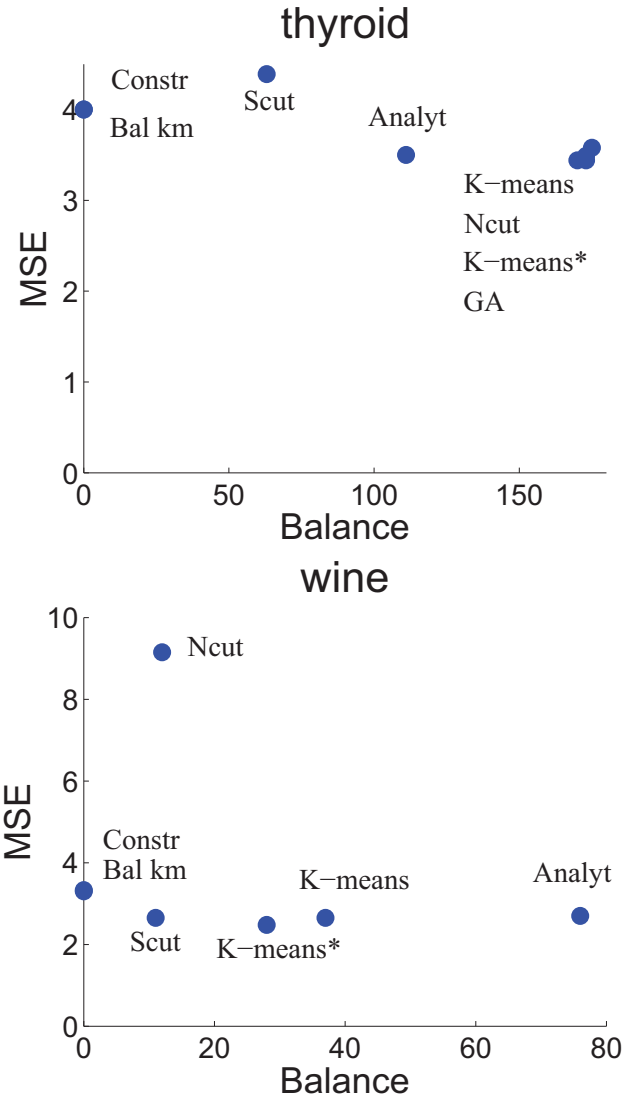


Figure 9.2: MSE *vs. balance for different methods. Means of 100 runs.*

10 Conclusions

In the publication **I** we have formulated the *TSE* as an analytic function and shown that the optimization of *TSE* can be made by gradient descent method. The results of the algorithm are comparable to *k*-means. As future work, the same technique could be used to produce clustering with infinity norm distance function.

In publication **II** we have introduced a completely new approach for optimizing *MSE*. The results are better than those of *k*-means++ and are comparable to repeated *k*-means.

In publication **III** we formulate l_2^2 *k*-clustering cost function using *TSE* and show that it leads to more balanced clusters than traditional clustering methods. The algorithm can be used when both good MSE and good balance are needed.

In publication **IV**, we introduce a balance-constrained clustering method, *balanced k-means*. The algorithm provides MSE optimization with the constraint that cluster sizes are balanced. The algorithm is fast compared to constrained *k*-means, and it provides clustering of datasets as big as over 5000 points. The algorithm can be used, for example, in workload balancing. As future work, a faster variant of balanced *k*-means could be produced. It should be fast enough to be used in the context of the publication **V** to achieve the theoretical result in practice.

In publication **V**, approximate MST is obtained theoretically in $O(n^{1.5})$ time compared to $O(n^2)$ of Prim's exact algorithm. The resulting MST was used in path-based clustering.

Overall, this thesis provides new alternatives to *k*-means clustering, either comparable to *k*-means, as in publications **I** and **II**, or having some special purpose, such as in publications **III** and **IV**.

References

[1] D. Aloise, A. Deshpande, P. Hansen, and P. Popat, "NP-hardness of Euclidean sum-of-squares clustering," *Mach. Learn.* **75,** 245–248 (2009).

[2] M. Inaba, N. Katoh, and H. Imai, "Applications of Weighted Voronoi Diagrams and Randomization to Variance-Based *k*-Clustering," in *Proceedings of the 10th Annual ACM symposium on computational geometry (SCG 1994)* (1994), pp. 332–339.

[3] L. Hubert and P. Arabie, "Comparing partitions," *Journal of Classification* **2,** 193–218 (1985).

[4] S. van Dongen, "Performance criteria for graph clustering and Markov cluster experiments," *Centrum voor Wiskunde en Informatica, INSR0012* (2000).

[5] N. Vinh, J. Epps, and J. Bailey, "Information theoretic measures for clusterings comparison: variants, properties, normalization and correction for change," *Journal of Machine Learning Research* **11,** 2837–2854 (2010).

[6] W. H. Equitz, "A New Vector Quantization Clustering Algorithm," *IEEE Trans. Acoust., Speech, Signal Processing* **37,** 1568–1575 (1989).

[7] D. Arthur and S. Vassilvitskii, "How Slow is the *k*-Means Method?," in *Proceedings of the 2006 Symposium on Computational Geometry (SoCG)* (2006), pp. 144–153.

[8] M. Zoubi and M. Rawi, "An efficient approach for computing silhouette coefficients," *Journal of Computer Science* **4,** 252–255 (2008).

[9] Q. Zhao, M. Xu, and P. Fränti, "Sum-of-Square Based Cluster Validity Index and Significance Analysis," in *Int. Conf. Adaptive*

and Natural Computing Algorithms (ICANNGA'09) (2009), pp. 313–322.

[10] J. Rissanen, *Optimal Estimation of Parameters* (Cambridge University Press, Cambridge, UK, 2012).

[11] J. MacQueen, "Some methods of classification and analysis of multivariate observations," in *Proc. 5th Berkeley Symp. Mathemat. Statist. Probability*, Vol. 1 (1967), pp. 281–296.

[12] D. Steinley and M. J. Brusco, "Initializing *k*-Means Batch Clustering: A Critical Evaluation of Several Techniques," *Journal of Classification* **24,** 99–121 (2007).

[13] J. M. Peña, J. A. Lozano, and P. Larrañaga, "An Empirical Comparison of Four Initialization Methods for the *k*-Means algorithm," *Pattern Recognition Letters* **20,** 1027–1040 (1999).

[14] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," in *SODA '07: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (2007), pp. 1027–1035.

[15] D. MacKay, Chap 20. An example inference task: Clustering in *Information Theory, Inference and Learning Algorithms* (Cambridge University Press, 2003).

[16] P. Fränti, O. Virmajoki, and V. Hautamäki, "Fast agglomerative clustering using a *k*-nearest neighbor graph," *IEEE Trans. on Pattern Analysis and Machine Intelligence* **28,** 1875–1881 (2006).

[17] P. Fränti and O. Virmajoki, "Iterative shrinking method for clustering problems," *Pattern Recognition* **39,** 761–765 (2006).

[18] P. Fränti and J. Kivijärvi, "Randomized local search algorithm for the clustering problem," *Pattern Anal. Appl.* **3,** 358–369 (2000).

[19] D. Pelleg and A. Moore, "X-means: Extending *k*-Means with Efficient Estimation of the Number of Clusters," in *Proceedings*

of the Seventeenth International Conference on Machine Learning (2000), pp. 727–734.

[20] A. Likas, N. Vlassis, and J. Verbeek, "The global *k*-means clustering algorithm," *Pattern Recognition* **36,** 451–461 (2003).

[21] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximun likelihood from incomplete data via the EM algorithm," *Journal of Royal Statistical Society B* **39,** 1–38 (1977).

[22] Q. Zhao, V. Hautamäki, I. Kärkkäinen, and P. Fränti, "Random swap EM algorithm for finite mixture models in image segmentation," in *16th IEEE International Conference on Image Processing (ICIP)* (2009), pp. 2397–2400.

[23] S.-G. Chen and P. Y. Hsieh, "Fast computation of the *N*th root," *Computers & Mathematics with Applications* **17,** 1423–1427 (1989).

[24] H. D. Vinod, "Integer programming and the theory of grouping," *Journal of Royal Statistical Association* **64,** 506–519 (1969).

[25] J. D. Carroll and A. Chaturvedi, Chap K-midranges clustering in *Advances in Data Science and Classification* (Springer, Berlin, 1998).

[26] H. Späth, *Cluster Dissection and Analysis: Theory, FORTRAN Programs, Examples* (Wiley, New York, 1985).

[27] L. Bobrowski and J. C. Bezdek, "c-Means Clustering with the l_1 and l_∞ norms," *IEEE Transactions on Systems, Man and Cybernetics* **21,** 545–554 (1991).

[28] D. Steinley, "*k*-Means Clustering: A Half-Century Synthesis," *British Journal of Mathematical and Statistical Psychology* **59,** 1–34 (2006).

[29] "Dataset page," Speech and Image Processing Unit, University of Eastern Finland, `http://cs.uef.fi/sipu/datasets/` (2015).

[30] S. Sahni and T. Gonzalez, "P-Complete Approximation Problems," *J. ACM* **23,** 555–565 (1976).

[31] W. F. de la Vega, M. Karpinski, C. Kenyon, and Y. Rabani, "Approximation schemes for clustering problems.," in *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing (STOC '03)* (2003), pp. 50–58.

[32] R. Nallusamy, K. Duraiswamy, R. Dhanalaksmi, and P. Parthiban, "Optimization of Non-Linear Multiple Traveling Salesman Problem Using *k*-Means Clustering, Shrink Wrap Algorithm and Meta-Heuristics," *International Journal of Nonlinear Science* **9,** 171 – 177 (2010).

[33] P. S. Bradley, K. P. Bennett, and A. Demiriz, "Constrained *k*-Means Clustering," *MSR-TR-2000-65, Microsoft Research* (2000).

[34] S. Zhu, D. Wang, and T. Li, "Data clustering with size constraints," *Knowledge-Based Systems* **23,** 883–889 (2010).

[35] A. Banerjee and J. Ghosh, "Frequency sensitive competitive learning for balanced clustering on high-dimensional hyperspheres," *IEEE Transactions on Neural Networks* **15,** 719 (2004).

[36] C. T. Althoff, A. Ulges, and A. Dengel, "Balanced Clustering for Content-based Image Browsing," in *GI-Informatiktage 2011* (2011).

[37] A. Banerjee and J. Ghosh, "On scaling up balanced clustering algorithms," in *Proceedings of the SIAM International Conference on Data Mining* (2002), pp. 333–349.

[38] L. Hagen and A. B. Kahng, "New Spectral Methods for Ratio Cut Partitioning and Clustering," *IEEE Transactions on Computer-Aided Design* **11,** 1074–1085 (1992).

[39] J. Shi and J. Malik, "Normalized Cuts and Image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **22,** 888–905 (2000).

[40] C. H. Q. Ding, X. He, H. Zha, M. Gu, and H. D. Simon, "A Min-max Cut Algorithm for Graph Partitioning and Data Clustering," in *Proceedings IEEE International Conference on Data Mining (ICDM)* (2001), pp. 107–114.

[41] Y. Chen, Y. Zhang, and X. Ji, "Size regularized cut for data clustering," in *Advances in Neural Information Processing Systems* (2005).

[42] Y. Kawahara, K. Nagano, and Y. Okamoto, "Submodular fractional programming for balanced clustering," *Pattern Recognition Letters* **32,** 235–243 (2011).

[43] Y. Liao, H. Qi, and W. Li, "Load-Balanced Clustering Algorithm With Distributed Self-Organization for Wireless Sensor Networks," *IEEE Sensors Journal* **13,** 1498–1506 (2013).

[44] L. Yao, X. Cui, and M. Wang, "An energy-balanced clustering routing algorithm for wireless sensor networks," in *Computer Science and Information Engineering, 2009 WRI World Congress on, IEEE*, Vol. 3 (2006), pp. 316–320.

[45] U. von Luxburg, "A Tutorial on Spectral Clustering," *Statistics and Computing* **17,** 395–416 (2007).

[46] T. D. Bie and N. Cristianini, "Fast SDP Relaxations of Graph Cut Clustering, Transduction, and Other Combinatorial Problems," *Journal of Machine Learning Research* **7,** 1409–1436 (2006).

[47] A. Frieze and M. Jerrum, "Improved Approximation Algorithms for MAX *k*-CUT and MAX BISECTION," *Algorithmica* **18,** 67–81 (1997).

[48] W. Zhu and C. Guo, "A Local Search Approximation Algorithm for Max-*k*-Cut of Graph and Hypergraph," in *Fourth International Symposium on Parallel Architectures, Algorithms and Programming* (2011), pp. 236–240.

[49] L. J. Schulman, "Clustering for edge-cost minimization," in *Proc. of the 32nd Ann. ACM Symp. on Theory of Computing (STOC)* (2000), pp. 547–555.

[50] N. Guttmann-Beck and R. Hassin, "Approximation Algorithms for Min-Sum P-Clustering," *Discrete Applied Mathematics* **89,** 125–142 (1998).

[51] H. Späth, *Cluster Analysis Algorithms for Data Reduction and Classification of Objects* (Wiley, New York, 1980).

[52] J. F. Sturm, O. Romanko, I. Polik, and T. Terlaky, "SeDuMi," (2009), `http://mloss.org/software/view/202/`.

[53] J. Löfberg, "YALMIP : A Toolbox for Modeling and Optimization in MATLAB," in *Proceedings of the CACSD Conference* (2004).

[54] A. Demiriz, K. P. Bennett, and P. S. Bradley, "Using Assignment Constraints To Avoid Empty Clusters in *k*-Means Clustering," in *Constrained Clustering: Advances in Algorithms, Theory, and Applications*, S. Basu, I. Davidson, and K. Wagstaff, eds. (Chapman & Hall/CRC, 2008).

[55] R. Burkhard, M. Dell'Amico, and S. Martello, *Assignment Problems (Revised reprint)* (SIAM, 2012).

[56] N. Karmarkar, "A New Polynomial Time Algorithm for Linear Programming," *Combinatorica* **4,** 373–395 (1984).

[57] G. Strang, "Karmarkars algorithm and its place in applied mathematics," *The Mathematical Intelligencer* **9,** 4–10 (1987).

[58] B. Fischer and J. M. Buhmann, "Path-based clustering for grouping of smooth curves and texture segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.* **25,** 513–518 (2003).

[59] B. Fischer and J. M. Buhmann, "Bagging for path-based clustering," *IEEE Trans. Pattern Anal. Mach. Intell.* **25,** 1411–1415 (2003).

[60] K. H. Kim and S. Choi, "Neighbor search with global geometry: A minimax message passing algorithm," in *24th International Conference on Machine Learning* (2007), pp. 401–408.

[61] H. Chang and D. Y. Yeung, "Robust path-based spectral clustering," *Pattern Recognition* **41,** 191–203 (2008).

[62] P. Fränti, J. Kivijärvi, T. Kaukoranta, and O. Nevalainen, "Genetic Algorithms for Large Scale Clustering Problem," *Comput. J.* **40,** 547 – 554 (1997).

[63] T. Cour, S. Yu, and J. Shi, "Ncut implementation," University of Pennsylvania, `http://www.cis.upenn.edu/~jshi/software/` (2004).

Paper **I**

M. I. Malinen and P. Fränti
"Clustering by analytic functions"
Information Sciences,
217, pp. 31–38, 2012.

Information Sciences 217 (2012) 31–38

Contents lists available at SciVerse ScienceDirect

Information Sciences

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

Clustering by analytic functions

Mikko I. Malinen *, Pasi Fränti

Speech and Image Processing Unit, School of Computing, University of Eastern Finland, P.O. Box 111, FIN-80101 Joensuu, Finland

ARTICLE INFO

Article history:
Received 27 January 2010
Received in revised form 12 April 2012
Accepted 10 June 2012
Available online 26 June 2012

Keywords:
Clustering
Analytic function
Mean squared error
k-Means

ABSTRACT

Data clustering is a combinatorial optimization problem. This article shows that clustering is also an optimization problem for an analytic function. The mean squared error, or in this case, the squared error can expressed as an analytic function. With an analytic function we benefit from the existence of standard optimization methods: the gradient of this function is calculated and the descent method is used to minimize the function.

1. Introduction

Euclidean sum-of-squares clustering is an NP-hard problem [2], where we group n data points into k clusters. Each cluster has a centre (centroid) which is the mean of the cluster and one tries to minimize the mean squared distance (mean squared error, MSE) of the data points from the nearest centroid. When the number of clusters k is constant, this problem becomes polynomial in time and can be solved in $O(n^{kd+1})$ time [14]. Although polynomial, this problem is slow to solve optimally. In practice, suboptimal algorithms are used. The method of k-means clustering [17] is fast and simple, although its worst-case running time is superpolynomial with a lower bound of $2^{\Omega(\sqrt{n})}$ for the number of iterations [3].

Given a set of observations $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$, where each observation is a d-dimensional real vector, then k-means clustering aims to partition the n observations into k sets $(k < n)$ $\mathbf{S} = \{S_1, S_2, \ldots, S_k\}$ so as to minimize the within-cluster sums of squares:

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in S_i} \|\mathbf{x}_j - \boldsymbol{\mu}_i\|^2, \tag{1}$$

where $\boldsymbol{\mu}_i$ is the mean of S_i. Given an initial set of k means $\mathbf{m}_1^{(1)}, \ldots, \mathbf{m}_k^{(1)}$, which may be specified randomly from the set of data points or by some heuristic [19,22,4], the k-means algorithm proceeds by alternating between two steps: [16].

Assignment step: Assign each observation to the cluster with the closest mean (i.e. partition the observations according to the Voronoi diagram generated by the means).

$$S_i^{(t)} = \left\{ \mathbf{x}_j : \left\| \mathbf{x}_j - \mathbf{m}_i^{(t)} \right\| \leqslant \|\mathbf{x}_j - \mathbf{m}_{i^*}^{(t)}\| \quad \forall \quad i^* = 1, \ldots, k \right\}. \tag{2}$$

Update step: Calculate the new means as the centroids of the observations in each cluster:

* Corresponding author.
E-mail addresses: mmali@cs.uef.fi (M.I. Malinen), franti@cs.uef.fi (P. Fränti).
URLs: http://cs.uef.fi/~mmali (M.I. Malinen), http://cs.uef.fi/~franti (P. Fränti).

0020-0255/$ - see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.ins.2012.06.018

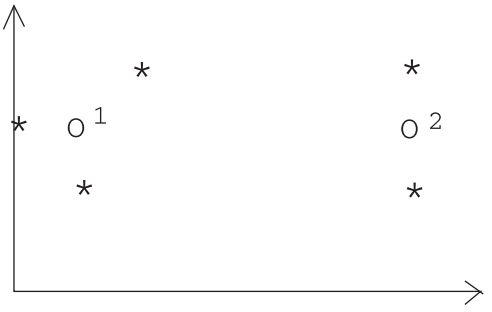


Fig. 1. A set of two clusters $i = 1, 2$ with five data points (0,3), (1,2), (2,4), (8,2), (8,4) in two dimensions (features) $j = 1, 2$. The feature j of data point k is represented as x_{kj}.

$$\mathbf{m}_i^{(t+1)} = \frac{1}{\left|S_i^{(t)}\right|} \sum_{\mathbf{x}_j \in S_i^{(t)}} \mathbf{x}_j. \tag{3}$$

The algorithm has converged when the assignments no longer change.

The advantage of *k*-means is that it finds a locally optimized solution for any given initial solution by repeating this simple two-step procedure. However, *k*-means cannot solve global problems in the clustering structure, and thus, it will work perfectly only if the global cluster structure is already optimized. By optimized global clustering structure we mean centroid locations from which optimal locations can be found by *k*-means. This is the main reason why slower agglomerative clustering is sometimes used [10,13,12], or other more complex *k*-means variants [11,18,4,15] are applied. Gaussian mixture models can be used (Expectation–Maximization algorithm) [8,25] and cut-based methods have been found to give competitive results [9]. To get a glimpse of the recent research in clustering, see [1,24,26], which deal with particle swarm optimization, ant-based clustering and minimum spanning tree based split-and-merge algorithm.

The method presented in this paper corresponds to *k*-means and is based on representing the squared error (SE) as an analytic function. The MSE or SE value can be calculated when the data points and centroid locations are known. The process involves finding the nearest centroid for each data point. An example dataset is shown in Fig. 1. We write c_{ij} for the centroid of cluster i, feature j. The squared error function can be written as

$$f(\bar{c}) = \sum_u \min_i \left\{ \sum_j (c_{ij} - x_{uj})^2 \right\}. \tag{4}$$

The min operation forces one to choose the nearest centroid for each data point. This function is not analytic because of the min operations. A question is whether we can express $f(\bar{c})$ as an analytic function which then could be given as input to a gradient-based optimization method. The answer is given in the following section.

2. Analytic clustering

2.1. Formulation of the method

We write the *p*-norm as

$$\|\bar{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}. \tag{5}$$

The maximum value of x_i's can be expressed as

$$\max(|x_i|) = \lim_{p\to\infty} \|\bar{x}\|_p = \lim_{p\to\infty} \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p} \tag{6}$$

Since we are interested in the *minimum* value, we take the inverses $\frac{1}{x_i}$ and find their maximum. Then another inverse is taken to obtain the minimum of the x_i:

$$\min(|x_i|) = \lim_{p\to\infty} \left(\sum_{i=1}^{n} \frac{1}{|x_i|^p} \right)^{-1/p} \tag{7}$$

2.2. Estimation of infinite power

Although calculations of the infinity norm without comparison operations are not possible, we can estimate the exact value by setting p to a high value. The estimation error is

$$\epsilon = \left(\sum_{i=1}^{n} \frac{1}{|x_i|^p}\right)^{-1/p} - \lim_{p_2 \to \infty} \left(\sum_{i=1}^{n} \frac{1}{|x_i|^{p_2}}\right)^{-1/p_2} \tag{8}$$

The estimation can be made up to any accuracy, the estimation error being

$$|\epsilon| > 0.$$

To see how close we can come in practice, a mathematical software package run was made:

$$1/\text{nthroot}((1/x1)\hat{}p + (1/x2)\hat{}p, p).$$

For example, with the values $x1,\ x2 = 500,\ p = 100$ we got the result 496.54. When the values of $x1$ and $x2$ are far from each other, we get an accurate estimate, but when the numbers are close to each other, an approximation error is present. In Table 1, the inaccuracy of the estimate is shown for different values of p and x_i. In this table, the estimate with two equal values $x_1 = x_2$ is calculated. In Fig. 2, the inaccuracy is calculated as a function of p. In this example, p cannot be increased much more, although it would give a more accurate answer. In Fig. 3, we see how large values of p can be used in maximum value calculations with this package. Moreover, in Fig. 4, we see how accurate the estimates can be using these maximum powers. On the basis of these results, we recommend scaling the values of x_i to the range $[0.5, 2]$ to achieve the best accuracy. Typically, dataset values are integers and range in magnitude from 0 to 500 or floats and range in magnitude from 0 to 1.

2.3. Analytic formulation of SE

Combining (4) and (7) yields

$$f(\bar{c}) = \sum_u \left[\lim_{p \to \infty} \left(\left(\sum_i \frac{1}{\left| \sum_j (c_{ij} - x_{uj})^2 \right|^p} \right)^{-1/p} \right) \right]. \tag{9}$$

Proceeding from (9) by removing lim, we can now write $\hat{f}(\bar{c})$ as an estimator for $f(\bar{c})$:

$$\hat{f}(\bar{c}) = \sum_u \left[\left(\sum_i \left(\sum_j (c_{ij} - x_{uj})^2 \right)^{-p} \right)^{-\frac{1}{p}} \right]. \tag{10}$$

This is an analytic estimator, although the exact $f(\bar{c})$ cannot be written as an analytic function when the data points lie in the middle of cluster centroids in a certain way.

Partial derivatives and the gradient can also be calculated. The formula for partial derivatives is calculated using the chain rule:

$$\frac{\partial \hat{f}(\bar{c})}{\partial c_{st}} = \sum_u \left[-\frac{1}{p} \cdot \left(\sum_i \left(\sum_j (c_{ij} - x_{uj})^2 \right)^{-p} \right)^{-\frac{p+1}{p}} \cdot \sum_i \left(-p \cdot \left(\sum_j (c_{ij} - x_{uj})^2 \right)^{-(p+1)} \right) \cdot 2 \cdot (c_{st} - x_{ut}) \right]. \tag{11}$$

Table 1
Inaccuracy of the estimate of the maximum value of ((6)) as p and x_i, $(i = 1,\ 2)$ change.

p	$x_i = 1$ (%)	$x_i = 10$ (%)	$x_i = 100$ (%)	$x_i = 500$ (%)
0	100	100	100	100
10	7	7	7	7
20	3	3	3	3
30	2	2	2	2
40	2	2	2	2
50	1.4	1.4	1.4	1.4
60	1.1	1.1	1.1	1.1
70	1.0	1.0	1.0	1.0
80	0.9	0.9	0.9	0.9
90	0.8	0.8	0.8	0.8
100	0.7	0.7	0.7	0.7
110	0.6	0.6	0.6	0.6
120	0.6	0.6	0.6	N/A

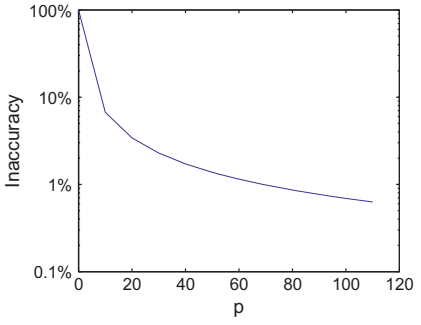


Fig. 2. Inaccuracy of estimate of the maximum value of ((6)) as a function of p ($x_i = 1$ to $x_i = 500,\ i = 1,\ 2$).

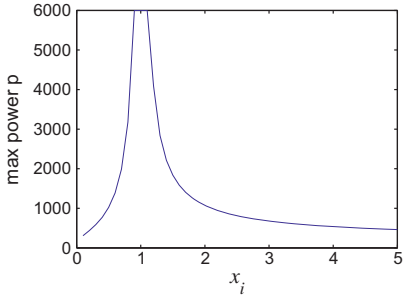


Fig. 3. Maximum power that can be calculated by a mathematical software package with different values of x_i.

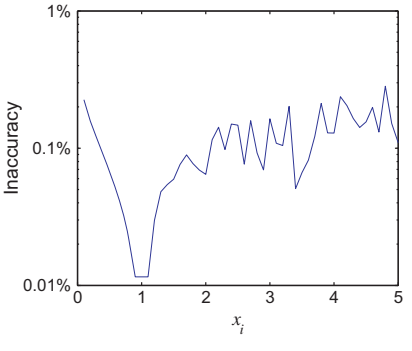


Fig. 4. Inaccuracy as a function of x_i, $i = 1,\ 2$, and when p is maximal.

2.4. Time complexity

For analysing the time complexity of calculating $\hat{f}(\bar{c})$, which is presented in ((10)), we know that $(\cdot)^{-p} = \frac{1}{(\cdot)^p}$ involves p divisions and that one division requires constant time in computer, and $(\cdot)^{\frac{1}{p}}$ takes $O(\log p)$ [7]. Using these, we can calculate

$$\begin{aligned} T(\hat{f}(\bar{c})) &= \left(d \cdot (\text{Mult} + \text{Add}) \cdot k \cdot (T(\wedge - p) + \text{Add}) + T\left(\wedge - \frac{1}{p}\right)\right) \cdot n = O(d \cdot \text{Mult} \cdot k \cdot T(\wedge - p) \cdot n) \\ &= O(d \cdot \text{Mult} \cdot k \cdot p \cdot n) = O(n \cdot d \cdot k \cdot p). \end{aligned} \tag{12}$$

The time complexity of calculating $\hat{f}(\bar{c})$ grows linearly with the number of data points n, dimensionality d, number of centroids k, and power p.

To calculate the time complexity of the partial derivative (s), which are presented in ((11)), we divide this into three parts, A, B, C:

$$
\begin{aligned}
A &= \left(\sum_i \left(\sum_j (c_{ij} - x_{uj})^2\right)^{-p}\right)^{-\frac{p+1}{p}} \\
B &= \sum_i \left(-p \cdot \left(\sum_j (c_{ij} - x_{uj})^2\right)^{-(p+1)}\right) \\
C &= (c_{st} - x_{ut}).
\end{aligned} \tag{13}
$$

Knowing that $(\cdot)^{-\frac{p+1}{p}} = (\cdot)^{-1} \cdot (\cdot)^{-\frac{1}{p}}$, we can write

$$
\begin{aligned}
T(A) &= d \cdot (\text{Mult} + \text{Add}) \cdot k \cdot (T(\wedge - p) + \text{Add}) + T\left(\wedge - \frac{1}{p}\right), \\
T(B) &= O(d \cdot (\text{Mult} + \text{Add}) \cdot T(\wedge - (p+1)) \cdot k), \\
T(C) &= \text{Subtr},
\end{aligned} \tag{14}
$$

and

$$
\begin{aligned}
T(\text{partial derivative}) &= O(T(A) + T(B) + T(C)) \cdot n) = O(T(B) \cdot n) = O(d \cdot \text{Mult} \cdot T(\wedge(p+1)) \cdot k \cdot n) = O(d \cdot p \cdot k \cdot n) \\
&= O(n \cdot d \cdot k \cdot p).
\end{aligned} \tag{15}
$$

To calculate all partial derivatives, we have to calculate part C for each partial derivative. The parts A and B are the same for all derivatives. Since we calculate part C n times, and there are $k \cdot d$ partial derivatives, we get

$$
T(\text{all partial derivatives}) = O(ndkp + n \cdot T(C) \cdot k \cdot d) = O(ndkp + n \cdot k \cdot d \cdot \text{Subtr}) = O(ndkp). \tag{16}
$$

This is linear in time for n, d, k and p, and differs only by the factor p from one iteration time complexity of the k-means $O(k \cdot n \cdot d)$.

2.5. Analytic optimization of SE

Since we can calculate the values of $\hat{f}(\bar{c})$ and the gradient, we can find a (local) minimum of $\hat{f}(\bar{c})$ by the gradient descent method. In the gradient descent method the points converge iteratively to a minimum:

$$
\bar{c}_{i+1} = \bar{c}_i - \nabla\hat{f}(\bar{c}_i) \cdot l, \tag{17}
$$

where l is the step length. The value of l can be calculated at every iteration, starting from some l_{max} and halving it recursively until $\hat{f}(\bar{c}_{i+1}) < \hat{f}(\bar{c}_i)$.

Eq. (11) for the partial derivatives depends on p. For any $p \geqslant 0$, either a local or the global minimum of (10) is found. Setting p large enough, we get a satisfactory estimator $\hat{f}(\bar{c})$, although there is always some bias in this estimator and a p that is too small may lead to a different clustering result.

There is also an alternative way to minimize $\hat{f}(\bar{c})$. Minimizing $\hat{f}(\bar{c})$ to the global minimum could be done by solving all $\bar{c}$ from (18) and trying them, one at a time, in $\hat{f}(\bar{c})$, because at a minimum point (global or local) all components of the gradient must be zero:

$$
\sum_{i,j}\left(\frac{\partial\hat{f}(\bar{c})}{\partial c_{ij}}\right)^2 = 0. \tag{18}
$$

This alternative way has only theoretical significance, since it is not known how to find all solutions of (18). There are at least $i_{max}!$ solutions to this equation, since from each solution (which surely exist), $i_{max}!$ solutions can be obtained by permuting the centroids.

The analytic clustering method presented here corresponds to the k-means algorithm [17]. It can be used to obtain a local minimum of the squared error function similarly to k-means, or to simulate the random swap algorithm [11] by changing one cluster centroid randomly. In the random swap algorithm, a centroid and a datapoint are chosen randomly, and a trial movement of this centroid to this datapoint is made. If the k-means with the new centroid provide better results than the earlier solution, the centroid remains swapped. Such trial swaps are then repeated for a fixed number of times.

Analytic clustering and k-means work in the same way, although their implementations differ. Their step length is different. The difference in the clustering result also originates from the approximation of the ∞-norm by the p-norm.

We have used an approximation to the infinity norm to find the nearest centroids for the datapoints, and used the sum-of-squares for the distance metric. The infinity norm, on the other hand, could be used to cluster with the infinity norm distance metric. Most partitioning clustering papers use the $p = 2$ (Euclidean norm) as the distance metric as we do, but some papers have experimented with different norms. For example, $p = 1$ gives the k-medians clustering, e.g. [23], and $p \to 0$ gives the categorical k-modes clustering. Papers on the k-midrange clustering (e.g. [6,20]) employ the infinity norm ($p = \infty$) in finding the range of a cluster. In [5] a $p = \infty$ formulation has been given for the more general fuzzy case. A description and comparison of different formulations has been given in [21]. With the infinity norm distance metric, the distance of a

data point from a centroid is the dominant feature of the difference vector between the data point and the centroid. Our contribution in this regard is that we can form an analytic estimator for the cost function even if the distance metric were the infinity norm. This would make the formula for $\hat{f}(\bar{c})$ and the formula for the partial derivatives a little bit more complicated but nevertheless possible as a future direction, and thus, it is omitted here.

3. Experiments

We test this new clustering method not by using the p-norm but using the min-function to calculate the distances to the nearest centroids and a line search instead of the gradient descent method. We use several small and mid-size datasets (see Fig. 5) and compare the results of the analytic clustering, the k-means clustering, the random swap clustering, and the analytic random swap clustering. The number of clusters is based on the known number of clusters in the datasets. The results are illustrated in Table 2 and show that analytic clustering and k-means clustering provide comparable results. In these experiments, the analytic random swap algorithm sometimes gives a better (lower) SE value than random swapping. We also calculated the Adjusted Rand index, a neutral measure of clustering performance beyond sum of squares, for ten runs of the analytic clustering and the k-means clustering as well as for the random swap variants of these. Runs are done for the s-sets. The means of the Rand indices are shown in Table 3. These results indicate that the clustering performance is very similar between the analytic and the traditional methods. The running time for the s-sets is reasonable (e.g., 4.6 s for analytic clus-

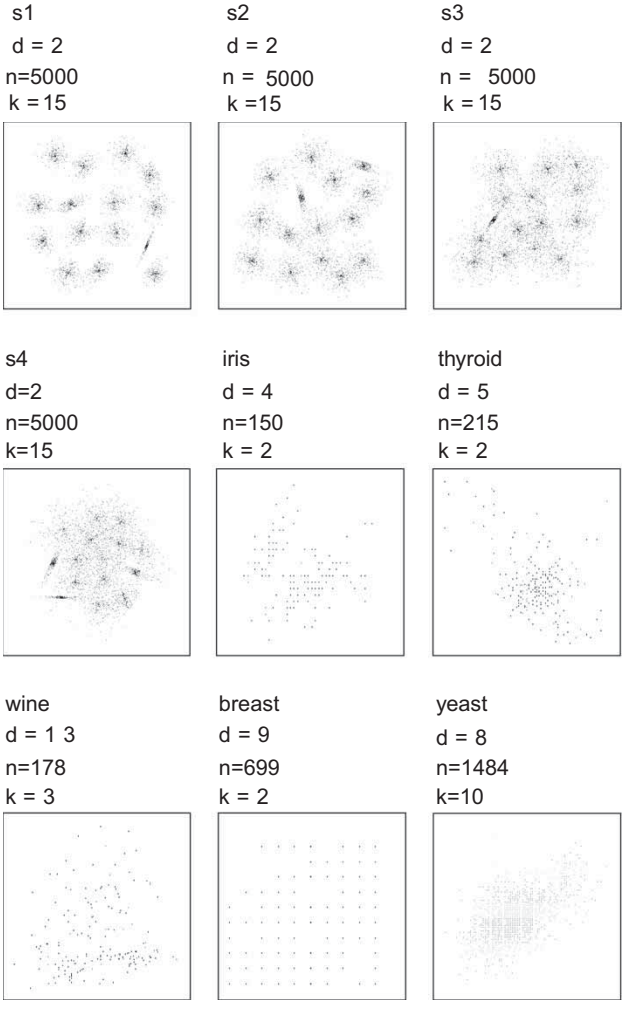


Fig. 5. Datasets $s1$, $s2$, $s3$, $s4$, iris, thyroid, wine, breast and yeast used in experiments. Two first dimensions are shown.

Table 2
Averages of SE values of 30 runs of analytic and traditional methods. The SE values are divided by 10^{13} or 10^6 (wine set) or 10^4 (breast set) or 1 (yeast set). Calculated using ((4)). Processing times in seconds for different datasets and methods.

Dataset	Squared error				Processing time			
	k-Means		Random swap		*k*-Means		Random swap	
	Anal.	Trad.	Anal.	Trad.	Anal.	Trad.	Anal.	Trad.
*s*1	1.93	1.91	1.37	1.39	4.73	0.04	52.46	0.36
*s*2	2.04	2.03	1.52	1.62	6.97	0.08	51.55	0.61
*s*3	1.89	1.91	1.76	1.78	4.59	0.06	59.03	0.58
*s*4	1.70	1.68	1.58	1.60	5.43	0.23	49.12	1.13
Iris	22.22	22.22	22.22	22.22	0.12	0.01	0.48	0.03
Thyroid	74.86	74.80	73.91	73.91	0.22	0.02	0.72	0.04
Wine	2.41	2.43	2.37	2.37	0.44	0.02	4.39	0.04
Breast	1.97	1.97	1.97	1.97	0.15	0.02	1.07	0.04
Yeast	48.87	48.79	45.83	46.06	5.15	0.12	50.00	0.91

Table 3
Adjusted Rand indices for analytic clustering and *k*-means clustering and for the random swap variants of these.

Dataset	Analytic	*k*-Means	Analytic random swap	Random swap
*s*1	0.86	0.87	0.95	0.95
*s*2	0.89	0.89	0.95	0.95
*s*3	0.85	0.87	0.95	0.95
*s*4	0.84	0.86	0.93	0.92

tering vs. 0.1 s for *k*-means). The proposed method can theoretically be applied to large datasets as well, or datasets with a large number of dimensions or clusters. The time complexity is linear with respect to all of these factors. However, in our implementation, we use line search to optimize and use min-function to calculate the nearest centroids, and we have experienced that time consuming increases heavily when these factors increase, and larger datasets are too heavy for this. See the running time comparisons in Table 2. The software used to compute the values in Table 2 is available at http://cs.uef.fi/sipu/soft.

Experiments with the *s*-sets show that the proposed approach leads to similar membership results for the individual data points. Out of the 15 centroids, typically 12–13 are approximately at the same locations and the other two or three at different locations.

4. Conclusions

We proposed a way to form an analytic squared error function. From this function, the partial derivatives can be calculated, and then a gradient descent method can be used to find a local minimum of the squared error. Analytic clustering and *k*-means clustering provide approximately the same result, whereas analytic random swap clustering sometimes gives a better result than random swapping. In *k*-means, there are two phases in one iteration, but in analytic clustering these two phases are combined into a single phase. As a future work, we could consider an implementation including also the gradient calculation and the use of the gradient descent method. Also, then, it would be natural to set a suitable value for the power p, for which now only an extreme theoretical upper limit can be calculated.

References

[1] A. Ahmadi, F. Karray, M.S. Kamel, Model order selection for multiple cooperative swarms clustering using stability analysis, Inform. Sci. 182 (2012) 169–183.
[2] D. Aloise, A. Deshpande, P. Hansen, P. Popat, NP-hardness of Euclidean sum-of-squares clustering, Mach. Learn. 75 (2009) 245–248.
[3] D. Arthur, S. Vassilvitskii, How slow is the *k*-means method?, in: Proceedings of the 2006 Symposium on Computational Geometry (SoCG), pp. 144–153.
[4] D. Arthur, S. Vassilvitskii, k-Means++: the advantages of careful seeding, in: SODA '07: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2007, pp. 1027–1035.
[5] L. Bobrowski, J.C. Bezdek, *c*-Means clustering with the l_1 and l_∞ norms, IEEE Transactions on Systems, Man and Cybernetics 21 (1991) 545–554.
[6] J.D. Carroll, A. Chaturvedi, *k*-Midranges clustering, in: A. Rizzi, M. Vichi, H.H. Bock (Eds.), Advances in Data Science and Classification, Springer, Berlin, 1998.
[7] S.G. Chen, P.Y. Hsieh, Fast computation of the *N*th root, Computers & Mathematics with Applications 17 (1989) 1423–1427.
[8] A.P. Dempster, N.M. Laird, D.B. Rubin, Maximun likelihood from incomplete data via the EM algorithm, Journal of Royal Statistical Society B 39 (1977) 1–38.
[9] C.H.Q. Ding, X. He, H. Zha, M. Gu, H.D. Simon, A min–max cut algorithm for graph partitioning and data clustering, in: Proceedings IEEE International Conference on Data Mining 2001 (ICDM), pp. 107–114.
[10] W.H. Equitz, A new vector quantization clustering algorithm, IEEE Trans. Acoust., Speech, Signal Proces. 37 (1989) 1568–1575.
[11] P. Fränti, J. Kivijärvi, Randomized local search algorithm for the clustering problem, Pattern Anal. Appl. 3 (2000) 358–369.
[12] P. Fränti, O. Virmajoki, Iterative shrinking method for clustering problems, Pattern Recogn. 39 (2006) 761–765.

[13] P. Fränti, O. Virmajoki, V. Hautamäki, Fast agglomerative clustering using a *k*-nearest neighbor graph, IEEE Trans. Pattern Anal. Mach. Intell. 28 (2006) 1875–1881.
[14] M. Inaba, N. Katoh, H. Imai, Applications of weighted voronoi diagrams and randomization to variance-based *k*-clustering, in: Proceedings of the 10th Annual ACM symposium on computational geometry (SCG 1994), 1994, pp. 332–339.
[15] A. Likas, N. Vlassis, J. Verbeek, The global *k*-means clustering algorithm, Pattern Recogn. 36 (2003) 451–461.
[16] D. MacKay, An example inference task: clustering, in: Information Theory, Inference and Learning Algorithms, Cambridge University Press, 2003, pp. 284–292 (Chapter 20).
[17] J. MacQueen, Some methods of classification and analysis of multivariate observations, in: Proc. 5th Berkeley Symp. Mathemat. Statist. Probability, vol. 1, 1967, pp. 281–296.
[18] D. Pelleg, A. Moore, *X*-Means: Extending *k*-means with efficient estimation of the number of clusters, in: Proceedings of the Seventeenth International Conference on Machine Learning, Morgan Kaufmann, San Francisco, 2000, pp. 727–734.
[19] J.M. Peña, J.A. Lozano, P. Larrañaga, An empirical comparison of four initialization methods for the *k*-means algorithm, Pattern Recogn. Lett. 20 (1999) 1027–1040.
[20] H. Späth, Cluster Dissection and Analysis: Theory FORTRAN Programs, Examples, Wiley, New York, 1985.
[21] D. Steinley, *k*-Means clustering: a half-century synthesis, Brit. J. Math. Stat. Psychol. 59 (2006) 1–34.
[22] D. Steinley, M.J. Brusco, Initializing *k*-means batch clustering: a critical evaluation of several techniques, J. Classif. 24 (2007) 99–121.
[23] H.D. Vinod, Integer programming and the theory of grouping, J. Roy. Stat. Assoc. 64 (1969) 506–519.
[24] L. Zhang, Q. Cao, A novel ant-based clustering algorithm using the kernel method, Inform. Sci. 181 (2011) 4658–4672.
[25] Q. Zhao, V. Hautamäki, I. Kärkkäinen, P. Fränti, Random swap EM algorithm for finite mixture models in image segmentation, in: 16th IEEE International Conference on Image Processing (ICIP), pp. 2397–2400.
[26] C. Zhong, D. Miao, P. Fränti, Minimum spanning tree based split-and-merge: a hierarchical clustering method, Inform. Sci. 181 (2011) 3397–3410.

Paper **II**

M. I. Malinen, R. Mariescu-Istodor
and P. Fränti
"K-means*: Clustering by gradual
data transformation"
Pattern Recognition,
47 (10), pp. 3376–3386, 2014.

Pattern Recognition 47 (2014) 3376–3386

Contents lists available at ScienceDirect

Pattern Recognition

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

K-means*: Clustering by gradual data transformation

Mikko I. Malinen*, Radu Mariescu-Istodor, Pasi Fränti

Speech and Image Processing Unit, School of Computing, University of Eastern Finland, Box 111, FIN-80101 Joensuu, Finland

ARTICLE INFO

Article history:
Received 30 September 2013
Received in revised form
27 March 2014
Accepted 29 March 2014
Available online 18 April 2014

Keywords:
Clustering
K-means
Data transformation

ABSTRACT

Traditional approach to clustering is to fit a model (partition or prototypes) for the given data. We propose a completely opposite approach by fitting the data into a given clustering model that is optimal for similar pathological data of equal size and dimensions. We then perform inverse transform from this pathological data back to the original data while refining the optimal clustering structure during the process. The key idea is that we do not need to find optimal global allocation of the prototypes. Instead, we only need to perform local fine-tuning of the clustering prototypes during the transformation in order to preserve the already optimal clustering structure.

1. Introduction

Euclidean sum-of-squares clustering is an NP-hard problem [1], where one assigns n data points to k clusters. The aim is to minimize the mean squared error (MSE), which is the mean distance of the data points from the nearest centroids. When the number of clusters k is constant, Euclidean sum-of-squares clustering can be done in polynomial $O(n^{kd+1})$ time [2], where d is the number of dimensions. This is slow in practice, since the power $kd+1$ is high, and thus, suboptimal algorithms are used. The *K*-means algorithm [3] is fast and simple, although its worst-case running time is high, since the upper bound for the number of iterations is $O(n^{kd})$ [4].

In k-means, given a set of data points $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$, one tries to assign the data points into k sets ($k < n$), $\mathbf{S} = \{S_1, S_2, \ldots, S_k\}$, so that MSE is minimized:

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in S_i} \|\mathbf{x}_j - \boldsymbol{\mu}_i\|^2$$

where $\boldsymbol{\mu}_i$ is the mean of S_i. An initial set of the k means $\mathbf{m}_1^{(1)}, \ldots, \mathbf{m}_k^{(1)}$ may be given randomly or by some heuristic [5–7]. The k-means algorithm alternates between the two steps [8]:

Assignment step: Assign the data points to clusters specified by the nearest centroid:

$$S_i^{(t)} = \left\{ \mathbf{x}_j : \|\mathbf{x}_j - \mathbf{m}_i^{(t)}\| \le \|\mathbf{x}_j - \mathbf{m}_{i^*}^{(t)}\|, \ \forall\ i^* = 1, \ldots, k \right\}$$

Update step: Calculate the mean of each cluster:

$$\mathbf{m}_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{\mathbf{x}_j \in S_i^{(t)}} \mathbf{x}_j$$

The k-means algorithm converges when the assignments no longer change. In practice, the k-means algorithm stops when the criterion of inertia does not vary significantly: it is useful to avoid non-convergence when the clusters are symmetrical, and in the other cluster configurations, to avoid too long time of convergence.

The main advantage of k-means is that it always finds a local optimum for any given initial centroid locations. The main drawback of k-means is that it cannot solve global problems in the clustering structure (see Fig. 1). By solved global clustering structure we mean such initial centroid locations from which the optimum can be reached by k-means. This is why slower agglomerative clustering [9–11], or more complex k-means variants [7,12–14] are sometimes used. *K*-means++ [7] is like k-means, but there is a more complex initialization of centroids. Gaussian mixture models can also be used (Expectation-Maximization algorithm) [15,16] and cut-based methods have been found to give competitive results [17]. To get a view of the recent research in clustering, see [18–20], which deal with analytic clustering, particle swarm optimization and minimum spanning tree based split-and-merge algorithm.

In this paper, we attack the clustering problem by a completely different approach than the traditional methods. Instead of trying to solve the correct global allocation of the clusters by fitting the clustering model to the data X, we do the opposite and fit the data to an optimal clustering structure. We first generate an artificial data X^* of the same size (n) and dimension (d) as the input data, so that the data vectors are divided into k perfectly separated clusters

* Corresponding author.
E-mail addresses: mmali@cs.uef.fi (M.I. Malinen), radum@cs.uef.fi (R. Mariescu-Istodor), franti@cs.uef.fi (P. Fränti).
URLS: http://cs.uef.fi/~mmali (M.I. Malinen), http://cs.uef.fi/~radum (R. Mariescu-Istodor), http://cs.uef.fi/pages/franti (P. Fränti).

http://dx.doi.org/10.1016/j.patcog.2014.03.034
0031-3203/

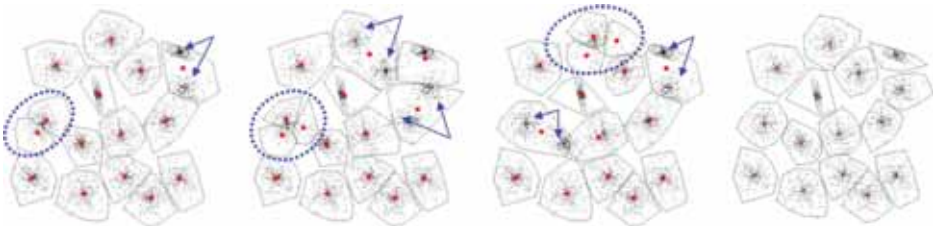


Fig. 1. Results of k-means for three random initializations (left) showing that k-means cannot solve global problems in the clustering structure. Circles show clusters that have too many centroids. Arrows show clusters that have too few centroids. Clustering result obtained by the proposed method (right).

without any variation. We then perform one-to-one bijective mapping of the input data to the artificial data ($X \rightarrow X^*$).

The key point is that we already have a clustering that is optimal for the artificial data, but not for the real data. In the next step, we then perform inverse transform of the artificial data back to the original data by a sequence of gradual changes. While doing this, the clustering model is updated after each change by k-means. If the changes are small, the data vectors will gradually move to their original position without breaking the clustering structure. The details of the algorithm including the pseudocode are given in Section 2. An online animator demonstrating the progress of the algorithm is available at http://cs.uef.fi/sipu/clustering/animator/. The animation starts when "Gradual k-means" is chosen from the menu.

The main design problems of this approach are to find a suitable artificial data structure, how to perform the mapping, and how to control the inverse transformation. We will demonstrate next that the proposed approach works with simple design choices, and overcomes the locality problem of k-means. It cannot be proven to provide optimal result every time, as there are pathological counter-examples where it fails to find the optimal solution. Nevertheless, we show by experiments that the method is significantly better than k-means, significantly better than k-means++ and competes equally with repeated k-means. It also rarely ends up to a bad solution that is typical to k-means. Experiments will show that only a few transformation steps are needed to obtain a good quality clustering.

2. *K*-means* algorithm

In the following subsections, we will go through the phases of the algorithm. For pseudocode, see Algorithm 1. We call this algorithm k-means*, because of the repeated use of k-means. However, instead of applying k-means to the original data points, we create another artificial dataset which is prearranged into k clearly separated zero-variance clusters.

2.1. Data initialization

The algorithm starts by choosing the artificial clustering structure and then dividing the data points among these equally. We do this by creating a new dataset X_2 and by assigning each data point in the original dataset X_1 to a corresponding data point in X_2, see Fig. 2. We consider seven different structures for the initialization:

- line
- diagonal
- random
- random with optimal partition
- initialization used in k-means++
- line with uneven clusters
- point.

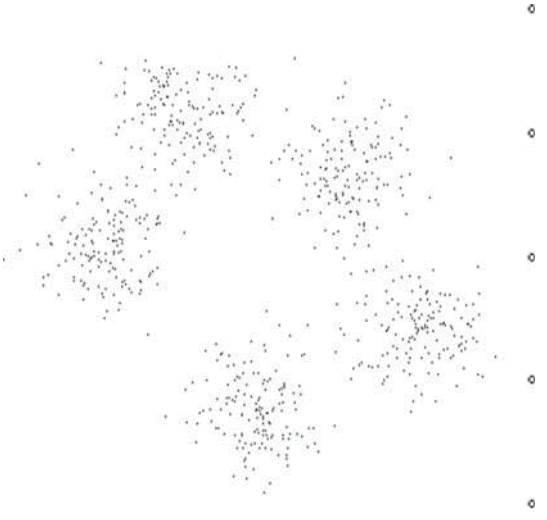


Fig. 2. Original dataset (left), and the corresponding artificial dataset using line init (right).

In the line structure, the clusters are arranged along a line. The k locations are set as the middle value of the range in each dimension, except the last dimension where the k clusters are distributed uniformly along the line, see Fig. 3 (left) and the animator http://cs.uef.fi/sipu/clustering/animator/. The range of 10% nearest to the borders is left without clusters. In the diagonal structure, the k locations are set uniformly to the diagonal of the range of the dataset. In the random structure, the initial clusters are selected randomly among the data point locations in the original dataset, see Fig. 3 (right). In these structuring strategies, data point locations are initialized randomly to these cluster locations. Even distribution among the clusters is a natural choice. To justify it further, lower cardinality clusters can more easily become empty later, which is an undesirable situation.

The fourth structure is random locations but using optimal partitions for the mapping. This means assigning the data points to the nearest clusters. The fifth structure corresponds to the initialization strategy used in k-means++ [7]. This initialization is done as follows: at any given time, let $D(X_i)$ denote the shortest distance from a data point X_i to its closest centroid we have already chosen.

Choose first centroid C_1 uniformly at random from X.

Repeat: Choose the next centroid as a point X_i, using a weighted probability distribution where a point is chosen with probability proportional to $D(X_i)^2$.

Until we have chosen a total of k centroids.

As a result, new centers are added more likely to the areas lacking centroids. The sixth structure is the line with uneven clusters, in which

we place twice more points to most centrally located half of the cluster locations. The seventh structure is the point. It is like line structure but we put the clusters in a very short line, which looks like a single point in larger scale. In this way the dataset "explodes" from a single point during the inverse transform. This structure is useful mainly for the visualization purpose in the web-animator. The *k*-means++-style structure with evenly distributed data points is the recommended structure because it works best in practice, and therefore we use it in further experiments. In choosing the structure, good results are achieved when there is a notable separation between clusters and evenly distributed data points in clusters.

Once the initial structure is chosen, each data point in the original dataset is assigned to a corresponding data point in the initial structure. The data points in this manually-created dataset are randomly but evenly located in this initial structure.

2.2. Inverse transformation steps

The algorithm proceeds by executing a given number of *steps*, which is a user-set integer parameter (steps > 1). Default value for steps is 20. At each step, all data points are transformed towards their original location by amount

$$\frac{1}{steps}\cdot(X_{1,i}-X_{2,i}), \tag{1}$$

where $X_{1,i}$ is the location of the i:th datapoint in the original data and $X_{2,i}$ is its location in the artificial structure. After every transform, *k*-means is executed given the previous codebook along with the modified dataset as input. After all the steps have been completed, the resulting codebook C is output.

It is possible, that two points that belong to the same cluster in the original dataset will be put to different clusters in the manually-created dataset. Then they smoothly move to final locations during the inverse transform.

Algorithm 1. *K*-means*.

Input: dataset X_1, number of clusters k, *steps*,
Output: Codebook C.

$n \leftarrow size(X_1)$
$[X_2, C] \leftarrow Initialize()$
for repeats=1 to steps **do**
  for i=1 to n **do**
    $X_{3,i} \leftarrow X_{2,i} + (repeats/steps)*(X_{1,i} - X_{2,i})$
  end for
  $C \leftarrow kmeans(X_3, k, C)$
end for
output C

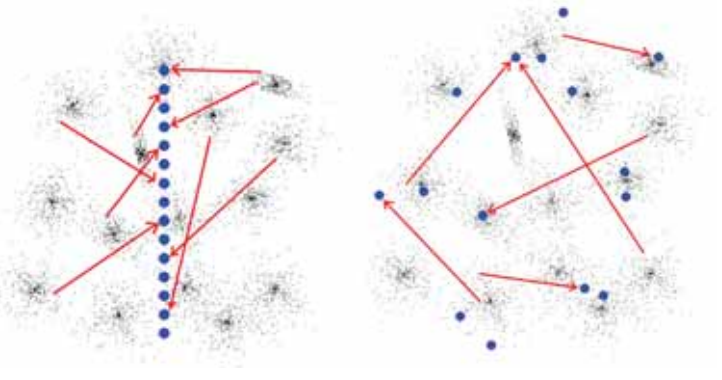


Fig. 3. Original dataset and line init (left) or random init (right) with sample mappings shown by arrows.

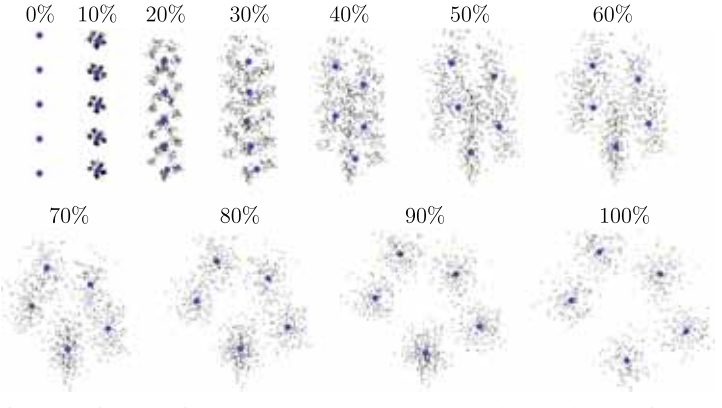


Fig. 4. Progress of the algorithm for a subset of 5 clusters of dataset a3. Data spreads towards the original dataset, and centroids follow in optimal locations. The subfigures correspond to phases 0%, 10%, 20%,…,100% completed.

2.3. Optimality considerations

The basic idea is that if the codebook was all the time optimal for all intermediate datasets, the generated final clustering would also be optimal for the original data. In fact, many times this optimality is reached; see Fig. 4 for an example how the algorithm proceeds. However, the optimality cannot be always guaranteed.

There are a couple of counter-examples, which may happen during the execution of the algorithm. The first is non-optimality of global allocation, which in some form is present in all practical clustering algorithms. Consider the setting in Fig. 5. The data points $x_{1..6}$ are traversing away from their centroid C_1. Two centroids would be needed there, one for the data points $x_{1..3}$ and another one for the data points $x_{4..6}$. On the other hand, the data points $x_{13..15}$ and $x_{16..18}$ are approaching each other and only one of the centroids C_3 or C_4 would be needed. This counter-example shows that this algorithm cannot guarantee optimal result, in general.

2.4. Empty cluster generation

Another undesired situation that may happen during the clustering is generation of an empty cluster, see Fig. 6. Here the data points $x_{1..6}$ are traversing away from their centroid C_2 and eventually leave the cluster empty. This is undesirable, because one cannot execute k-means with an empty cluster. However, this problem is easy to detect and can be fixed in most cases by a random swap strategy [12]. Here the problematic centroid is swapped to a new location randomly chosen from the data points. We move the centroids of empty clusters in the same manner.

2.5. Time complexity

The worst case complexities of the phases are listed in Table 1. The overall time complexity is not more than for the k-means, see Table 1. The proposed algorithm is asymptotically faster than global k-means and even faster than the fast variant of global k-means, see Table 2.

The derivation of the complexities in Table 1 is straightforward, and we therefore discuss here only the empty cluster detection and removal phases. There are n data points, which will be assigned to k centroids. To detect empty clusters we have to go

Table 1
Time complexity of the proposed algorithm.

Algorithm	k free	$k=O(n)$	$k=O(\sqrt{n})$	$k=O(1)$
Theoretical				
Initialization	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Dataset transform	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Empty clusters removal	$O(kn)$	$O(n^2)$	$O(n^{1.5})$	$O(n)$
k-means	$O(kn^{kd+1})$	$O(n^{O(n)\cdot d+2})$	$O(n^{O(\sqrt{n}d+3/2)})$	$O(n^{kd+1})$
Algorithm total	$O(kn^{kd+1})$	$O(n^{O(n)\cdot d+2})$	$O(n^{O(\sqrt{n}d+3/2)})$	$O(n^{kd+1})$
Fixed k-means				
Initialization	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Dataset transform	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Empty clusters removal	$O(kn)$	$O(n^2)$	$O(n^{1.5})$	$O(n)$
k-means	$O(kn)$	$O(n^2)$	$O(n^{1.5})$	$O(n)$
Algorithm total	$O(kn)$	$O(n^2)$	$O(n^{1.5})$	$O(n)$

Table 2
Time complexity comparison for k-means* and global k-means.

Algorithm	Time complexity for fixed k-means
Global k-means	$O(n\cdot k\cdot$ complexity of $k-$means$)=O(k^2\cdot n^2)$
Fast global k-means	$O(k\cdot$ complexity of $k-$means$)=O(k^2\cdot n)$
K-means*	$O($steps $\cdot$ complexity of $k-$means$)=O($steps $\cdot k\cdot n)$

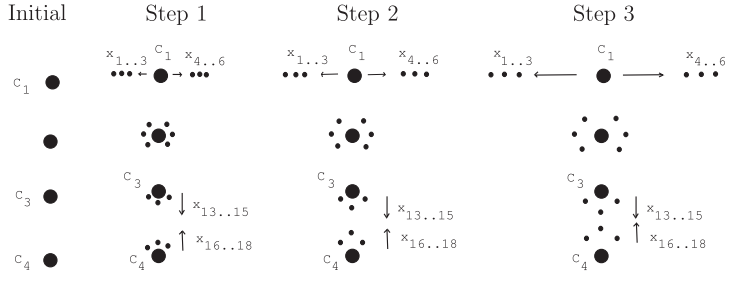


Fig. 5. Clustering that leads to non-optimal solution.

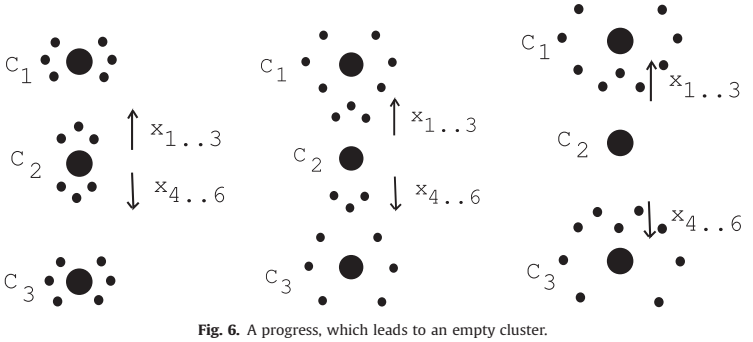


Fig. 6. A progress, which leads to an empty cluster.

through all the n points and find for them the nearest of the k centroids. So detecting empty clusters takes $O(kn)$ time.

For the empty clusters removal phase, we introduce two variants. The first is a one, which is more accurate, but slower, $O(k^2n)$ in time complexity. The second is a faster variant with $O(kn)$ time complexity. We present now first the accurate and then the fast variant.

Accurate removal: For the removal phase, there are k centroids, and therefore, at most $k-1$ empty clusters. Each empty cluster is replaced by a new location from one of the n datapoints. The new location is chosen so that it belongs to a cluster with more than one point. To find such a location takes $O(k)$ time in the worst case. The number of points in a cluster is calculated in the detection phase. Also, the new location is chosen so that there is not another centroid in that location. To check this it takes $O(k)$ time per location. After changing centroid location we have to detect again empty clusters. This loop together with the detection we repeat until all the at most $k-1$ empty clusters are filled. So the total time complexity for empty cluster removals is $O(k^2n)$.

Fast removal: In the detection phase, also the number of points per cluster and the nearest data points from the centroids of the non-empty clusters are calculated. The subphases of the removal are as follows:

- Move the centroids of the non-empty clusters to the calculated nearest data points, $T_1 = O(k)$.
- For all the $< k$ centroids, that form the empty clusters:
  - choose the biggest cluster, that has more than one data point, $T_2 = O(k)$.
  - choose the first free data point from this cluster, and put the centroid there, $T_3 = O(n)$.
  - re-partition this cluster, $T_4 = O(n)$.

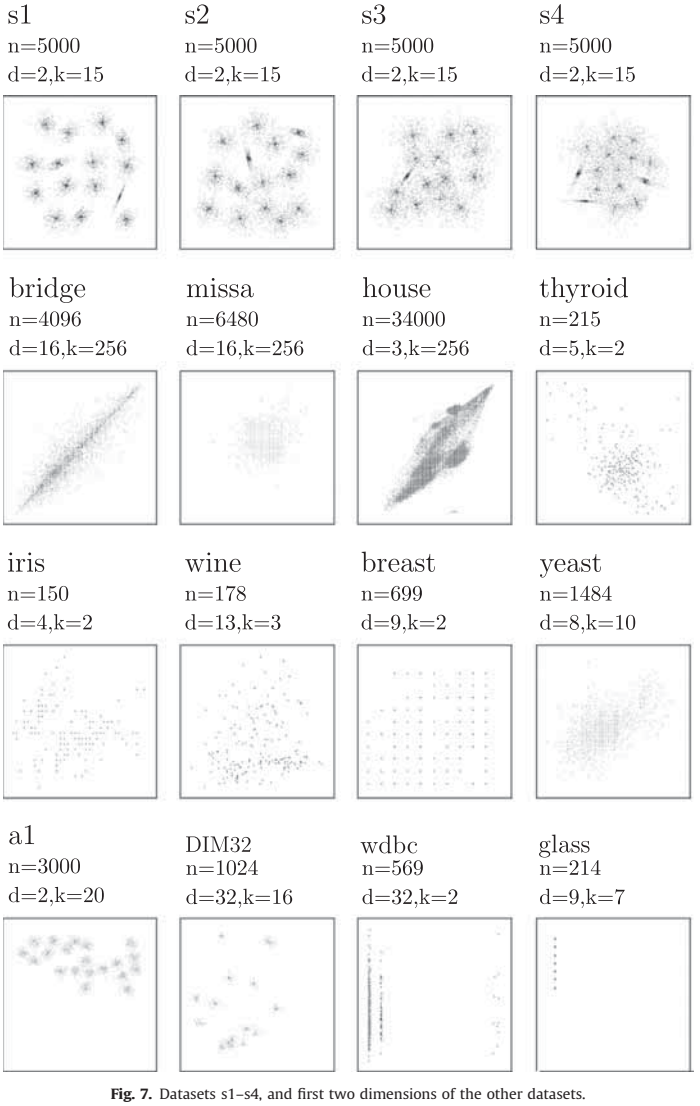


Fig. 7. Datasets s1–s4, and first two dimensions of the other datasets.

The total time complexity of removals is $T_1+k\cdot(T_2+T_3+T_4)=O(kn)$. This variant suffers somewhat from the fact that the centroids are moved to their nearest datapoints to ensure non-empty clusters.

Theoretically, k-means is the bottleneck of the algorithm. In the worst case, it takes $O(kn^{kd+1})$ time, which results in total time complexity of $O(n^{kd+1})$ when k is constant. This over-estimates the expected time complexity, which in practice, can be significantly lower. By limiting the number of k-means iterations to a constant, the time complexity reduces to linear time $O(n)$, when k is constant. When k equals to $\sqrt{n}$, the time complexity is $O(n^{1.5})$.

3. Experimental results

We ran the algorithm with a different number of steps and for several datasets. For MSE calculation we use the formula

$$\mathrm{MSE}=\frac{\sum_{j=1}^{k}\sum_{X_i\in C_j}\|X_i-C_j\|^2}{n\cdot d},$$

where MSE is normalized per feature. Some of the datasets used in the experiments are plotted in Fig. 7. All the datasets can be found in the SIPU web page http://cs.uef.fi/sipu/datasets. Some intermediate datasets and codebooks for a subset of a3 were plotted already in Fig. 4. The sets s1, s2, s3 and s4 are artificial datasets consisting of Gaussian clusters with the same variance but increasing overlap. Given 15 seeds, data points are randomly generated around them. In a1 and DIM sets the clusters are clearly separated whereas in s1–s4 they are more overlapping. These sets are chosen because they are still easy enough for a good algorithm to find the clusters correctly but hard enough for a bad algorithm to fail. We performed several runs by varying the number of steps between 1..20, 1000, 100,000, and 500,000. Most relevant results are collected in Table 3, and the results for the number of steps 2..20 are plotted in Fig. 8.

From the experience we observe that 20 steps are enough for this algorithm (Fig. 8 and Table 3). Many clustering results of these datasets stabilize at around 6 steps. More steps give only a marginal additional benefit, but at the cost of longer execution time. For some of the datasets, even just 1 step gives the best result. In these cases, initial positions for centroids just happen to be good. Phases of clustering show that 1 step gives as good result as 2 steps for a particular run for a particular dataset (Fig. 9). When the number of steps is large, the results sometimes get worse, because the codebook stays too tightly in a local optimum and the change of dataset is too marginal.

We tested the algorithm against k-means, k-means++ [7], global k-means [14] and repeated k-means. As a comparison, we made also runs with alternative structures. The results indicate that, on average, the best structures are the initial structure used in k-means++ and the random, see Table 4. The proposed algorithm with the k-means++-style initialization structure is better than k-means++ itself in the case of 15 out of 19 datasets. For one dataset the results are equal and for three datasets it is worse. These results show that the proposed algorithm is favorable to k-means++. The individual cases when it fails are due to statistical reasons. A clustering algorithm cannot be guaranteed to be better than other in every case. In real-world applications, k-means is often applied by repeating it several times starting from different random initializations and the best solution is kept finally. The intrinsic difference between our approach and the above trick is that we use educated calculation to obtain the centroids to current step, where the previous steps contribute to the current step, whereas repeated k-means initializes randomly at every repeat. From Table 5, we can see that the proposed algorithm is significantly better than k-means and k-means++. In most cases, it competes equally with repeated k-means, but in the case of high dimensionality datasets it works significantly better.

For high-dimensional clustered data, k-means++-style initial structure works best. We therefore recommend this initialization for high-dimensional unknown distributions. In most other cases, the random structure is equally good and can be used as an alternative, see Table 4.

Overall, different initial artificial structures lead to different clustering results. Our experiments did not reveal any unsuccessful cases in this. The worst results were obtained by random structure with optimal partition, but even for it, the results were at the same level as that of k-means. We did not observe any systematic dependency between the result and the size, dimensionality or type of data.

The method can also be considered as a post-processing algorithm similarly as k-means. We tested the method with the initial structure given by (complete) k-means, (complete) k-means++ and by Random Swap [12] (one of the best methods available). Results for these have been added in Table 6. We can see that the results for the proposed method using Random Swap as preprocessing are significantly better than running Repeated k-means.

We calculated also Adjusted Rand index [21], Van Dongen index [22] and Normalized mutual information index [23], to validate the clustering quality. The results in Table 7 indicate that the proposed method has a clear advantage over k-means.

Finding optimal codebook with high probability is another important goal of clustering. We used dataset s2 to compare the results of

Table 3
MSE for dataset s2 as a function of number of steps. K-means++-style structure. Mean of 200 runs except when steps $\geq$ 1000. (*) estimated from the best known result in [11].

Number of steps (k-means*) or repeats (repeated k-means)	K-means*		Repeated k-means	
	MSE ($\times 10^9$)	Time	MSE ($\times 10^9$)	Time
2	1.55	1 s	1.72	0.1 s
3	1.54	1 s	1.65	0.1 s
4	1.57	1 s	1.56	0.1 s
20	1.47	2 s	1.35	1 s
100	1.46	5 s	1.33	3 s
1000	1.45	24 s	1.33	9 s
100,000	1.33	26 min	1.33	58 min
500,000	1.33	128 min	1.33	290 min
K-means	1.94	0.2 s		
Global k-means	1.33	6 min		
Fast global k-means	1.33	3 s		
Optimal*	1.33	N/A		

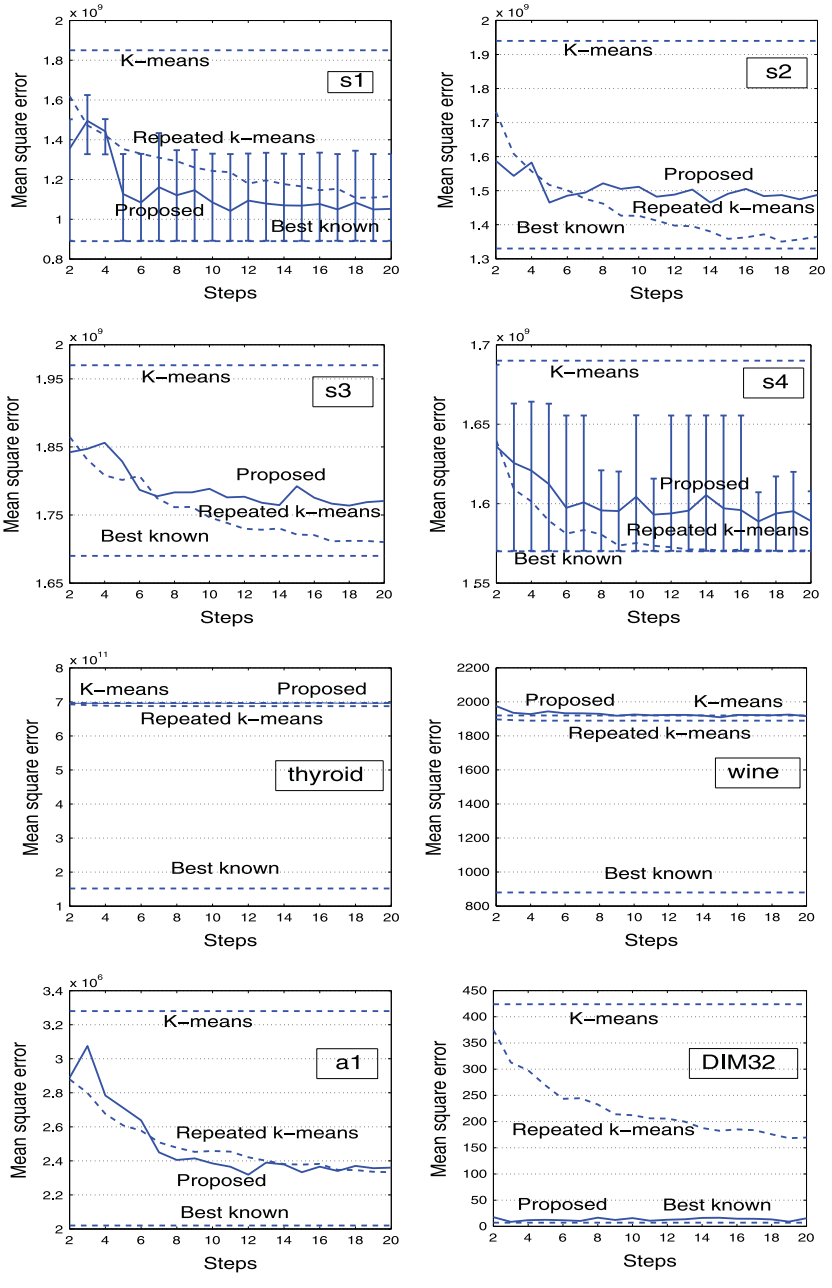


Fig. 8. Results of the algorithm (average over 200 runs) for datasets s1, s2, s3, s4, thyroid, wine, a1 and DIM32 with a different number of steps. For repeated *k*-means there are equal number of repeats than there are steps in the proposed algorithm. For s1 and s4 sets also 75% error bounds are shown. Step size 20 will be selected.

the proposed algorithm (using 20 steps), and results of the *k*-means and *k*-means++ algorithms to the known ground truth codebook of s2. We calculated how many clusters are mis-located, i.e., how many swaps of centroids would be needed to correct the global allocation of a codebook to that of the ground truth. Of the 50 runs, 18 ended up to the optimal allocation, whereas *k*-means succeeded only with 7 runs, see Table 8. Among these 50 test runs the proposed algorithm had never more than 1 incorrect cluster allocation, whereas *k*-means had up to 4 and *k*-means++ had up to 2 in the worst case. Fig. 10 demonstrates typical results.

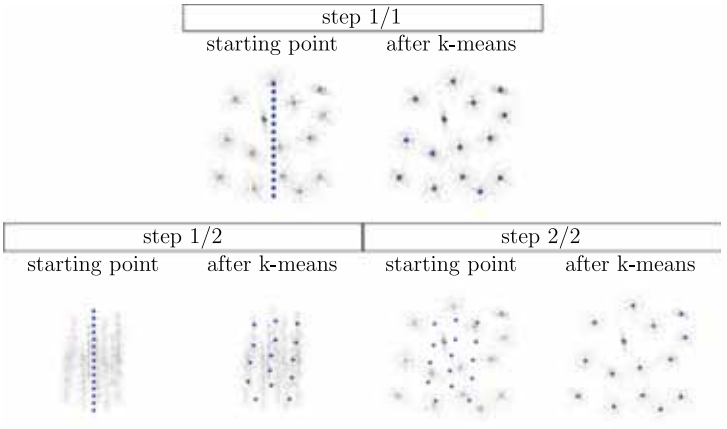


Fig. 9. Phases of clustering for 1 step and 2 steps for dataset s2.

Table 4
MSE for different datasets, averages over several (≥ 10) runs, 10 or 20 steps are used. Most significant digits are shown.

Dataset	*K*-means*					
	Diagonal	Line	Random	*k*-means++ style	Random + optimal partition	Line with uneven clusters
s1	1.21	**1.01**	1.22	1.05	1.93	1.04
s2	1.65	1.52	1.41	**1.40**	2.04	1.46
s3	1.75	**1.71**	1.74	1.78	1.95	1.73
s4	1.67	1.63	1.60	**1.59**	1.70	1.64
a1	2.40	2.40	2.35	2.38	3.07	**2.25**
DIM32	151	136	64	**7.10**	517	113
DIM64	98	168	65	**3.31**	466	157
DIM128	153	92	101	**2.10**	573	132
DIM256	135	159	60	**0.92**	674	125
Bridge	**165**	**165**	**165**	**165**	167	168
Missa	**5.11**	5.15	5.24	5.19	5.32	5.16
House	9.67	**9.48**	9.55	9.49	9.80	9.88
Thyroid	6.93	**6.92**	6.96	6.96	6.98	**6.92**
Iris	**2.33**	**2.33**	**2.33**	2.42	2.42	**2.33**
Wine	**1.89**	1.90	**1.89**	1.93	1.92	**1.89**
Breast	**3.13**	**3.13**	**3.13**	**3.13**	**3.13**	**3.13**
Yeast	0.044	0.051	**0.037**	0.041	0.039	0.050
wdbc	**2.62**	**2.62**	**2.62**	**2.62**	**2.62**	**2.62**
Glass	**0.22**	0.23	**0.22**	**0.22**	0.23	0.24
Best	7	8	7	10	2	6

Table 5
MSE for different datasets, averages over several (≥ 10) runs. Most significant digits are shown. (*) The best known results are obtained from among all the methods or by 2 h run of random swap algorithm [12].

Dataset	Dimensionality	*K*-means	Repeated *k*-means	*K*-means++	*K*-means* (proposed)	Fast GKM	Best known*
s1	2	1.85	1.07	1.28	1.05	**0.89**	**0.89**
s2	2	1.94	1.38	1.55	1.40	**1.33**	**1.33**
s3	2	1.97	1.71	1.95	1.78	**1.69**	**1.69**
s4	2	1.69	**1.57**	1.70	1.59	**1.57**	**1.57**
a1	2	3.28	2.32	2.66	2.38	**2.02**	**2.02**
DIM32	32	424	159	7.18	**7.10**	**7.10**	**7.10**
DIM64	64	498	181	3.39	**3.31**	3.39	**3.31**
DIM128	128	615	276	2.17	**2.10**	2.17	**2.10**
DIM256	256	671	296	0.99	**0.92**	0.99	**0.92**
Bridge	16	168	166	177	165	164	**161**
Missa	16	5.33	5.28	5.62	5.19	5.34	**5.11**
House	3	9.88	9.63	6.38	9.49	5.94	**5.86**
Thyroid	5	6.97	6.88	6.96	6.96	**1.52**	**1.52**
Iris	4	3.70	2.33	2.60	2.42	**2.02**	**2.02**
Wine	13	1.92	1.89	0.89	1.93	**0.88**	**0.88**

Table 5 (*continued*)

Dataset	Dimensionality	*K*-means	Repeated *k*-means	*K*-means++	*K*-means* (proposed)	Fast GKM	Best known*
Breast	9	**3.13**	**3.13**	3.20	**3.13**	**3.13**	**3.13**
Yeast	8	0.0041	**0.0038**	0.061	0.041	0.0039	**0.0038**
wdbc	31	2.62	2.61	**1.28**	2.62	2.62	**1.28**
Glass	9	0.16	**0.15**	0.28	0.22	0.16	**0.15**
Best		1	4	1	5	10	19

Table 6
MSE for *k*-means* as postprocessing, having different clustering algorithms as preprocessing. Averages over 20 runs, 20 steps are used. Most significant digits are shown.

Dataset	Repeated *k*-means	*K*-means*			
		K-means	*K*-means++	Random swap, 20 swap trials	Random swap, 100 swap trials
s1	1.07	0.99	1.08	0.99	**0.89**
s2	1.38	1.53	1.51	1.46	**1.33**
s3	1.71	1.80	1.76	1.77	**1.69**
s4	**1.57**	1.58	1.59	1.59	**1.57**
a1	2.32	2.54	2.37	2.31	**2.02**
DIM32	159	79.4	11.68	44.8	**7.10**
DIM64	181	59.4	**3.31**	48.5	9.35
DIM128	276	44.7	**2.10**	67.9	**2.10**
DIM256	296	107.1	**0.92**	16.2	16.5
Bridge	166	**164**	**164**	**164**	**164**
Missa	5.28	5.20	5.19	5.19	**5.18**
House	9.63	9.43	9.42	9.42	**9.30**
Thyroid	**6.88**	6.95	6.93	6.89	**6.88**
Iris	**2.33**	**2.33**	**2.33**	2.38	**2.33**
Wine	**1.89**	1.93	1.93	1.90	**1.89**
Breast	**3.13**	**3.13**	**3.13**	**3.13**	**3.13**
Yeast	**0.0038**	0.042	0.040	0.039	**0.0038**
wdbc	**2.61**	2.62	2.62	2.62	2.62
Glass	**0.15**	0.21	0.21	0.21	**0.15**
Best	8	3	6	2	16

Table 7
Adjusted Rand, Normalized Van Dongen and NMI indices for s-sets. Line structure (Rand), *K*-means++ initialization structure (NVD and NMI), 10 steps, mean of 30 runs (Rand) and mean of 10 runs (NVD and NMI). Best value for Rand is 1, for NVD it is 0 and for NMI it is 1.

Adjusted rand

Dataset	*k*-means	Proposed	GKM
s1	0.85	0.98	1.00
s2	0.86	0.93	0.99
s3	0.83	0.95	0.96
s4	0.83	0.87	0.94

NMI

Dataset	*k*-means	Proposed	GKM
s1	0.94	0.98	1.00
s2	0.96	0.97	0.99
s3	0.91	0.93	0.97
s4	0.91	0.93	0.95

Normalized Van Dongen

Dataset	*k*-means	GKM	Proposed
s1	0.08	0.03	0.001
s2	0.04	0.04	0.004
s3	0.09	0.06	0.02
s4	0.09	0.04	0.03

Table 8
Occurrences of wrong clusters obtained by the *k*-means, *k*-means++ and proposed algorithms in 50 runs for s2.

Incorrect clusters	*K*-means (%)	*k*-means++ (%)	Proposed (line structure) (%)
0	14	28	36
1	38	60	64
2	34	12	0
3	10	0	0
4	2	0	0
Total	100	100	100

The reason why the algorithm works well is that starting from an artificial structure, we have an optimal clustering. Then, when making the gradual inverse transform, we do not have to optimize the structure of clustering (it is already optimal). It is enough that the data points move one by one from clusters to others by *k*-means operations. The operation is the same as in *k*-means, but the clustering of the starting point is already optimal. If the structure remains optimal during the transformation, an optimal result will be obtained. Bare *k*-means cannot do this except only in special cases, that is usually is tried to compensate by using Repeated *k*-means or *k*-means++.

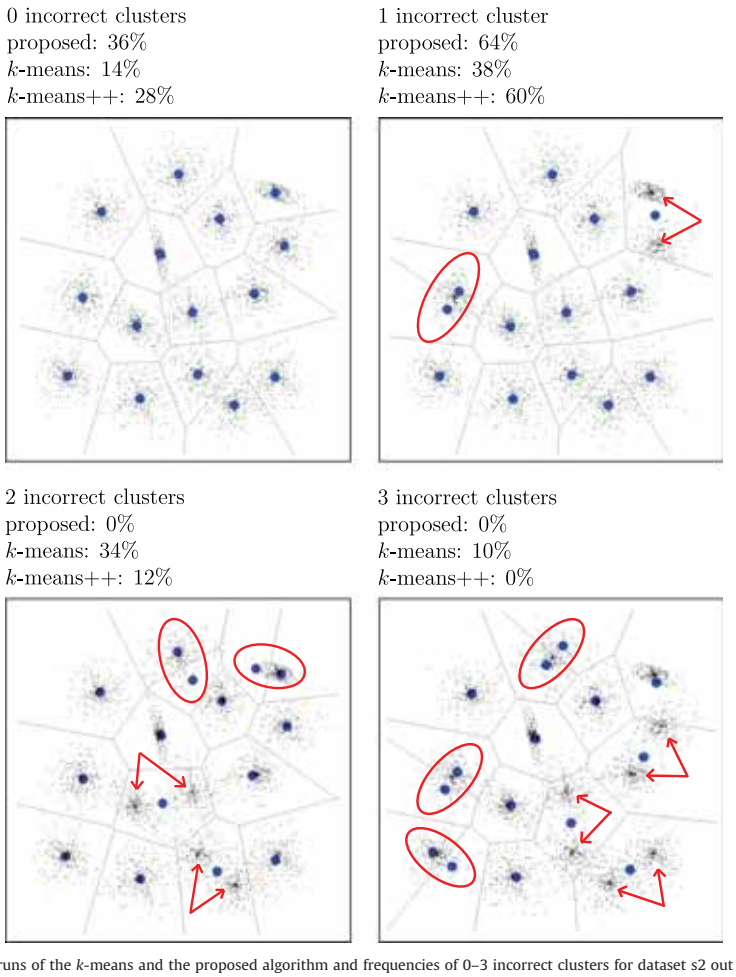


Fig. 10. Sample runs of the *k*-means and the proposed algorithm and frequencies of 0–3 incorrect clusters for dataset *s2* out of 50 test runs.

4. Conclusions

We have proposed an alternative approach for clustering by fitting the data to the clustering model and not vice versa. Instead of solving the clustering problem as such, the problem is to find a proper inverse transform from the artificial data with optimal cluster allocation, to the original data. Although it cannot solve all pathological cases, we have demonstrated that the algorithm, with a relatively simple design, can solve the problem in many cases.

The method is designed as a clustering algorithm where the initial structure is not important. We only considered simple structures, of which the initialization of *k*-means++ is most complicated (note that entire *k*-means++ is not applied). However, it could also be considered as a post-processing algorithm similarly as *k*-means. But then it is not limited to be post-processing to *k*-means++ but for any other algorithm.

Future work is how to optimize the number of steps in order to avoid extensive computation but still retain the quality. Adding randomness to the process could also be used to avoid the pathological cases. The optimality of these variants and their efficiency in comparison to other algorithms have also theoretical interest.

Conflict of interest statement

None declared.

References

[1] D. Aloise, A. Deshpande, P. Hansen, P. Popat, NP-hardness of Euclidean sum-of-squares clustering, Mach. Learn. 75 (2009) 245–248.
[2] M. Inaba, N. Katoh, H. Imai, Applications of weighted voronoi diagrams and randomization to variance-based *k*-clustering, in: Proceedings of the 10th Annual ACM Symposium on Computational Geometry (SCG 1994), 1994, pp. 332–339.
[3] J. MacQueen, Some methods of classification and analysis of multivariate observations, in: Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability, vol. 1, 1967, pp. 281–296.
[4] D. Arthur, S. Vassilvitskii, How slow is the *k*-means method?, in: Proceedings of the 2006 Symposium on Computational Geometry (SoCG), 2006, pp. 144–153.
[5] 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) (1999) 1027–1040.
[6] D. Steinley, M.J. Brusco, Initializing K-means batch clustering: a critical evaluation of several techniques, J. Class. 24 (1) (2007) 99–121.
[7] D. Arthur, S. Vassilvitskii, k-means++: the advantages of careful seeding, in: SODA '07: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2007, pp. 1027–1035.

[8] D. MacKay, "Chapter 20. An Example Inference task: Clustering", in: Information Theory, Inference and Learning Algorithms, Cambridge University Press, Cambridge, 2003, pp. 284–292.
[9] W.H. Equitz, A new vector quantization clustering algorithm, IEEE Trans. Acoust. Speech Signal Process. 37 (1989) 1568–1575.
[10] P. Fränti, O. Virmajoki, V. Hautamäki, Fast agglomerative clustering using a k-nearest neighbor graph, IEEE Trans. Pattern Anal. Mach. Intell. 28 (11) (2006) 1875–1881.
[11] P. Fränti, O. Virmajoki, Iterative shrinking method for clustering problems, Pattern Recognit. 39 (5) (2006) 761–765.
[12] P. Fränti, J. Kivijärvi, Randomized local search algorithm for the clustering problem, Pattern Anal. Appl. 3 (4) (2000) 358–369.
[13] D. Pelleg, A. Moore, X-means: extending k-means with efficient estimation of the number of clusters, in: Proceedings of the Seventeenth International Conference on Machine Learning, Morgan Kaufmann, San Francisco, 2000, pp. 727–734.
[14] A. Likas, N. Vlassis, J. Verbeek, The global k-means clustering algorithm, Pattern Recognit. 36 (2003) 451–461.
[15] A.P. Dempster, N.M. Laird, D.B. Rubin, Maximun likelihood from incomplete data via the EM algorithm, J. R. Stat. Soc. B 39 (1977) 1–38.
[16] Q. Zhao, V. Hautamäki, I. Kärkkäinen, P. Fränti, Random swap EM algorithm for finite mixture models in image segmentation, in: 16th IEEE International Conference on Image Processing (ICIP), 2009, pp. 2397–2400.
[17] C.H.Q. Ding, X. He, H. Zha, M. Gu, H.D. Simon, A min-max cut algorithm for graph partitioning and data clustering, in: Proceedings of IEEE International Conference on Data Mining (ICDM), 2001, pp. 107–114.
[18] M.I. Malinen, P. Fränti, Clustering by analytic functions, Inf. Sci. 217 (0) (2012) 31–38.
[19] A. Ahmadi, F. Karray, M.S. Kamel, Model order selection for multiple cooperative swarms clustering using stability analysis, Inf. Sci. 182 (1) (2012) 169–183.
[20] C. Zhong, D. Miao, P. Fränti, Minimum spanning tree based split-and-merge: a hierarchical clustering method, Inf. Sci. 181 (16) (2011) 3397–3410.
[21] L. Hubert, P. Arabie, Comparing partitions, J. Class. 2 (1) (1985) 193–218.
[22] S. van Dongen, Performance criteria for graph clustering and markov cluster experiments, Technical Report INSR0012, Centrum voor Wiskunde en Informatica, 2000.
[23] N. Vinh, J. Epps, J. Bailey, Information theoretic measures for clusterings comparison: variants, properties, normalization and correction for change, J. Mach. Learn. Res. 11 (2010) 2837–2854.

Mikko Malinen received the B.Sc. and M.Sc. degrees in communications engineering from Helsinki University of Technology, Espoo, Finland, in 2006 and 2009, respectively. Currently he is a doctoral student at the University of Eastern Finland. His research interests include data clustering and data compression.

Radu Mariescu-Istodor received the B.Sc. degree in information technology from West University of Timisoara, Romania, in 2011 and M.Sc. degree in computer science from University of Eastern Finland, in 2013. Currently he is a doctoral student at the University of Eastern Finland. His research includes data clustering and GPS trajectory analysis.

Pasi Fränti received his MSc and PhD degrees in computer science from the University of Turku, Finland, in 1991 and 1994, respectively. From 1996 to 1999 he was a postdoctoral researcher funded by the Academy of Finland. Since 2000, he has been a professor in the University of Eastern Finland (Joensuu) where he is leading the speech & image processing unit (SIPU).

Prof. Fränti has published over 50 refereed journal and over 130 conference papers. His primary research interests are in clustering, image compression and mobile location-based applications.

Paper **III**

M. I. Malinen and P. Fränti
"All-pairwise squared distances
lead to balanced clustering"
(manuscript), 2015.

All-pairwise Squared Distances Lead to More Balanced Clustering

Mikko I. Malinen[1,*], Pasi Fränti[1]

Speech and Image Processing Unit, School of Computing, University of Eastern Finland, Box 111, FIN-80101 Joensuu, FINLAND

Abstract

All pairwise squared distances has been used as a cost function in clustering. In this paper, we show that it will lead to more balanced clustering than centroid-based distance functions like in k-means. It is formulated as a cut-based method, and it is closely related to MAX k-CUT method. We introduce two algorithms for the problem which are both faster than the existing one based on l_2^2-Stirling approximation. The first algorithm uses semidefinite programming as in MAX k-CUT. The second algorithm is an on-line variant of classical k-means. We show by experiments that the proposed approach provides better overall joint optimisation of mean squared error and cluster balance than the compared methods.

*Corresponding author

Email addresses: mmali@cs.uef.fi (Mikko I. Malinen), franti@cs.uef.fi (Pasi Fränti)

URL: http://cs.uef.fi/∼mmali (Mikko I. Malinen), http://cs.uef.fi/pages/franti/ (Pasi Fränti)

[1]The authors are with the University of Eastern Finland.

Keywords:

Clustering, Balanced clustering, Squared cut, Scut, MAX k-CUT problem, Semidefinite programming

1. Introduction

1.1. K-means clustering

Euclidean sum-of-squares clustering (k-means clustering) is an NP-hard problem [1] where one groups n points into k clusters. Clusters are represented by *centre points (centroids)* and the aim is to minimise the *mean squared error* (*MSE*), calculated as the mean distance of points from their nearest centroid. K-means clustering minimises

$$\text{Cost} = TSE_1 + TSE_2 + ... + TSE_k, \tag{1}$$

where TSE_i is the total squared error of the ith cluster. This can also be written as

$$\text{Cost} = n_1 \cdot MSE_1 + n_2 \cdot MSE_2 + ... + n_k \cdot MSE_k, \tag{2}$$

where n_i is the number of points and MSE_i is the mean squared error of ith cluster. TSE and MSE for a single cluster j are calculated as

$$TSE_j = \sum_{X_i \in C_j} ||X_i - C_j||^2 \tag{3}$$

$$MSE_j = \frac{TSE_j}{n_i}. \tag{4}$$

When k is constant, the clustering problem can be solved in polynomial $O(n^{kd+1})$ time [2]. Although polynomial, this is slow, and suboptimal algorithms are therefore used. K-means algorithm [3] is fast and simple, although its worst-case number of iterations is $O(n^{kd})$. The advantage of k-means is that it solves a local optimum starting from any initial centroid locations by a simple iterative two-step procedure. A drawback of k-means is that it cannot always solve the optimal global clustering structure. By optimised global clustering structure we mean centroid locations from which the global optimum can be solved by k-means. This is the main reason why slower agglomerative clustering [4, 5, 6], or more complex k-means variants [7, 8, 9, 10] are used. Gaussian mixture models (EM algorithm) [11, 12] and cut-based methods have also been found to give competitive results [13]. Recent research has considered clustering by analytic function [14], and trying to fit

the data into the model instead of fitting the model to the data [15].

1.2. Balanced clustering

Sometimes a balanced clustering result is desirable. Balanced clustering is defined as a clustering where the points are evenly distributed into the clusters. In other words, every cluster includes either $\lfloor n/k \rfloor$ or $\lceil n/k \rceil$ points. We define a balanced clustering as a problem which aims at maximizing balance and minimising some other cost function such as *MSE*. Balanced clustering is desirable in workload-balancing algorithms. For example, one algorithm to multiple traveling salesman problem [16] clusters the cities so that each cluster is solved by one salesman. It is desirable that each salesman has an equal workload.

Balanced clustering, in general, is a 2-objective optimisation problem, in which two aims contradict each other: to minimise a cost function such as *MSE*, and to balance cluster sizes at the same time. Traditional clustering aims at minimising *MSE* completely without considering cluster size balance. Balancing, on the other hand, would be trivial if we did not care about *MSE* simply by dividing vectors into equal size clusters randomly. For optimizing both, there are two approaches: *Balance-constrained* and *balance-driven*

clustering.

In balance-constrained clustering, cluster size balance is a mandatory requirement that must be met, and minimising *MSE* is a secondary criterion. In balance-driven clustering, balanced clustering is an aim, but it is not mandatory. It is a compromize between the two goals, namely the balance and the *MSE*. The solution is a weighted cost function between *MSE* and the balance, or it is a heuristic, which aims at minimising *MSE* but indirectly creates a more balanced result than optimizing *MSE* alone.

Next, we review existing methods that aim at balanced clustering. Bradley et al. [17] present a constrained k-means algorithm in which the assignment step of k-means is implemented as a linear program in which a minimum number of points in a cluster is set as a constraint. In our recent paper [18] we present balanced k-means algorithm which has fixed size clusters. It solves the k-means assignment step as an assignment problem. The method in [19] tries to find a partition close to the given partition, but so that cluster size constraints are fulfilled. Banerjee and Ghosh [20] present an algorithm based on frequency sensitive competitive learning (FSCL) where the centroids com-

pete for points. It multiplicatively scales the error (distance from the data point to the centroid) by the number of times that a centroid has won in the past. Thus, bigger clusters are less likely to gain points in the future. Althoff et al. [21] uses FSCL, but their solution incorporates additive distance bias instead of multiplicative distance bias. They report that their algorithm is more stable for high-dimensional feature spaces. Banerjee and Ghosh [22] introduced a fast ($O(kNlogN)$) algorithm for balanced clustering in three steps: sampling the given data, clustering the sampled data and populating the clusters with the data points that were not sampled in the first step. *Size regularized cut* SRCut [23] is defined as the sum of the inter-cluster similarity and a regularization term measuring the relative size of two clusters. In [24] there is a balancing aiming term in cost function. There are also application-based solutions in networking [25], which aim at network load balancing, where clustering is done by self-organization without central control. In [26], energy-balanced routing between sensors is aimed so that most suitable balanced amount of nodes will be the members of the clusters.

Classification of some algorithms into these two classes can be found from Table 1.

Table 1: Classification of some balanced clustering algorithms.

Balance-constrained	Type
Balanced k-means [18]	k-means
Constrained k-means [17]	k-means
Size constrained [19]	integer linear programming
Balance-driven	**Type**
FSCL [20]	assignment
FSCL additive bias [21]	assignment
Cluster sampled data [22]	k-means
Ratio cut [27]	divisive
Ncut [28]	divisive
Mcut [13]	divisive
SRcut [23]	divisive
Submodular fractional programming [24]	submodular fractional programming

1.3. Cut-based methods

Cut-based clustering is a process where the dataset is cut into smaller parts based on similarity $S(X_l, X_s)$ or cost $d(X_l, X_s)$ between pairs of points. By $\text{cut}(A, B)$ one means partitioning a dataset into two parts A and B, and the value of $\text{cut}(A, B)$ is the total weight between all pairs of points between the sets A and B:

$$\text{cut}(A, B) = \sum_{X_l \in A, X_s \in B} w_{ls}. \tag{5}$$

The weigths w can be defined either as distances or similarities between the two points. Unless otherwise noted, we use (squared) Euclidean distances

in this paper. The $\text{cut}(A, B)$ equals the total pairwise weights of $A \cup B$ subtracted by the pairwise weights within the parts A and B:

$$\text{cut}(A, B) = W - W(A) - W(B), \tag{6}$$

where

$$W = \sum_{l=1}^{n-1} \sum_{s=i+1}^{n} w_{ls}, \tag{7}$$

$$W(A) = \sum_{X_l \in A, X_s \in A} w_{ls}, \tag{8}$$

and $W(B)$ respectively.

In cut-based clustering, two common objective functions are *Ratio cut* [27] and *Normalised cut, Ncut,* [28]. In these cost functions the weights are the similarities between the points. In Ratio Cut, the cost of a cut is normalised by the number of points n_A or n_B, while in Ncut it is normalised by similarities to all other points in the dataset. Both of these normalisations

favour balanced cuts [29], p.401. One minimises the following definitions:

$$\text{RatioCut}(A,B) = \frac{\text{cut}(A,\bar{A})}{n_A} + \frac{\text{cut}(B,\bar{B})}{n_B} \tag{9}$$

$$\text{Ncut}(A,B) = \frac{\text{cut}(A,\bar{A})}{assoc(A,X)} + \frac{\text{cut}(B,\bar{B})}{assoc(B,X)} \tag{10}$$

$$= \frac{\text{cut}(A,\bar{A})}{W(A) + \text{cut}(A,\bar{A})} + \frac{\text{cut}(B,\bar{B})}{W(B) + \text{cut}(B,\bar{B})} \tag{11}$$

where $\bar{A}$ is the complement point set of A, $\bar{B}$ is the complement point set of B, $W(A)$ is the total similarities between the pairs of points within cluster A, and the association $assoc(A,X)$ is the total similarities between points in partition A and all points:

$$assoc(A,X) = W(A) + \text{cut}(A,\bar{A}). \tag{12}$$

As an example, following the formulas (9) and (11), in Figure 1 the Ratio cut would be $\text{RatioCut}(A,B) = (0.33+0.25+0.50+0.50+0.33+0.33)/2 + (0.33+0.25+0.50+0.50+0.33+0.33)/3 \approx 1.87$ and the Ncut would be $\text{Ncut}(A,B) = (0.33+0.25+0.50+0.50+0.33+0.33)/(1+(0.33+0.25+0.50+0.50+0.33+0.33)) + (0.33+0.25+0.50+0.50+0.33+0.33)/(2.75+(0.33+0.25+0.50+0.50+0.33+0.33)) \approx 1.14$. Optimising the cost functions (9) and

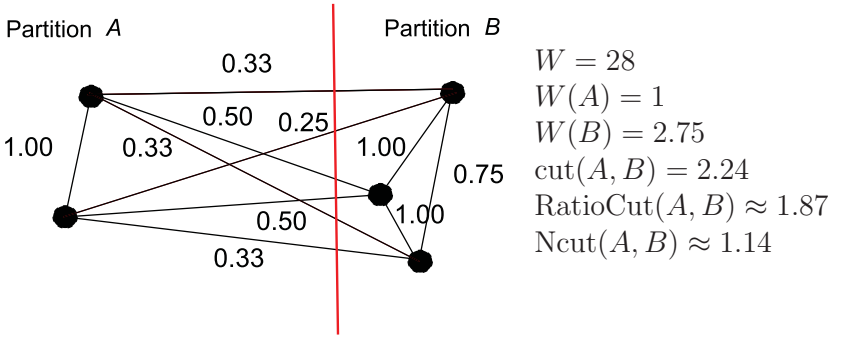


Figure 1: An example of a cut.

(11) aims at minimising the cuts (the numerators), while at the same time maximising the denominators. In practice, one approximates this problem by relaxation, i.e. solving a nearby easier problem. Relaxing Ncut leads to normalised spectral clustering, while relaxing RatioCut leads to unnormalised spectral clustering [29]. There exists also semidefinite programming -based relaxation for Ncut [30].

The paper [13] presents a cut-based clustering algorithm *Mcut*, which tends to make balanced clusters. In their algorithm, similarity of each pair of points is considered. They aim at minimising $\mathrm{cut}(A, B)$, the similarity of partitions A and B while maximising the similarities within the partitions

($W(A)$ and $W(B)$) at the same time. The cost function is

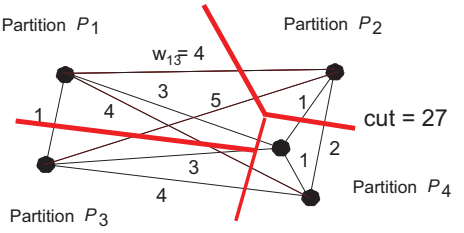
$$\mathrm{Mcut}(A,B) = \frac{\mathrm{cut}(A,B)}{W(A)} + \frac{\mathrm{cut}(A,B)}{W(B)}. \tag{13}$$

As an example, following the formula (13), in Figure 1 the Mcut is $\mathrm{Mcut}(A,B) = (0.33+0.25+0.50+0.50+0.33+0.33)/1 + (0.33+0.25+0.50+0.50+0.33+0.33)/2.75 \approx 3.05$.

The weakness of Ratio cut, Ncut and Mcut is that they cut the graph only in two parts. This implies that a divisive algorithm should be applied, which would result in unequal partition. For example balanced clustering with $k = 2$ would be 50%-50% partition of the points. Further dividing with the same criterion would end up to 50%-25%-25% partition. But we would like to have a method, which optimises the balance for the desired number of clusters jointly, which would enable a lower MSE.

1.4. MAX k-CUT method

In weighted MAX k-CUT problem [31] one partitions a graph into k subgraphs so that the sum of weights of edges between the subgraphs is maximised. The weights are distances. MAX k-CUT aims at partitioning



$$\begin{aligned}
&\text{cut}(P_1, \bar{P}_1) = 12\\
&\text{cut}(P_2, \bar{P}_2) = 12\\
&\text{cut}(P_3, \bar{P}_3) = 13\\
&\text{cut}(P_4, \bar{P}_4) = 17\\
&\textstyle\sum = 54.\ 1/2 \cdot 54 = 27
\end{aligned}$$

Figure 2: An example of MAX k-CUT, when $k = 4$.

the data into k clusters $P_1, ..., P_k$. Following the notation of Section 1.3 and writing factor $1/2$ in order to avoid summing the weights twice, the MAX k-CUT problem is defined as

$$\max_{P_j, 1 \leq j \leq k} \frac{1}{2} \sum_{j=1}^{k} \text{cut}(P_j, \bar{P}_j). \tag{14}$$

See an example of MAX k-CUT in Figure 2. This is an NP-hard problem [32] for general weights. No polynomial time exact algorithm is known to solve this.

If we use Euclidean distance for the weights of the edges between every pair of points, then taking optimal weighted MAX k-CUT results in minimum intra-cluster pairwise distances among any k-CUT. If we use *squared* distances as weights of the edges we end up with minimum intra-cluster pairwise squared distances. If we use squared Euclidean distances as weights, the

problem is expected to remain NP-hard.

1.5. Scut

In this paper we deal with *Squared cut, Scut* method, which uses all pairwise squared distances as cost function. This cost function has been presented in [33], where it is called l_2^2 k-clustering. However, we formulate it by using TSE's of clusters and show that the method leads to more balanced clustering problem than TSE itself. It is formulated as a cut-based method and it has been shown that it is a close relative to MAX k-CUT method [34]. We present two algorithms for the problem; both more practical than the exhaustive search proposed in [35] to l_2^2 k-clustering. The first algorithm is based on semidefinite programming similar to MAX k-CUT, and the second one is an on-line k-means algorithm directly optimising the cost function.

A general k-Clustering problem in [34] defines the cost by calculating all pairwise distances within the clusters for any arbitrary weighted graphs. The paper [36] studies the problem when distances satisfy the triangle inequality. Paper by Schulman [33] gave probabilistic algorithms for l_2^2 k-Clustering. The running time is linear if dimension $d = o(\log n/ \log\log n)$ but otherwise it is $n^{O(\log\log n)}$. De la Vega et al [35] improved and extended Schulman's result, giving a true polynomial time approximation algorithm for arbitrary dimen-

sion. However, even their algorithm is too slow in practise. We therefore present faster algorithms for the Squared cut method.

In Scut, we form the graph by assigning squared Euclidean distances as weights of the edges between every pair of points. In a single cluster j, intra-cluster pairwise squared distances $= n_j \cdot TSE_j$, see a proof in [37], p.52. This generalisation to all clusters is known as the Huygens' theorem, which states that total squared error (TSE) equals to the sum over all clusters, over all squared distances between pairs of entities within that cluster divided by its cardinality:

$$W(A_j) = n_{A_j} \cdot TSE(A_j) \quad \forall j$$

Huygens' theorem is crucial for our method because it relates the pairwise distances to intra-cluster TSE, and thus, to the Scut cost function:

$$\text{Scut} = n_1 \cdot TSE_1 + n_2 \cdot TSE_2 + ... + n_k \cdot TSE_k, \tag{15}$$

where n_j is the number of points and TSE_j is the total squared error of the

Algorithm 1 Scut
Input: dataset X, number of clusters k
Output: partitioning of points P

for each edge of the graph **do**
  Weight of edge $w_{ij} \leftarrow$ Euclidean_distance$(X_i, X_j)^2$
end for
Approximate MAX k-CUT.
Output partitioning of points P.

jth cluster. Based on (3), this may also be written as

$$\text{Scut} = n_1^2 \cdot MSE_1 + n_2^2 \cdot MSE_2 + ... + n_k^2 \cdot MSE_k, \tag{16}$$

where MSE_j is the mean squared error of jth cluster. In cut-notation the cost function is total pairwise weights minus the value of MAX k-CUT:

$$\text{Scut} = W - \max_{P_j, 1 \le j \le k} \frac{1}{2} \sum_{i=1}^{k} \text{cut}(P_j, \bar{P}_j). \tag{17}$$

From this we conclude that using squared distances and optimising MAX k-CUT results in optimisation of the Scut cost function (15). For approximating Scut, the Algorithm 1 can be used. Our cut-based method has an *MSE*-based cost function and it tends to balance clusters because of the n_j^2 factors in (16). This can be seen by the following simple example where two clusters have the same squared error: $MSE_1 = MSE_2 = MSE$ (Figure 3). Total er-

Figure 3: Two different sized clusters with the same MSE.

ror of these are $2^2 \cdot MSE_1 = 4 \cdot MSE$, and $10^2 \cdot MSE_2 = 100 \cdot MSE$. Adding one more point would increase the error by $(n+1)^2 \cdot MSE - n^2 \cdot MSE = (2n+1) \cdot MSE$. In the example in Figure 3, the cost would increase by $5 \cdot MSE$ (cluster 1) and $21 \cdot MSE$ (cluster 2). The cost function therefore always favours putting points into a smaller cluster, and therefore, it tends to make more balanced clusters. Figure 4 demonstrates the calculation of the cost.

2. Approximating Scut

2.1. Approximation algorithms

The weighted MAX k-CUT is an NP-hard problem and it can be solved by an approximation algorithm based on *semidefinite programming* (SDP)

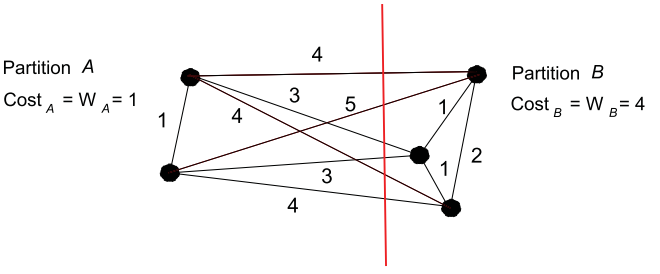


Figure 4: Calculation of the cost. Edge weights are squared Euclidean distances.

in polynomial time [31]. Although polynomial time the algorithm is slow. According to our experiments it can only be used for datasets with just over 150 points. A faster approximation algorithm has been presented in [32]. It begins with an arbitrary partitioning of the points, and moves a point from one subset to another if the sum of weights of edges across different subsets decreases. The algorithm stops when no further improvements can be attained by all possible moving of one point. In Section 2.3, we will propose even a faster algorithm, which instead of maximising MAX k-CUT, it minimises the Scut cost function (15). Nevertheless, the result will be the same as that of MAX k-CUT.

2.2. Approximation ratio

The goodness of all of these three approximation algorithms is $\alpha_k > 1 - k^{-1}$. The constant α_k is a goodness measure of MAX k-CUT approximation

algorithm, and corresponds to the goodness of the sum of inter-cluster edge weights that are cut away. To be able to say what the goodness is with respect to the cost function (15), we need to calculate the remaining intra-cluster edge weights. For this we need to calculate the dataset-specific total sum of pairwise weights W. This corresponds to weights when nothing has been cut off. It is calculated by treating the whole dataset as one cluster. Goodness with respect to the cost function can then be calculated by the following analysis (see also Figure 5). ϵ_k is the value of the cost function by approximation divided by optimal value of the cost function. That is

$$\begin{aligned} \epsilon_k &= \frac{W - w(\mathrm{P}(k))}{W - w(\mathrm{P}(k)^*)} \\ &= \frac{W - w(\mathrm{P}(k))}{\max(0, W - \frac{1}{\alpha_k} \cdot w(\mathrm{P}(k)))} \end{aligned} \tag{18}$$

For example, since we have a lower bound $\alpha_k > 1 - k^{-1}$, we get an upper bound for ϵ_k. Then $\epsilon_k > 1$. This ϵ_k can be treated as an expected approximation ratio for the proposed algorithm. However, it is dataset-specific. In practise, the denominator in equation (18) becomes zero in all the cases we tried, so in these cases all what can be said is $\epsilon_k < \infty$. Thus, we have to be satisfied with having α_k, the approximation ratio for MAX k-CUT.

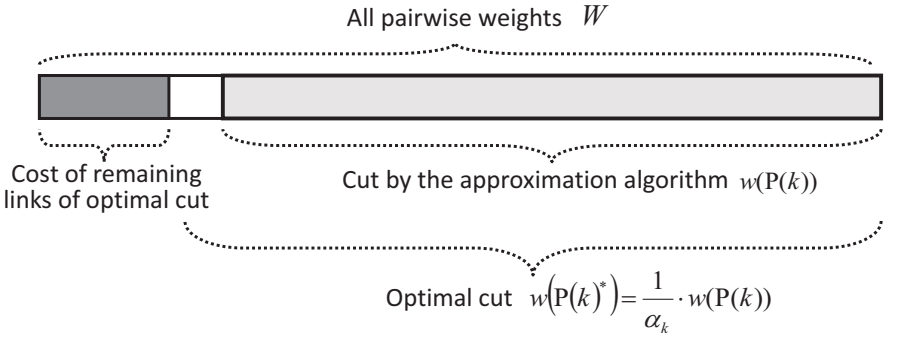


Figure 5: Derivation of the approximation ratio.

2.3. Fast approximation algorithm for Scut

We next define on-line k-means variant for the Scut method. In the algorithm, points are repeatedly re-partitioned to the cluster which provides lowest value for the Scut cost function. The partition of the points is done one-by-one, and a change of cluster will cause immediate update of the two affeceted clusters (their centroid and size). We use the fact that calculating the pairwise total squared distance within clusters is the same as calculating the Scut cost function in *TSE* form (15). We derive next a fast $O(1)$ update formula which calculates how much the value of the cost function changes when a point is moved from one cluster to another. We keep on moving points to other clusters as long as the cost function decreases, see Algorithm 2. The

Algorithm 2 Fast approximation algorithm for Scut
Input: dataset X, number of clusters k, number of points n
Output: partitioning of points P

```
Create some initial partitioning P.
changed ← TRUE
while changed do
  changed ← FALSE
  for i = 1 to n do
    for l = 1 to k do
      if ΔCost < 0 then
        move point i to cluster l
        update centroids and TSE's
        changed ← TRUE
      end if
    end for
  end for
end while
Output partitioning of points P.
```

approximation ratio is the same as in (18), where $\alpha_k > 1 - k^{-1}$. The update formula follows the merge cost in agglomerative clustering algorithm [4]. It includes the change of *TSE* when adding a point, the change of *TSE* when removing a point, and the overall cost with respect to cost function (15).

Addition:

$$\Delta TSE_{add} = \frac{n_A}{n_A + 1} \cdot ||C_A - X_i||^2 \tag{19}$$

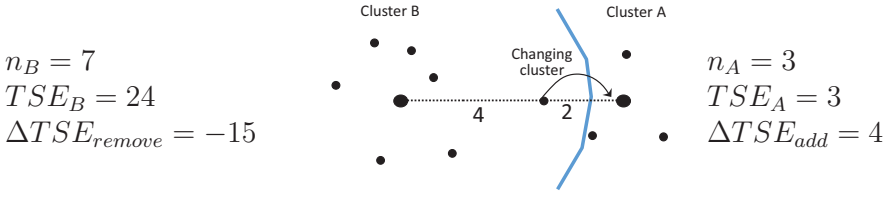


Figure 6: Changing point from cluster B to A decreasing cost by 121.02.

Removal:

$$\begin{aligned}
\Delta TSE_{remove} &= -\frac{n_B - 1}{n_B} \cdot ||\frac{n_B}{n_B - 1} \cdot C_B - \frac{1}{n_B - 1} \cdot X_i - X_i||^2 \\
&= -\frac{n_B - 1}{n_B} ||\frac{n_B}{n_B - 1} \cdot C_B - \frac{n_B}{n_B - 1} \cdot X_i||^2 \\
&= -\frac{n_B}{n_B - 1} \cdot ||C_B - X_i||^2
\end{aligned} \tag{20}$$

Total cost before the move with respect to the two clusters, is:

$$\text{Scut}_{before} = n_A \cdot TSE_A + n_B \cdot TSE_B, \tag{21}$$

where n_A and n_B are the number of points in the clusters A and B before the operation, C_A and C_B are the centroid locations before the operation and X_i

is the data point involved in the operation. Total cost after the move is:

$$\text{Scut}_{after} = (n_A + 1) \cdot (TSE_A + \Delta TSE_{add}) + (n_B - 1) \cdot (TSE_B + \Delta TSE_{remove}) \tag{22}$$

From these we get the change in cost

$$\Delta\text{Scut} = \text{Scut}_{after} - \text{Scut}_{before} \tag{23}$$

$$= TSE_A - TSE_B + (n_A + 1) \cdot \Delta TSE_{add} + (n_B - 1) \cdot \Delta TSE_{remove}, \tag{24}$$

$$= TSE_A - TSE_B + (n_A + 1) \cdot \frac{n_A}{n_A + 1} \cdot ||C_A - X_i||^2 \tag{25}$$

$$+ (n_B - 1) \cdot -\frac{n_B}{n_B - 1} \cdot ||C_B - X_i||^2. \tag{26}$$

See an example case of a point changing cluster in Figure 6, where the changes in TSE:s are: $\Delta TSE_{add} = 3/4 \cdot 2^2 = 3.00$ and $\Delta TSE_{remove} = -7/6 \cdot 4^2 = -18.67$. In Figure 6, the change in cost function would be $\Delta\text{Scut} = 3 - 24 + (3 + 1) \cdot 3 + (7 - 1) \cdot -18.67 = -121.02$.

3. Experiments

For solving the semidefinite program instances we use SeDuMi solver [38]

Algorithm 3 Balance
Input: number of points n, number of clusters k, an array of cluster sizes $cluster\_size(1..k)$.
Output: *balance*.

```
balance ← 0
for j = 1 to k do
  if cluster_size(j) > ceil(n/k) then
    balance ← balance + cluster_size(j) − ceil(n/k);
  end if
end for
balance ← 2 · balance;
output balance
```

and Yalmip modelling language [39]. We use datasets from SIPU[2]. Earth mover's distance (EMD) measures the distance between two probability distributions [40]. EMD is not usable as such in our calculations, because it requires distance between bins (or clusters). To compare how close the obtained clustering is to balance-constrained clustering (equal distribution of sizes $\lceil n/k \rceil$), we measure the balance by calculating the difference in the cluster sizes and a balanced n/k distribution, calculated by Algorithm 3. We first compare Scut with SDP algorithm against repeated k-means. The best results of 100 repeats (lowest distance) are chosen. In SDP algorithm we repeat only the point assignment phase. See an example solution in Figure 7.

[2] http://cs.uef.fi/sipu/datasets

Table 2: Balances and execution times of the proposed Scut method with the SDP algorithm and k-means clustering. 100 repeats, in SDP algorithm only the point assignment phase is repeated.

Dataset	points n	clusters k	balance		time	
			repeated Scut	repeated k-means	repeated Scut	repeated k-means
iris	150	3	2	6	8h 25min	0.50s
SUBSAMPLES:						
s1	150	15	42	30	9h 35min	0.70s
s1	50	3	2	6	34s	0.44s
s1	50	2	0	8	28s	0.34s
s2	150	15	48	24	6h 50min	0.76s
s2	50	3	2	4	27s	0.40s
s2	50	2	0	4	32s	0.38s
s3	150	15	44	28	7h 46min	0.89s
s3	50	3	2	6	31s	0.43s
s3	50	2	0	2	26s	0.41s
s4	150	15	40	30	7h 01min	0.93s
s4	50	3	0	6	28s	0.42s
s4	50	2	0	0	30s	0.36s
a1	50	20	4	4	11s	0.45s
DIM32	50	16	0	6	8s	0.46s
iris	50	3	0	10	33s	0.44s
thyroid	50	2	0	28	28s	0.38s
wine	50	3	2	6	30s	0.40s
breast	50	2	2	34	18s	0.35s
yeast_times100	50	10	8	8	10s	0.48s
glass	50	7	6	6	9s	0.44s
wdbc	50	2	0	20	11s	0.28s
best			**14 times**	**4 times**		

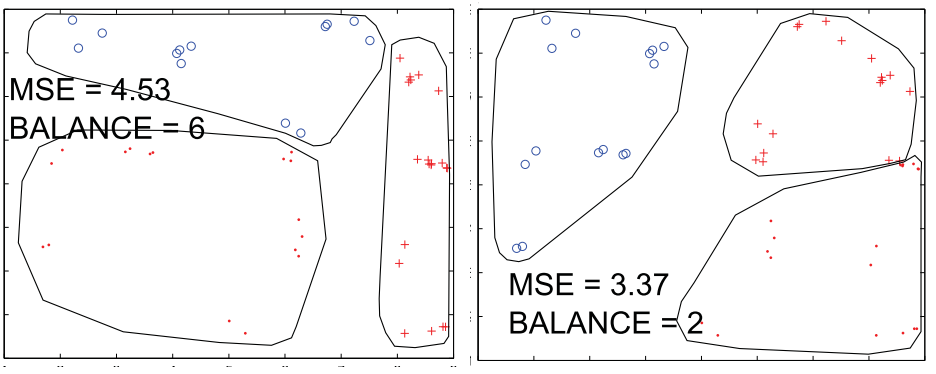


Figure 7: Example clustering results with the repeated k-means (left) and the proposed method (right) for a subset of 50 points of dataset s1.

Table 3: Best balances and total execution times of the proposed Scut with the fast approximation algorithm and k-means clustering for 100 runs.

Dataset	points n	clusters k	balance		time	
			Scut-fast	repeated k-means	Scut-fast	repeated k-means
s1	5000	15	180	184	4min	2.3s
s2	5000	15	160	172	4min	4.0s
s3	5000	15	260	338	5min	3.6s
s4	5000	15	392	458	6min	7.0s
a1	3000	20	36	40	5min	3.2s
DIM32	1024	16	0	0	42s	2.6s
iris	150	3	4	6	0.9s	0.4s
thyroid	215	2	126	168	1.0s	0.3s
wine	178	3	22	22	0.8s	0.3s
breast	699	2	216	230	1.3s	0.3s
yeast_times100	1484	10	298	362	1min 21s	4.2s
glass	214	7	110	106	4.6s	1.1s
wdbc	569	2	546	546	0.9s	0.4s

Table 4: Algorithms for joint EMD* and MSE comparison

Algorithm	Reference
Scut	Proposed
k-means	[3]
Constrained k-means	[17]
Genetic algorithm	[41]
Ncut	[28]

The results in Table 2 show that 64% of the clustering results are more balanced with the proposed method than with the repeated k-means method. They were equally balanced in 18% of the cases, and in the remaining 18% of the cases a k-means result was more balanced. Optimisation works well with small datasets (systematically better than k-means) but with bigger datasets the benefit remains smaller. The time complexity is polynomial, but the computing time increases fast when the number of points increases. With 50 points the computing time is approximately 20 s, but with 150 points it is approximately 7 hours. The memory requirement for 150 points is 4.4 GB. The results in Table 3 are for the fast approximation algorithm for which we can use bigger datasets. In 9 cases the repeated Scut gave better result than repeated k-means, in 3 cases it was equal and in 1 case it was worse.

We also conducted a joint comparison of balance and *MSE* by repeating

the algorithms Scut, k-means, constrained k-means, genetic algorithm and Ncut (Table 4). Genetic algorithm combines properties of several clusterings in one generation to make better clustering for the next generation. It is the best representative for optimising *MSE*. In Scut and repeated k-means we chose results with the best balance. In constrained k-means, cluster size parameters were set to balance=0 and *MSE* was then optimised. In Ncut we used implementation [42] by T. Cour, S. Yu and J. Shi from University of Pennsylvania. We used 100 repetitions for all algorithms and chose the best results, see Figure 8. Genetic algorithm optimises *MSE* best, but the result is less balanced. Scut always provides balance of 0 or 2 whilst constrained k-means always 0. Ncut did not perform well in this experiment. Overall, Scut performs well in both balance and *MSE*, and is a Pareto-optimal point in 3 out of 4 cases, meaning that no other algorithm provides better results both in balance and *MSE* for the same data. According to visual inspections of the 2-d datasets s1 and s4, the points are properly clustered with no overlapping between clusters, when either the proposed method or k-means in used.

We calculated all pairwise squared distances W for some datasets and the mean value for the approximated MAX k-CUT obtained by 100 runs of the fast algorithm, see Table 5. We see that in most cases the cut contains

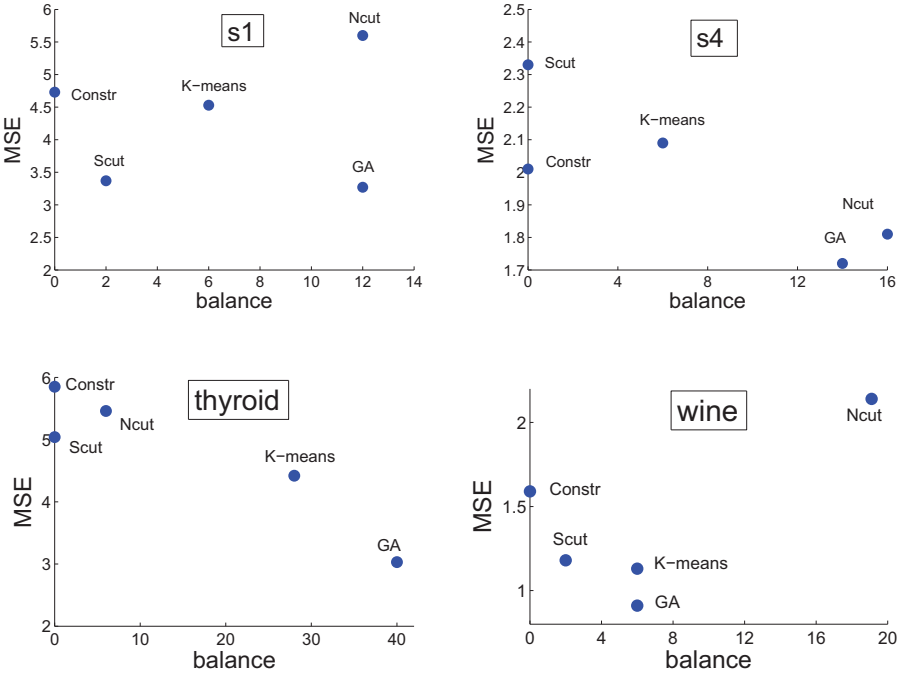


Figure 8: Joint comparison of EMD and MSE.

Table 5: All pairwise square distances W and mean value of 100 runs for the arcs of approximated MAX k-CUT. Only significant numbers are shown.

Dataset	number of clusters	W	approximated MAX k-CUT	% from W
thyroid	2	2.30	1.67	73%
breast	2	34	29	85%
wdbc	2	5.05	4.80	95%
iris	3	8.94	8.24	92%
s1	15	2.884	2.879	99.8%
DIM32	16	9.827	9.826	99.99%

over 90% of all the edge weights. This gives a high value in approximation factor ϵ, in practise $\epsilon = \infty$ for the tested sets. This means that a guaranteed approximation cannot be made for many datasets.

4. Conclusions

We have formulated all-pairwise squared distances cost function as cut-based method called Squared cut (Scut) using *MSE* and cluster sizes. We showed that this method leads to more balanced clustering. We use the solution of MAX k-CUT problem to minimise pairwise intra-cluster squared distances and Huygens' theorem to show that this corresponds to minimisation of the cost function. Since Scut is expected to be an NP-hard problem, it cannot be solved for practical-sized datasets. We applied an algorithm based on approximation of MAX k-CUT, and also introduced a fast on-line k-means

algorithm to minimise the cost function directly. We showed by experiments that the proposed approach provides better overall joint optimization of *MSE* and cluster balance than the compared methods.

[1] D. Aloise, A. Deshpande, P. Hansen, P. Popat, NP-hardness of Euclidean sum-of-squares clustering, Mach. Learn. 75 (2009) 245–248.

[2] M. Inaba, N. Katoh, H. Imai, Applications of Weighted Voronoi Diagrams and Randomization to Variance-Based k-Clustering, in Proceedings of the 10th Annual ACM symposium on computational geometry (SCG 1994) (1994) 332–339.

[3] J. MacQueen, Some methods of classification and analysis of multivariate observations., Proc. 5th Berkeley Symp. Mathemat. Statist. Probability 1 (1967) 281–296.

[4] W. H. Equitz, A New Vector Quantization Clustering Algorithm, IEEE Trans. Acoust., Speech, Signal Processing 37 (1989) 1568–1575.

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

[6] P. Fränti, O. Virmajoki, Iterative shrinking method for clustering problems, Pattern Recognition 39 (2006) 761–765.

[7] P. Fränti, J. Kivijärvi, Randomized local search algorithm for the clustering problem, Pattern Anal. Appl. 3 (2000) 358–369.

[8] D. Pelleg, A. Moore, X-means: Extending k-means with efficient estimation of the number of clusters, in: Proceedings of the Seventeenth International Conference on Machine Learning, Morgan Kaufmann, San Francisco, 2000, pp. 727–734.

[9] D. Arthur, S. Vassilvitskii, k-means++: the advantages of careful seeding, in: SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2007, pp. 1027–1035.

[10] A. Likas, N. Vlassis, J. Verbeek, The global k-means clustering algorithm, Pattern Recognition 36 (2003) 451–461.

[11] A. P. Dempster, N. M. Laird, D. B. Rubin, Maximun likelihood from incomplete data via the EM algorithm, Journal of Royal Statistical Society B 39 (1977) 1–38.

[12] Q. Zhao, V. Hautamäki, I. Kärkkäinen, P. Fränti, Random swap EM algorithm for finite mixture models in image segmentation, in: 16th IEEE International Conference on Image Processing (ICIP), 2009, pp. 2397–2400.

[13] C. H. Q. Ding, X. He, H. Zha, M. Gu, H. D. Simon, A min-max cut algorithm for graph partitioning and data clustering, in: Proceedings IEEE International Conference on Data Mining (ICDM), 2001, pp. 107–114.

[14] M. I. Malinen, P. Fränti, Clustering by analytic functions, Information Sciences 217 (2012) 31 – 38.

[15] M. I. Malinen, P. Fränti, K-means*: Clustering by gradual data transformation, Pattern Recognition 47 (2014) 3376 – 3386.

[16] R. Nallusamy, K. Duraiswamy, R. Dhanalaksmi, P. Parthiban, Optimization of non-linear multiple traveling salesman problem using k-means clustering, shrink wrap algorithm and meta-heuristics, International Journal of Nonlinear Science 9 (2010) 171 – 177.

[17] P. S. Bradley, K. P. Bennett, A. Demiriz, Constrained K-Means Clustering, Technical Report, MSR-TR-2000-65, Microsoft Research, 2000.

[18] M. I. Malinen, P. Fränti, Balanced k-means for clustering, in: Joint Int. Workshop on Structural, Syntactic, and Statistical Pattern Recognition (S+SSPR 2014), LNCS 8621, Joensuu, Finland, 2014.

[19] S. Zhu, D. Wang, T. Li, Data clustering with size constraints, Knowledge-Based Systems 23 (2010) 883–889.

[20] A. Banerjee, J. Ghosh, Frequency sensitive competitive learning for balanced clustering on high-dimensional hyperspheres, IEEE Transactions on Neural Networks 15 (2004) 719.

[21] C. T. Althoff, A. Ulges, A. Dengel, Balanced clustering for content-based image browsing, in: GI-Informatiktage 2011, Gesellschaft fr Informatik e.V., 2011.

[22] A. Banerjee, J. Ghosh, On scaling up balanced clustering algorithms, in: In Proceedings of the SIAM International Conference on Data Mining, 2002, pp. 333–349.

[23] Y. Chen, Y. Zhang, X. Ji, Size regularized cut for data clustering, in: Advances in Neural Information Processing Systems, 2005, 2005.

[24] Y. Kawahara, K. Nagano, Y. Okamoto, Submodular fractional program-

ming for balanced clustering, Pattern Recognition Letters 32 (2011) 235–243.

[25] Y. Liao, H. Qi, W. Li, Load-Balanced Clustering Algorithm With Distributed Self-Organization for Wireless Sensor Networks, Sensors Journal, IEEE 13 (2013) 1498–1506.

[26] L. Yao, X. Cui, M. Wang, An energy-balanced clustering routing algorithm for wireless sensor networks, in: Computer Science and Information Engineering, 2009 WRI World Congress on, IEEE, volume 3, 2006.

[27] L. Hagen, A. B. Kahng, New spectral methods for ratio cut partitioning and clustering, IEEE Transactions on Computer-Aided Design 11 (1992) 1074–1085.

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

[29] U. von Luxburg, A tutorial on spectral clustering, Statistics and Computing 17 (2007) 395–416.

[30] T. D. Bie, N. Cristianini, Fast sdp relaxations of graph cut clustering,

transduction, and other combinatorial problems, J. Mach. Learn. Res. 7 (2006) 1409–1436.

[31] A. Frieze, M. Jerrum, Improved approximation algorithms for max-k-cut and max bisection, Algorithmica 18 (1997) 67–81.

[32] W. Zhu, C. Guo, A local search approximation algorithm for max-k-cut of graph and hypergraph, in: Fourth International Symposium on Parallel Architectures, Algorithms and Programming, 2011, pp. 236–240.

[33] L. J. Schulman, Clustering for edge-cost minimization, in: Proc. of the 32nd Ann. ACM Symp. on Theory of Computing (STOC), 2000, pp. 547–555.

[34] S. Sahni, T. Gonzalez, P-complete approximation problems, J. ACM 23 (1976) 555–565.

[35] W. F. de la Vega, M. Karpinski, C. Kenyon, Y. Rabani, Approximation schemes for clustering problems., in: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing (STOC '03), ACM, New York, NY, USA, 2003, pp. 50–58.

[36] N. Guttmann-Beck, R. Hassin, Approximation algorithms for min-sum p-clustering, Discrete Applied Mathematics 89 (1998) 125–142.

[37] H. Späth, Cluster analysis algorithms for data reduction and classification of objects, Wiley, New York, 1980.

[38] J. F. Sturm, O. Romanko, I. Polik, T. Terlaky, Sedumi, 2009. http://mloss.org/software/view/202/.

[39] J. Löfberg, Yalmip : A toolbox for modeling and optimization in MATLAB, in: Proceedings of the CACSD Conference, Taipei, Taiwan, 2004. URL: http://users.isy.liu.se/johanl/yalmip.

[40] E. Levina, P. Bickel, The earthmovers distance is the mallows distance: Some insights from statistics, in: Proceedings of ICCV 2001, Vancouver, Canada, 2001.

[41] P. Fränti, J. Kivijärvi, T. Kaukoranta, O. Nevalainen, Genetic algorithms for large scale clustering problem, Comput. J. 40 (1997) 547 – 554.

[42] T. Cour, S. Yu, J. Shi, Ncut implementation, 2004. URL: http://www.cis.upenn.edu/∼jshi/software/.

Paper **IV**

M. I. Malinen and P. Fränti
"Balanced *k*-means for clustering"
Joint Int. Workshop on Structural, Syntactic, and Statistical Pattern Recognition (S+SSPR 2014),
LNCS 8621, 32–41,
Joensuu, Finland,
20–22 August, 2014.

Balanced K-Means for Clustering

Mikko I. Malinen and Pasi Fränti

School of Computing, University of Eastern Finland,
Box 111, FIN-80101 Joensuu, Finland
{mmali,franti}@cs.uef.fi
http://cs.uef.fi/~mmali, http://cs.uef.fi/pages/franti

Abstract. We present a k-means-based clustering algorithm, which optimizes mean square error, for given cluster sizes. A straightforward application is balanced clustering, where the sizes of each cluster are equal. In k-means assignment phase, the algorithm solves the assignment problem by Hungarian algorithm. This is a novel approach, and makes the assignment phase time complexity $O(n^3)$, which is faster than the previous $O(k^{3.5}n^{3.5})$ time linear programming used in constrained k-means. This enables clustering of bigger datasets of size over 5000 points.

Keywords: clustering, balanced clustering, assignment problem, Hungarian algorithm.

1 Introduction

Euclidean sum-of-squares clustering is an NP-hard problem [1], which groups n data points into k clusters so that intra-cluster distances are low and inter-cluster distances are high. Each group is represented by a center point (centroid). The most common criterion to optimize is the mean square error (MSE):

$$\text{MSE} = \sum_{j=1}^{k} \sum_{X_i \in C_j} \frac{|| X_i - C_j ||^2}{n}, \tag{1}$$

where X_i denotes data point locations and C_j denotes centroid locations. K-means [19] is the most commonly used clustering algorithm, which provides a local minimum of MSE given the number of clusters as input. K-means algorithm consists of two repeatedly executed steps:

Assignment Step: Assign the data points to clusters specified by the nearest centroid:

$$P_j^{(t)} = \{X_i : \|X_i - C_j^{(t)}\| \leq \|X_i - C_{j^*}^{(t)}\| \quad \forall \quad j^* = 1, ..., k\}$$

Update Step: Calculate the mean of each cluster:

$$C_j^{(t+1)} = \frac{1}{|P_j^{(t)}|} \sum_{X_i \in P_j^{(t)}} X_i$$

P. Fränti et al. (Eds.): S+SSPR 2014, LNCS 8621, pp. 32–41, 2014.

These steps are repeated until centroid locations do not change anymore. K-means assignment step and update step are optimal with respect to MSE: The partitioning step minimizes MSE for a given set of centroids; the update step minimizes MSE for a given partitioning. The solution therefore converges to a local optimum but without guarantee of global optimality. To get better results than in k-means, slower agglomerative algorithms [10,13,12] or more complex k-means variants [3,11,21,18] are sometimes used.

In *balanced clustering* there are an equal number of points in each cluster. Balanced clustering is desirable for example in divide-and-conquer methods where the divide step is done by clustering. Examples can be found in circuit design [14] and in photo query systems [2], where the photos are clustered according to their content. Applications can also be used in workload balancing algorithms. For example, in [20] multiple traveling salesman problem clusters the cities, so that each salesman operates in one cluster. It is desirable that each salesman has equal workload. Networking utilizes balanced clustering to obtain some desirable goals [17,23].

We next review existing balanced clustering algorithms. In *frequency sensitive competitive learning* (FSCL) the centroids compete of points [5]. It multiplicatively increases the distance of the centroids to the data point by the times the centroid has already won points. Bigger clusters are therefore less likely to win more points. The method in [2] uses FSCL, but with additive bias instead of multiplicative bias. The method in [4] uses a fast ($O(kNlogN)$) algorithm for balanced clustering based on three steps: sample the given data, cluster the sampled data and populate the clusters with the data points that were not sampled. The article [6] and book chapter [9] present a *constrained k-means algorithm*, which is like k-means, but the assignment step is implemented as a linear program, in which the minimum number of points τ_h of clusters can be set as parameters. The constrained k-means clustering algorithm works as follows:

Given m points in $\mathbb{R}^n$, minimum cluster membership values $\tau_h \geq 0, h = 1, ..., k$ and cluster centers $C_1^{(t)}, C_2^{(t)}, ..., C_k^{(t)}$ at iteration t, compute $C_1^{(t+1)}, C_2^{(t+1)}$, $..., C_k^{(t+1)}$ at iteration $t+1$ using the following 2 steps:

Cluster Assignment. Let $T_{i,h}^t$ be a solution to the following linear program with $C_h^{(t)}$ fixed:

$$\text{minimize}_T \sum_{i=1}^{m} \sum_{h=1}^{k} T_{i,h} \cdot (\frac{1}{2}||X_i - C_h^{(t)}||_2^2) \tag{2}$$

$$\text{subject to} \sum_{i=1}^{m} T_{i,h} \geq \tau_h, h = 1, ..., k \tag{3}$$

$$\sum_{h=1}^{k} T_{i,h} = 1, i = 1, ..., m \tag{4}$$

$$T_{i,h} \geq 0, i = 1, ..., m, h = 1, ..., k. \tag{5}$$

Cluster Update. Update $C_h^{(t+1)}$ as follows:

$$C_h^{(t+1)} = \begin{cases} \frac{\sum_{i=1}^{m} T_{i,h}^{(t)} X_i}{\sum_{i=1}^{m} T_{i,h}^{(t)}} & \text{if} \quad \sum_{i=1}^{m} T_{i,h}^{(t)} > 0, \\ C_h^{(t)} & \text{otherwise.} \end{cases}$$

These steps are repeated until $C_h^{(t+1)} = C_h^{(t)}, \quad \forall h = 1, ..., k$.

A cut-based method *Ratio cut* [14] includes cluster sizes in its cost function

$$\text{RatioCut}(P_1, ..., P_k) = \sum_{i=1}^{k} \frac{\text{cut}(P_i, \bar{P}_i)}{|P_i|}.$$

Here P_i:s are the partitions. *Size regularized cut* SRCut [8] is defined as the sum of the inter-cluster similarity and a regularization term measuring the relative size of two clusters. In [16] there is a balancing aiming term in cost function and [24] tries to find a partition close to the given partition, but so that cluster size constraints are fulfilled. There are also application-based solutions in networking [17], which aim at network load balancing, where clustering is done by self-organization without central control. In [23], energy-balanced routing between sensors is aimed so that most suitable balanced amount of nodes will be the members of the clusters.

Balanced clustering, in general, is a 2-objective optimization problem, in which two aims contradict each other: to minimize MSE and to balance cluster sizes. Traditional clustering aims at minimizing MSE without considering cluster size balance. Balancing, on the other hand, would be trivial if we did not care about MSE; simply by dividing points to equal size clusters randomly. For optimizing both, there are two alternative approaches: *Balance-constrained* and *balance-driven* clustering.

In balance-constrained clustering, cluster size balance is a mandatory requirement that must be met, and minimizing MSE is a secondary criterion. In balance-driven clustering, balance is an aim but not mandatory. It is a compromize between these two goals, namely the balance and the MSE. The solution can be a weighted compromize between MSE and the balance, or a heuristic that aims at minimizing MSE but indirectly creates a more balanced result than standard k-means. Existing algorithms are grouped into these two classes in Table 1.

In this paper, we formulate balanced k-means, so that it belongs to the first category. It is otherwise the same as standard k-means but it guarantees balanced cluster sizes. It is also a special case of constrained k-means, where cluster sizes are set equal. However, instead of using linear programming in the assignment phase, we formulate the partitioning as a pairing problem [7], which can be solved optimally by Hungarian algorithm in $O(n^3)$ time.

Table 1. Classification of some balanced clustering algorithms

Balance-constrained
Balanced k-means (proposed)
Constrained k-means [6]
Size constrained [24]
Balance-driven
FSCL [5]
FSCL with additive bias [2]
Cluster sampled data [4]
Ratio cut [14]
SRcut [8]
Submodular fractional programming [16]

2 Balanced *k*-Means

To describe balanced k-means, we need to define what is an assignment problem. The formal definition of assignment problem (or linear assignment problem) is as follows. Given two sets (A and S), of equal size, and a weight function $W : A \times S \to \mathbb{R}$. The goal is to find a bijection $f : A \to S$ so that the cost function is minimized:

$$\text{Cost} = \sum_{a \in A} W(a, f(a)).$$

In the context of the proposed algorithm, sets A and S correspond respectively to cluster slots and to data points, see Figure 1.

In balanced k-means, we proceed as in k-means, but the assignment phase is different: Instead of selecting the nearest centroids we have n pre-allocated slots (n/k slots per cluster), and datapoints can be assigned only to these slots, see Figure 1. This will force all clusters to be of same size assuming that $\lceil n/k \rceil = \lfloor n/k \rfloor = n/k$. Otherwise there will be ($n \mod k$) clusters of size $\lceil n/k \rceil$, and $k - (n \mod k)$ clusters of size $\lfloor n/k \rfloor$.

To find assignment that minimizes MSE, we solve an assignment problem using Hungarian algorithm [7]. First we construct a bipartite graph consisting n datapoints and n cluster slots, see Figure 2. We then partition the cluster slots in clusters of as even number of slots as possible.

We give centroid locations to partitioned cluster slots, one centroid to each cluster. The initial centroid locations can be drawn randomly from all data points. The edge weight is the squared distance from the point to the cluster centroid it is assigned to. Contrary to standard assignment problem with fixed weights, here the weights dynamically change after each k-means iteration according to the newly calculated centroids. After this, we perform the Hungarian algorithm to get the minimal weight pairing. The squared distances are stored in a $n \times n$ matrix, for the sake of the Hungarian algorithm. The update step is

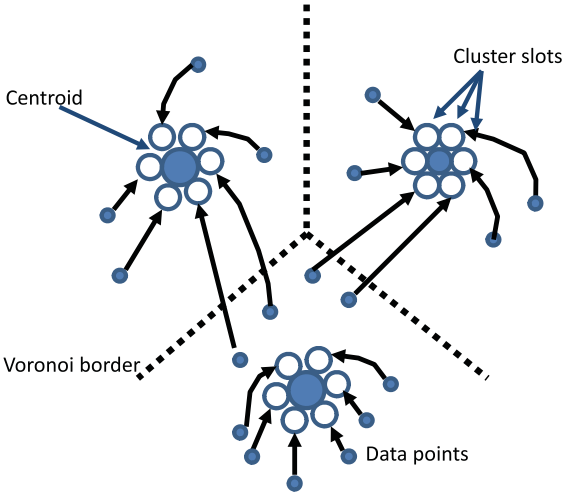


Fig. 1. Assigning points to centroids via cluster slots

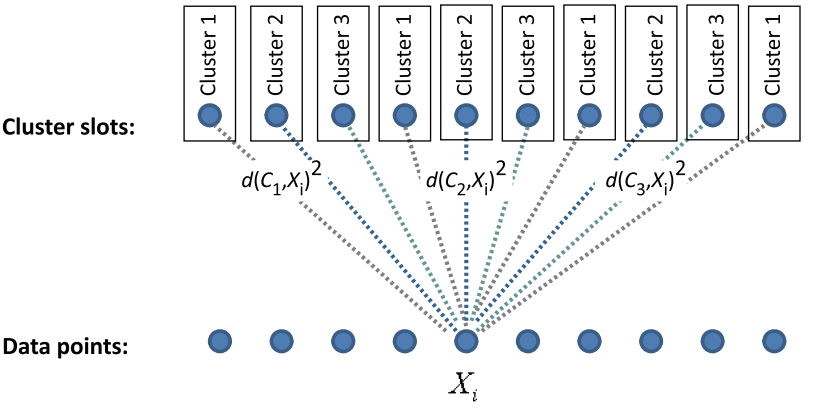


Fig. 2. Minimum MSE calculation with balanced clusters. Modeling with bipartite graph.

similar to that of k-means, where the new centroids are calculated as the means of the data points assigned to each cluster:

$$C_i^{(t+1)} = \frac{1}{n_i} \cdot \sum_{X_j \in C_i^{(t)}} X_j. \tag{6}$$

The weights of the edges are updated immediately after the update step. The pseudocode of the algorithm is in Algorithm 1. In calculation of edge weights, the number of cluster slot is denoted by a and mod is used in calculation of cluster where a cluster slot belongs to. The edge weights are calculated by

$$W(a,i) = dist(X_i, C^t_{(a \bmod k)+1})^2 \quad \forall a \in [1,n] \quad \forall i \in [1,n]. \tag{7}$$

Algorithm 1. Balanced k-means
Input: dataset X, number of clusters k
Output: partitioning of dataset.

Initialize centroid locations C^0.
$t \leftarrow 0$
repeat
  Assignment step:
    Calculate edge weights.
    Solve an Assignment problem.
  Update step:
    Calculate new centroid locations C^{t+1}
  $t \leftarrow t + 1$
until centroid locations do not change.
Output partitioning.

After convergence of the algorithm the partition of points X_i, $i \in [1, n]$, is

$$X_{f(a)} \in P_{(a \bmod k)+1}. \tag{8}$$

There is a convergence result in [6] (Proposition 2.3) for constrained k-means. The result says that the algorithm terminates in a finite number of iterations at a partitioning that is locally optimal. At each iteration, the cluster assignment step cannot increase the objective function of constrained k-means (3) in [6]. The cluster update step will either strictly decrease the value of the objective function or the algorithm will terminate. Since there are a finite number of ways to assign m points to k clusters so that cluster h has at least τ_h points, since constrained k-means algorithm does not permit repeated assignments, and since the objective of constrained k-means (3) in [6] is strictly nonincreasing and bounded below by zero, the algorithm must terminate at some cluster assignment that is locally optimal. The same convergence result applies to balanced k-means as well. The assignment step is optimal with respect to MSE because of pairing and the update step is optimal, because MSE is clusterwise minimized as is in k-means.

3 Time Complexity

Time complexity of the assignment step in k-means is $O(k \cdot n)$. Constrained k-means involves linear programming. It takes $O(v^{3.5})$ time, where v is the number of variables, by Karmarkars projective algorithm [15,22], which is the fastest interior point algorithm known to the authors. Since $v = k \cdot n$, the time complexity is $O(k^{3.5}n^{3.5})$. The assignment step of the proposed balanced k-means algorithm can be solved in $O(n^3)$ time with the Hungarian algorithm. This makes it much faster than in the constrained k-means, and allows therefore significantly bigger datasets to be clustered.

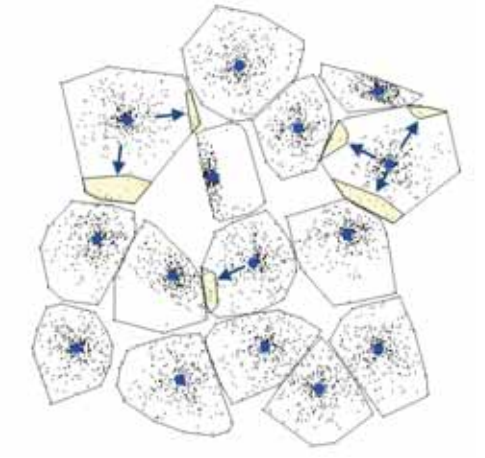


Fig. 3. Sample clustering result. Most significant differences between balanced clustering and standard k-means (non-balanced) clustering are marked and pointed out by arrows.

Table 2. MSE, standard deviation of MSE and time/run of 100 runs

Dataset	Size	Clusters	Algorithm	Best	Mean	St.dev.	Time
s2	5000	15	Balanced k-means	**2.86**	(one run)	(one run)	**1h 40min**
			Constrained k-means	-	-	-	-
s1	1000	15	Balanced k-means	2.89	(one run)	(one run)	**47s**
subset			Constrained k-means	**2.61**	(one run)	(one run)	26min
s1	500	15	Balanced k-means	3.48	3.73	0.21	**8s**
subset			Constrained k-means	**3.34**	**3.36**	0.16	30s
			K-means	2.54	4.21	1.19	0.01s
s1	500	7	Balanced k-means	14.2	15.7	1.7	10s
subset			Constrained k-means	**14.1**	**15.6**	1.6	**8s**
s2	500	15	Balanced k-means	3.60	3.77	0.12	**8s**
subset			Constrained k-means	**3.42**	**3.43**	0.08	29s
s3	500	15	Balanced k-means	3.60	3.69	0.17	**9s**
subset			Constrained k-means	**3.55**	**3.57**	0.12	35s
s4	500	15	Balanced k-means	3.46	3.61	1.68	**12s**
subset			Constrained k-means	**3.42**	**3.53**	0.20	45s
thyroid	215	2	Balanced k-means	**4.00**	**4.00**	0.001	2.5s
			Constrained k-means	**4.00**	**4.00**	0.001	**0.25s**
wine	178	3	Balanced k-means	**3.31**	3.33	0.031	0.36s
			Constrained k-means	**3.31**	**3.31**	0.000	**0.12s**
iris	150	3	Balanced k-means	**9.35**	3.39	0.43	0.34s
			Constrained k-means	**9.35**	**3.35**	0.001	**0.14s**

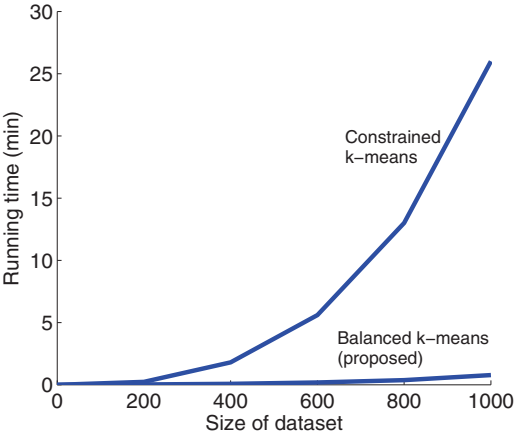


Fig. 4. Running time with different-sized subsets of s1 dataset

4 Experiments

In the experiments we use artificial datasets s1-s4, which have Gaussian clusters with increasing overlap and real-world datasets thyroid, wine and iris. The source of the datasets is `http://cs.uef.fi/sipu/datasets/`. As a platform, Intel Core i5-3470 3.20GHz processor was used. We have been able to cluster datasets of size 5000 points. One example partitioning can be seen in Figure 3, for which the running time was 1h40min. Comparison of MSE values of constrained k-means and balanced k-means is shown in Table 2, running times in Figure 4. The results indicate that constrained k-means gives slightly better MSE in many cases, but that balanced k-means is significantly faster when the size of dataset increases. For dataset of size 5000 constrained k-means could no longer provide result within one day. The difference in MSE is most likely due to the fact that balanced k-means strictly forces balance within ± 1 points, but constrained k-means does not. It may happen, that constrained k-means has many clusters of size $\lfloor n/k \rfloor$, but some smaller amount of clusters of size bigger than $\lceil n/k \rceil$.

5 Conclusions

We have presented balanced k-means clustering algorithm which guarantees equal-sized clusters. The algorithm is a special case of constrained k-means, where cluster sizes are equal, but much faster. The experimental results show that the balanced k-means gives slightly higher MSE-values to that of the constrained k-means, but about 3 times faster already for small datasets. Balanced k-means is able to cluster bigger datasets than constrained k-means. However, even the proposed method may still be too slow for practical application and therefore, our future work will focus on finding some faster sub-optimal algorithm for the assignment step.

References

1. Aloise, D., Deshpande, A., Hansen, P., Popat, P.: NP-hardness of Euclidean sum-of-squares clustering. Mach. Learn. 75, 245–248 (2009)
2. Althoff, C.T., Ulges, A., Dengel, A.: Balanced clustering for content-based image browsing. In: GI-Informatiktage 2011. Gesellschaft für Informatik e.V. (March 2011)
3. Arthur, D., Vassilvitskii, S.: k-means++: the advantages of careful seeding. In: SODA 2007: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1027–1035. Society for Industrial and Applied Mathematics, Philadelphia (2007)
4. Banerjee, A., Ghosh, J.: On scaling up balanced clustering algorithms. In: Proceedings of the SIAM International Conference on Data Mining, pp. 333–349 (2002)
5. Banerjee, A., Ghosh, J.: Frequency sensitive competitive learning for balanced clustering on high-dimensional hyperspheres. IEEE Transactions on Neural Networks 15, 719 (2004)
6. Bradley, P.S., Bennett, K.P., Demiriz, A.: Constrained k-means clustering. Tech. rep., MSR-TR-2000-65, Microsoft Research (2000)
7. Burkhard, R., Dell'Amico, M., Martello, S.: Assignment Problems (Revised reprint). SIAM (2012)
8. Chen, Y., Zhang, Y., Ji, X.: Size regularized cut for data clustering. In: Advances in Neural Information Processing Systems (2005)
9. Demiriz, A., Bennett, K.P., Bradley, P.S.: Using assignment constraints to avoid empty clusters in k-means clustering. In: Basu, S., Davidson, I., Wagstaff, K. (eds.) Constrained Clustering: Advances in Algorithms, Theory, and Applications. Chapman & Hall/CRC Data Mining and Knowledge Discovery Series (2008)
10. Equitz, W.H.: A New Vector Quantization Clustering Algorithm. IEEE Trans. Acoust., Speech, Signal Processing 37, 1568–1575 (1989)
11. Fränti, P., Kivijärvi, J.: Randomized local search algorithm for the clustering problem. Pattern Anal. Appl. 3(4), 358–369 (2000)
12. Fränti, P., Virmajoki, O.: Iterative shrinking method for clustering problems. Pattern Recognition 39(5), 761–765 (2006)
13. Fränti, P., Virmajoki, O., Hautamäki, V.: Fast agglomerative clustering using a k-nearest neighbor graph. IEEE Trans. on Pattern Analysis and Machine Intelligence 28(11), 1875–1881 (2006)
14. Hagen, L., Kahng, A.B.: New spectral methods for ratio cut partitioning and clustering. IEEE Transactions on Computer-Aided Design 11(9), 1074–1085 (1992)
15. Karmarkar, N.: A new polynomial time algorithm for linear programming. Combinatorica 4(4), 373–395 (1984)
16. Kawahara, Y., Nagano, K., Okamoto, Y.: Submodular fractional programming for balanced clustering. Pattern Recognition Letters 32(2), 235–243 (2011)
17. Liao, Y., Qi, H., Li, W.: Load-Balanced Clustering Algorithm With Distributed Self-Organization for Wireless Sensor Networks. IEEE Sensors Journal 13(5), 1498–1506 (2013)
18. Likas, A., Vlassis, N., Verbeek, J.: The global k-means clustering algorithm. Pattern Recognition 36, 451–461 (2003)
19. MacQueen, J.: Some methods of classification and analysis of multivariate observations. In: Proc. 5th Berkeley Symp. Mathemat. Statist. Probability, vol. 1, pp. 281–296 (1967)

20. Nallusamy, R., Duraiswamy, K., Dhanalaksmi, R., Parthiban, P.: Optimization of non-linear multiple traveling salesman problem using k-means clustering, shrink wrap algorithm and meta-heuristics. International Journal of Nonlinear Science 9(2), 171–177 (2010)
21. Pelleg, D., Moore, A.: X-means: Extending k-means with efficient estimation of the number of clusters. In: Proceedings of the Seventeenth International Conference on Machine Learning, pp. 727–734. Morgan Kaufmann, San Francisco (2000)
22. Strang, G.: Karmarkars algorithm and its place in applied mathematics. The Mathematical Intelligencer 9(2), 4–10 (1987)
23. Yao, L., Cui, X., Wang, M.: An energy-balanced clustering routing algorithm for wireless sensor networks. In: 2009 WRI World Congress on Computer Science and Information Engineering, vol. 3. IEEE (2006)
24. Zhu, S., Wang, D., Li, T.: Data clustering with size constraints. Knowledge-Based Systems 23(8), 883–889 (2010)

Paper **V**

C. Zhong, M. I. Malinen, D. Miao and P. Fränti

"A fast minimum spanning tree algorithm based on *K*-means"

Information Sciences,

295, pp. 1–17, 2015.

Information Sciences 295 (2015) 1–17

Contents lists available at ScienceDirect

Information Sciences

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

A fast minimum spanning tree algorithm based on *K*-means

Caiming Zhong [a,*], Mikko Malinen [b], Duoqian Miao [c], Pasi Fränti [b]

[a] College of Science and Technology, Ningbo University, Ningbo 315211, PR China
[b] School of Computing, University of Eastern Finland, P.O. Box 111, FIN-80101 Joensuu, Finland
[c] Department of Computer Science and Technology, Tongji University, Shanghai 201804, PR China

ARTICLE INFO

Article history:
Received 14 June 2014
Received in revised form 25 September 2014
Accepted 3 October 2014
Available online 14 October 2014

Keywords:
Minimum spanning tree
Clustering
Manifold learning
K-means

ABSTRACT

Minimum spanning trees (MSTs) have long been used in data mining, pattern recognition and machine learning. However, it is difficult to apply traditional MST algorithms to a large dataset since the time complexity of the algorithms is quadratic. In this paper, we present a fast MST (FMST) algorithm on the complete graph of N points. The proposed algorithm employs a divide-and-conquer scheme to produce an approximate MST with theoretical time complexity of $O(N^{1.5})$, which is faster than the conventional MST algorithms with $O(N^2)$. It consists of two stages. In the first stage, called the divide-and-conquer stage, *K*-means is employed to partition a dataset into $\sqrt{N}$ clusters. Then an exact MST algorithm is applied to each cluster and the produced $\sqrt{N}$ MSTs are connected in terms of a proposed criterion to form an approximate MST. In the second stage, called the refinement stage, the clusters produced in the first stage form $\sqrt{N}-1$ neighboring pairs, and the dataset is repartitioned into $\sqrt{N}-1$ clusters with the purpose of partitioning the neighboring boundaries of a neighboring pair into a cluster. With the $\sqrt{N}-1$ clusters, another approximate MST is constructed. Finally, the two approximate MSTs are combined into a graph and a more accurate MST is generated from it. The proposed algorithm can be regarded as a framework, since any exact MST algorithm can be incorporated into the framework to reduce its running time. Experimental results show that the proposed approximate MST algorithm is computationally efficient, and the approximation is close to the exact MST so that in practical applications the performance does not suffer.

1. Introduction

A minimum spanning tree (MST) is a spanning tree of an undirected and weighted graph such that the sum of the weights is minimized. As it can roughly estimate the intrinsic structure of a dataset, MST has been broadly applied in image segmentation [2,47], cluster analysis [46,51–53], classification [27], manifold learning [48,49], density estimation [30], diversity estimation [33], and some applications of the variant problems of MST [10,36,43]. Since the pioneering algorithm of computing an MST was proposed by Otakar Borůvka in 1926 [6], the studies of the problem have focused on finding the optimal exact MST algorithm, fast and approximate MST algorithms, distributed MST algorithms and parallel MST algorithms.

The studies on constructing an exact MST start with Borůvka's algorithm [6]. This algorithm begins with each vertex of a graph being a tree. Then for each tree it iteratively selects the shortest edge connecting the tree to the rest, and combines the edge into the forest formed by all the trees, until the forest is connected. The computational complexity of this algorithm is

* Corresponding author. Tel.: +86 21 69589867.
E-mail address: zhongcaiming@nbu.edu.cn (C. Zhong).

http://dx.doi.org/10.1016/j.ins.2014.10.012
0020-0255/

$O(E \log V)$, where E is the number of edges, and V is the number of vertices in the graph. Similar algorithms have been invented by Choquet [13], Florek et al. [19] and Sollin [42], respectively.

One of the most typical examples is Prim's algorithm, which was proposed by Jarník [26], Prim [39] and Dijkstra [15]. It first arbitrarily selects a vertex as a tree, and then repeatedly adds the shortest edge that connects a new vertex to the tree, until all the vertices are included. The time complexity of Prim's algorithm is $O(E \log V)$. If Fibonacci heap is employed to implement a min-priority queue to find the shortest edge, the computational time is reduced to $O(E + V \log V)$ [14].

Kruskal's algorithm is another widely used exact MST algorithm [32]. In this algorithm, all the edges are sorted by their weights in non-decreasing order. It starts with each vertex being a tree, and iteratively combines the trees by adding edges in the sorted order excluding those leading to a cycle, until all the trees are combined into one tree. The running time of Kruskal's algorithm is $O(E \log V)$.

Several fast MST algorithms have been proposed. For a sparse graph, Yao [50], and Cheriton and Tarjan [11] proposed algorithms with $O(E \log \log V)$ time. Fredman and Tarjan [20] proposed the Fibonacci heap as a data structure of implementing the priority queue for constructing an exact MST. With the heaps, the computational complexity is reduced to $O(E\beta(E, V))$, where $\beta(E, V) = \min\{i | \log^{(i)} V \leqslant E/V\}$. Gabow et al. [21] incorporated the idea of *Packets* [22] into the Fibonacci heap, and reduced the complexity to $O(E \log \beta(E, V))$.

Recent progress on the exact MST algorithm was made by Chazelle [9]. He discovered a new heap structure, called soft heap, to implement the priority queue, and as a result, the time complexity is reduced to $O(E\alpha(E, V))$, where α is the inverse of the Ackermann function. March et al. [35] proposed a dual-tree on a kd-tree and a dual-tree on a cover-tree for constructing MST, with claimed time complexity as $O(N \log N\alpha(N)) \approx O(N \log N)$.

Distributed MST and parallel MST algorithms have also been studied in the literature. The first algorithm of the distributed MST problem was presented by Gallager et al. [23]. The algorithm supposes that a processor exits at each vertex and knows initially only the weights of the adjacent edges. It runs in $O(V \log V)$ time. Several faster $O(V)$ time distributed MST algorithms have been proposed by Awerbuch [3] and Abdel-Wahab et al. [1], respectively. Peleg and Rubinovich [37] presented a lower bound of time complexity $O(D + \sqrt{V}/\log V)$ for constructing a distributed MST on a network, where $D = \Omega(\log V)$ is the diameter of the network. Moreover, Khan and Pandurangan [29] proposed a distributed approximate MST algorithm on networks and its complexity is $\widetilde{O}(D + L)$, where L is the local shortest path diameter.

Chong et al. [12] presented a parallel algorithm to construct an MST in $O(\log V)$ time by employing a linear number of processors. Pettie and Ramachandran [38] proposed a randomized parallel algorithm to compute a minimum spanning forest, which also runs in logarithmic time. Bader and Cong [4] presented four parallel algorithms, of which three algorithms are variants of Borůvka's. For different graphs, their algorithms can find MSTs four to six times faster using eight processors than the sequential algorithms.

Several approximate MST algorithms have been proposed. The algorithms in [7,44] are composed of two steps. In the first step, a sparse graph is extracted from the complete graph, and then in the second step, an exact MST algorithm is applied to the extracted graph. In these algorithms, different methods for extracting sparse graphs have been employed. For example, Vaidya [44] used a group of grids to partition a dataset into cubical boxes of identical size. For each box, a representative point was determined. Any two representatives of two cubical boxes were connected if the corresponding edge length was between two given thresholds. Within a cubical box, points were connected to the representative. Callahan and Kosaraju [7] applied a well-separated pair decomposition of the dataset to extract a sparse graph.

Recent studies that focused on finding an approximate MST and applying it to clustering can be found in [34,45]. Wang et al. [45] employed a divide-and-conquer scheme to construct an approximate MST. However, their goal was not to find the MST but merely to detect the long edges of the MST at an early stage for clustering. An initial spanning tree is constructed by randomly storing the dataset in a list, in which each data point is connected to its predecessor (or successor). At the same time, the weight of each edge from a data point to its predecessor (or successor) are assigned. To optimize the spanning tree, the dataset is divided into multiple subsets with a divisive hierarchical clustering algorithm (DHCA), and the nearest neighbor of a data point within a subset is found by a brute force search. Accordingly, the spanning tree is updated. The algorithm is performed repeatedly and the spanning tree is optimized further after each run.

Lai et al. [34] proposed an approximate MST algorithm based on Hilbert curve for clustering. It consists of two phases. The first phase is to construct an approximate MST with the Hilbert curve, and the second phase is to partition the dataset into subsets by measuring the densities of the points along the approximate MST with a specified density threshold. The process of constructing an approximate MST is iterative and the number of iterations is $(d + 1)$, where d is the number of dimensions of the dataset. In each iteration, an approximate MST is generated similarly as in Prim's algorithm. The main difference is that Lai's method maintains a min-priority queue by considering the approximate MST produced in the last iteration and the neighbors of the visited points determined by a Hilbert sorted linear list, while Prim's algorithm considers all the neighbors of a visited point. However, the accuracy of Lai's method depends on the order of the Hilbert curve and the number of neighbors of a visited point in the linear list.

In this paper, we propose an approximate and fast MST (FMST) algorithm based on the divide-and-conquer technique, of which the preliminary version of the idea was presented in a conference paper [54]. It consists of two stages: divide-and-conquer and refinement. In the divide-and-conquer stage, the dataset is partitioned by K-means into $\sqrt{N}$ clusters, and the exact MSTs of all the clusters are constructed and merged. In the refinement stage, boundaries of the clusters are considered. It runs in $O(N^{1.5})$ time when Prim's or Kruskal's algorithm is used in its divide-and-conquer stage, and in practical use does not reduce the quality compared to an exact MST.

The rest of this paper is organized as follows. In Section 2, the fast divide-and-conquer MST algorithm is presented. The time complexity of the proposed method is analyzed in Section 3, and experiments on the efficiency and accuracy of the proposed algorithm are given in Section 4. Finally, we conclude this work in Section 5.

2. Proposed method

2.1. Overview of the proposed method

The efficiency of constructing an MST or a K nearest neighbor graph (KNNG) is determined by the number of comparisons of the distances between two data points. In the methods like brute force for KNNG and Kruskal's for MST, many unnecessary comparisons exist. For example, to find the K nearest neighbor of a point, it is not necessary to search the entire dataset but a small local portion; to construct an MST with Kruskal's algorithm in a complete graph, it is not necessary to sort all $N(N-1)/2$ edges but to find $(1+\alpha)N$ edges with least weights, where $(N-3)/2 \gg \alpha \geqslant -1/N$. With this observation in mind, we employ a divide-and-conquer technique to build an MST with improved efficiency.

In general, a divide-and-conquer paradigm consists of three steps according to [14]:

1. Divide step. The problem is divided into a collection of subproblems that are similar to the original problem but smaller in size.
2. Conquer step. The subproblems are solved separately, and corresponding subresults are achieved.
3. Combine step. The subresults are combined to form the final result of the problem.

Following this divide-and-conquer paradigm, we constructed a two-stage fast approximate MST method as follows:

1. Divide-and-conquer stage
   - 1.1 Divide step. For a given dataset of N data points, K-means is applied to partition the dataset into $\sqrt{N}$ subsets.
   - 1.2 Conquer step. An exact MST algorithm such as Kruskal's or Prim's algorithm is employed to construct an exact MST for each subset.
   - 1.3 Combine step. $\sqrt{N}$ MSTs are combined using a connection criterion to form a primary approximate MST.
2. Refinement stage
   - 2.1 Partitions focused on borders of the clusters produced in the previous stage are constructed.
   - 2.2 A secondary approximate MST is constructed with the conquer and combine steps in the previous stage.
   - 2.3 The two approximate MSTs are merged and a new more accurate is obtained by using an exact MST algorithm.

The process is illustrated in Fig. 1. In the first stage, an approximate MST is produced. However, its accuracy is insufficient compared to the corresponding exact MST, because many of the data points that are located on the boundaries of the subsets are connected incorrectly in the MST. This is because an exact MST algorithm is applied only to data points within a subset but not to those crossing the boundaries of the subsets. To compensate for the drawback, a refinement stage is designed.

In the refinement stage, we re-partition the dataset so that the neighboring data points from different subsets will belong to the same partition. After this, the two approximate MSTs are merged, and the number of edges in the combined graph is at most $2(N-1)$. The final MST is built from this graph by an exact MST algorithm. The details of the method will be described in the following subsections.

2.2. Partition dataset with K-means

For two points connected by an edge in an MST, at least one is the nearest neighbor of the other, which implies that the connections have a locality property. Therefore, in the divide step, it is expected that the subsets preserve this locality. As K-means can partition some of local neighboring data points into the same group, we employ K-means to partition the dataset.

K-means requires the number of clusters to be known and the initial center points to be determined, and we will discuss these two problems below.

2.2.1. The number of clusters K

In this study, we set the number of clusters K to $\sqrt{N}$ based on the following two reasons. One is that the maximum number of clusters in some clustering algorithms is often set to $\sqrt{N}$ as a rule of thumb [5,41]. That means if a dataset is partitioned into $\sqrt{N}$ subsets, each subset may consist of data points coming from an identical genuine cluster so that the requirement of the locality property when constructing an MST is met.

The other reason is that the overall time complexity of the proposed approximate MST algorithm is minimized if K is set to $\sqrt{N}$, assuming that the data points are equally divided into the clusters. This choice will be theoretically and experimentally studied in more detail in Sections 3 and 4, respectively.

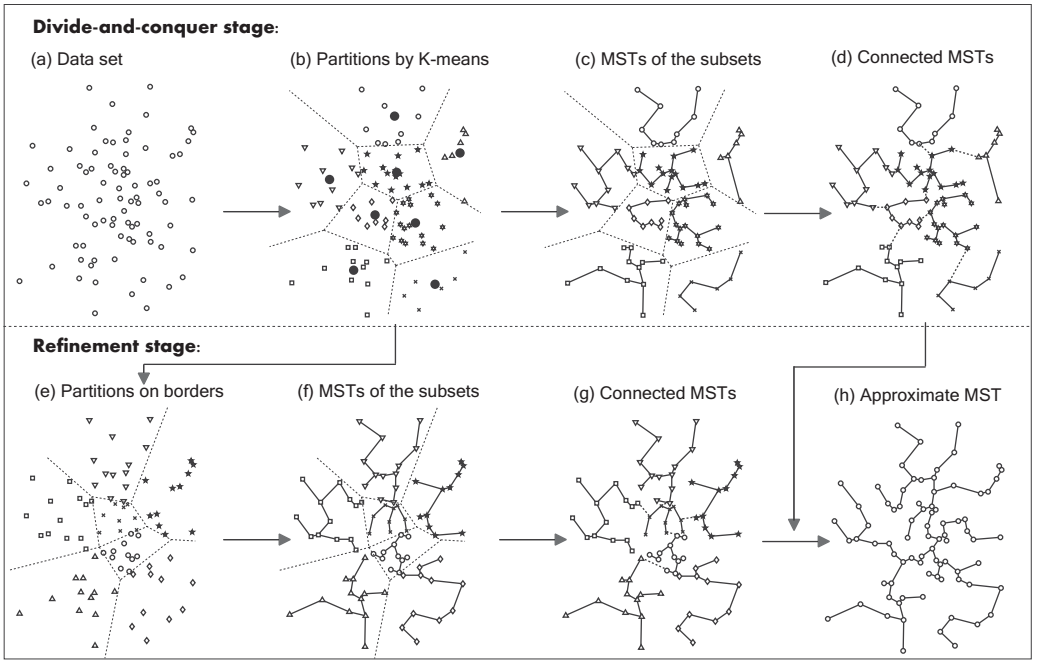


Fig. 1. The scheme of the proposed FMST algorithm. (a) A given dataset. (b) The dataset is partitioned into $\sqrt{N}$ subsets by K-means. The dashed lines form the corresponding Voronoi graph with respect to cluster centers (the big gray circles). (c) An exact MST algorithm is applied to each subset. (d) MSTs of the subsets are connected. (e) The dataset is partitioned again so that the neighboring data points in different subsets of (b) are partitioned into identical partitions. (f) An exact MST algorithm such as Prim's algorithm is used again on the secondary partition. (g) MSTs of the subsets are connected. (h) A more accurate approximate MST is produced by merging the two approximate MSTs in (d) and (g) respectively.

2.2.2. Initialization of K-means

Clustering results of K-means are sensitive to the initial cluster centers. A bad selection of the initial cluster centers may have negative effects on the time complexity and accuracy of the proposed method. However, we still randomly select the initial centers due to the following considerations.

First, although a random selection may lead to a skewed partition, such as a linear partition, the time complexity of the proposed method is still $O(N^{1.5})$, see Theorem 2 in Section 4. Second, in the proposed method, a refinement stage is designed to cope with the data points on the cluster boundaries. This process makes the accuracy relatively stable, and random selection of initial cluster centers is reasonable.

2.2.3. Divide-and-conquer algorithm

After the dataset has been divided into $\sqrt{N}$ subsets by K-means, the MSTs of the subsets are constructed with an exact MST algorithm, such as Prim's or Kruskal's. This corresponds to the conquer step in the divide and conquer scheme, it is trivial and illustrated in Fig. 1(c). The algorithm of K-means based on divide and conquer is described as follows:

Divide and Conquer Using K-means (DAC)
Input: Dataset X;
Output: MSTs of the subsets partitioned from X

Step 1. Set the number of subsets $K = \sqrt{N}$.
Step 2. Apply K-means to X to achieve K subsets $S = \{S_1, \ldots, S_K\}$, where the initial centers are randomly selected.
Step 3. Apply an exact MST algorithm to each subset in S, and an MST of S_i, denoted by $MST(S_i)$, is obtained, where $1 \leqslant i \leqslant K$.

The next step is to combine the MSTs of the K subsets into a whole MST.

2.3. Combine MSTs of the K subsets

An intuitive solution to combining MSTs is brute force: For the MST of a cluster, the shortest edge between it and the MSTs of other clusters is computed. But this solution is time consuming, and therefore a fast MST-based effective solution is also presented. The two solutions are discussed below.

2.3.1. Brute force solution

Suppose we combine a subset S_l with another subset, where $1 \leqslant l \leqslant K$. Let x_i, x_j be data points and $x_i \in S_l, x_j \in X - S_l$. The edge that connects S_l to another subset can be found by brute force:

$$e = \arg\min_{e_i \in E_l} \rho(e_i) \tag{1}$$

where $E_l = \{e(x_i, x_j) | x_i \in S_l \wedge x_j \in X - S_l\}, e(x_i, x_j)$ is the edge between vertices x_i and x_j, $\rho(e_i)$ is the weight of edge e_i. The whole MST is obtained by iteratively adding e into the MSTs and finding the new connecting edge between the merged subset and the remaining part. This process is similar to single-link clustering [21].

However, the computational cost of the brute force method is high. Suppose that each subset has an equal size of N/K, and K is an even number. The running time T_c of combining the K trees into the whole MST is:

$$\begin{aligned} T_c &= 2 \times \left\{ \frac{N}{K} \times \frac{(K-1) \times N}{K} + \frac{2 \times N}{K} \times \frac{(K-2) \times N}{K} + \cdots + \frac{(K/2) \times N}{K} \times \frac{(K/2) \times N}{K} \right\} = \left(\frac{K^2}{6} + \frac{K}{4} - \frac{1}{6} \right) \times \frac{N^2}{K} \\ &= O(KN^2) = O(N^{2.5}) \end{aligned} \tag{2}$$

Consequently, a more efficient combining method is needed.

2.3.2. MST-based solution

The efficiency of the combining process can be improved in two aspects. First, in each combining iteration only one pair of neighboring subsets is considered in finding the connecting edge. Intuitively, it is not necessary to take into account subsets that are far from each other, because no edge in an exact MST connects the subsets. This consideration will save some computations. Second, to determine the connecting edge of a pair of neighboring subsets, the data points in the two subsets will be scanned only once. The implementation of the two techniques is discussed in detail.

Determine the neighboring subsets. As the aforementioned brute force solution runs in the same way as single-link clustering [24] and all the information required by single-link can be provided by the corresponding MST of the same data, we make use of the MST to determine the neighboring subsets and improve the efficiency of the combination process.

If each subset has one representative, an MST of the representatives of the K subsets can roughly indicate which pairs of subsets could be connected. For simplicity, the mean point, called the center, of a subset is selected as its representative. After an MST of the centers (MST_{cen}) is constructed, each pair of subsets whose centers are connected by an edge of MST_{cen} is combined. Although not all of the neighboring subsets can be discovered by MST_{cen}, the dedicated refinement stage could remedy this drawback to some extent.

The centers of the subsets in Fig. 1(c) are illustrated as the solid points in Fig. 2(a), and MST_{cen} is composed of the dashed edges in Fig. 2(b).

Determine the connecting edges. To combine MSTs of a pair of neighboring subsets, an intuitive way is to find the shortest edge between the two subsets and connect the MSTs by this edge. Under the condition of an average partition, finding the shortest edge between two subsets takes N steps, and therefore, the time complexity of the whole connection process is $O(N^{1.5})$. Although this does not increase the total time complexity of the proposed method, the absolute running time is still somewhat high.

To make the connecting process faster, a novel way to detect the connecting edges is illustrated in Fig. 3. Here, c_2 and c_4 are the centers of the subset S_2 and S_4, respectively. Suppose a is the nearest point to c_4 from S_2, and b is the nearest point to c_2 from S_4. The edge $e(a, b)$ is selected as the connecting edge between S_2 and S_4. The computational cost of this is low. Although the edges found are not always optimal, this can be compensated by the refinement stage.

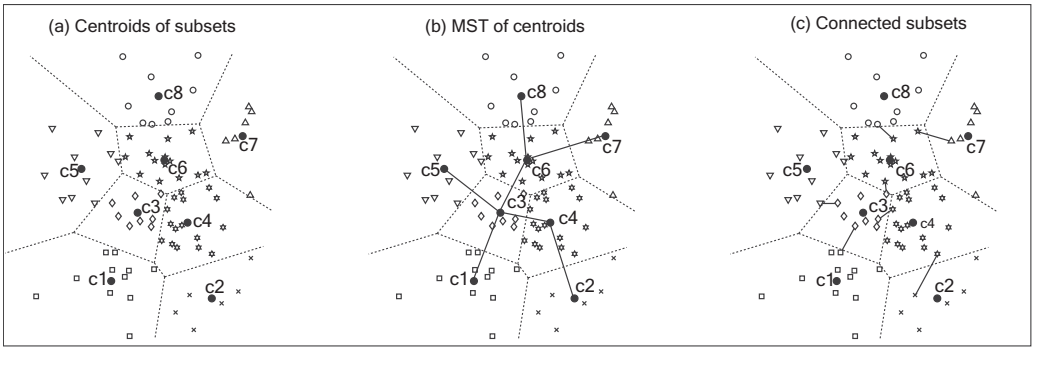


Fig. 2. The combine step of MSTs of the proposed algorithm. In (a), centers of the partitions (c1, . . . , c8) are calculated. In (b), a MST of the centers, MST_{cen}, is constructed with an exact MST algorithm. In (c), each pair of subsets whose centers are neighbors with respect to MST_{cen} in (b) is connected.

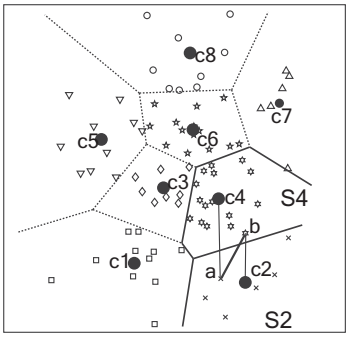


Fig. 3. Detecting the connecting edge between S_4 and S_2.

Consequently, the algorithm for combining the MSTs of the subsets is summarized as follows:

Combine Algorithm (CA)
Input: MSTs of the subsets partitioned from X : $MST(S_1), \ldots, MST(S_K)$.
Output: Approximate MST of X, denoted by MST_1, and MST of the centers of $S_1, \ldots, S_K$, denoted by MST_{cen};

Step 1. Compute the center c_i of subset $S_i, 1 \leqslant i \leqslant K$.
Step 2. Construct an MST, MST_{cen}, of $c_1, \ldots, c_K$ by an exact MST algorithm.
Step 3. For each pair of subsets (S_i, S_j) that their centers c_i and c_j are connected by an edge $e \in MST_{cen}$, discover the edge by **DCE** (Detect the Connecting Edge) that connects $MST(S_i)$ and $MST(S_j)$.
Step 4. Add all the connecting edges discovered in Step 3 to $MST(S_1), \ldots, MST(S_K)$, and MST_1 is achieved.

Detect the Connecting Edge (DCE)
Input: A pair of subsets to be connected, (S_i, S_j);
Output: The edge connecting $MST(S_i)$ and $MST(S_j)$;

Step 1. Find the data point $a \in S_i$ such that the distance between a and the center of S_j is minimized.
Step 2. Find the data point $b \in S_j$ such that the distance between b and the center of S_i is minimized.
Step 3. Select edge $e(a, b)$ as the connecting edge.

2.4. Refine the MST focusing on boundaries

However, the accuracy of the approximate MST achieved so far is far from the exact MST. The reason is that, when the MST of a subset is built, the data points that lie in the boundary of the subset are considered only within the subset, but not across the boundaries. In Fig. 4, subsets S_6 and S_3 have a common boundary, and their MSTs are constructed independently. In the MST of S_3, point a and b are connected to each other. But in the exact MST they are connected to the points in S_6 rather than in S_3. Therefore, data points located on the boundaries are prone to be misconnected. Based on this observation, the refinement stage is designed.

2.4.1. Partition dataset focusing on boundaries

In this step, another complimentary partition is constructed so that the clusters would locate at the boundary areas of the previous K-means partition. We first calculate the midpoints of each edge of MST_{cen}. These midpoints generally lie near the boundaries, and are therefore employed as the initial cluster centers. The dataset is then partitioned by K-means. The partition process of this stage is different from that of the first stage. In this stage, the initial cluster centers are specified and the maximum number of iterations is set to 1 for the purpose of focusing on the boundaries. Since MST_{cen} has $\sqrt{N}-1$ edges, there will be $\sqrt{N}-1$ clusters in this stage. The process is illustrated in Fig. 5.

In Fig. 5(a), the midpoints of the edges of MST_{cen} are computed as $m_1, \ldots, m_7$. In Fig. 5(b), the dataset is partitioned with respect to these seven midpoints.

2.4.2. Build secondary approximate MST

After the dataset has been re-partitioned, the conquer and combine steps are similar to those used for producing the primary approximate MST. The algorithm is summarized as follows:

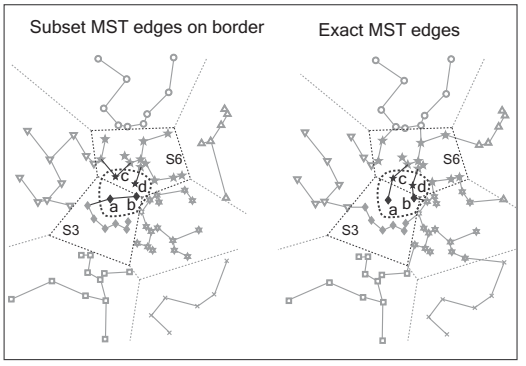


Fig. 4. The data points on the subset boundaries are prone to be misconnected.

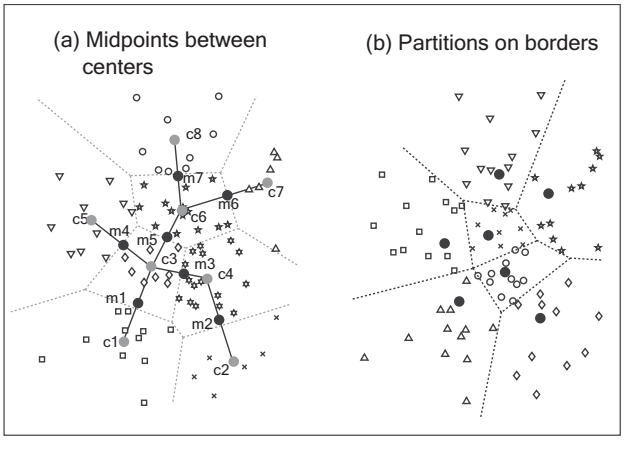


Fig. 5. Boundary-based partition. In (a), the black solid points, $m_1, \ldots, m_7$, are the midpoints of the edges of MST_{cen}. In (b), each data point is assigned to its nearest midpoint, and the dataset is partitioned by the midpoints. The corresponding Voronoi graph is with respect to the midpoints.

Secondary Approximate MST (SAM)
Input: MST of the subset centers MST_{cen}, dataset X;
Output: Approximate MST of X, MST_2;

Step 1. Compute the midpoint m_i of an edge $e_i \in MST_{cen}$, where $1 \leqslant i \leqslant K-1$.
Step 2. Partition dataset X into $K-1$ subsets, $S'_1, \ldots, S'_{K-1}$, by assigning each point to its nearest point from $m_1, \ldots, m_{K-1}$.
Step 3. Build MSTs, $MST(S'_1), \ldots, MST(S'_{K-1})$, with an exact MST algorithm.
Step 4. Combine the $K-1$ MSTs with **CA** to produce an approximate MST MST_2.

2.5. Combine two rounds of approximate MSTs

So far we have two approximate MSTs on dataset X, MST_1 and MST_2. To produce the final approximate MST, we first merge the two approximate MSTs to produce a graph, which has no more than $2(N-1)$ edges, and then apply an exact MST algorithm to this graph to achieve the final approximate MST of X.

Finally, the overall algorithm of the proposed method is summarized as follows:

Fast MST (FMST)
Input: Dataset X;
Output: Approximate MST of X;

(*continued on next page*)

Step 1. Apply **DAC** to X to produce the K MSTs.
Step 2. Apply **CA** to the K MSTs to produce the first approximate MST, MST_1, and the MST of the subset centers, MST_{cen}.
Step 3. Apply **SAM** to MST_{cen} and X to generate the secondary approximate MST, MST_2.
Step 4. Merge MST_1 and MST_2 into a graph G.
Step 5. Apply an exact MST algorithm to G, and the final approximate MST is achieved.

3. Complexity and accuracy analysis

3.1. Complexity analysis

The overall time complexity of the proposed algorithm **FMST**, T_{FMST}, can be evaluated as:

$$T_{FMST} = T_{DAC} + T_{CA} + T_{SAM} + T_{COM} \tag{3}$$

where T_{DAC}, T_{CA} and T_{SAM} are the time complexities of the algorithms **DAC**, **CA** and **SAM**, respectively, and T_{COM} is the running time of an exact MST algorithm on the combination of MST_1 and MST_2.

DAC consists of two operations: partitioning the dataset X with K-means and constructing the MSTs of the subsets with an exact MST algorithm. Now we consider the time complexity of DAC by the following theorems.

Theorem 1. *Suppose a dataset with N points is equally partitioned into K subsets by K-means, and an MST of each subset is produced by an exact algorithm. If the total running time for partitioning the dataset and constructing MSTs of the K subsets is T, then* $\arg\min_K T = \sqrt{N}$.

Proof. Suppose the dataset is partitioned into K clusters equally so that the number of data points in each cluster equals N/K. The time complexity of partitioning the dataset and constructing the MSTs of K subsets are $T_1 = NKId$ and $T_2 = K(N/K)^2$, respectively, where I is the number of iterations of K-means and d is the dimension of the dataset. The total complexity is $T = T_1 + T_2 = NKId + N^2/K$. To find the optimal K corresponding to the minimum T, we solve $\partial T/\partial K = NId - N^2/K^2 = 0$ which results in $K = \sqrt{N/Id}$. Therefore, $K = \sqrt{N}$ and $T = O(N^{1.5})$ under the assumption that $I \ll N$ and $d \ll N$. Because convergence of K-means is not necessary in our method, we set I to 20 in all of our experiments. For very high dimensional datasets, $d \ll N$ may not hold, but for modern large datasets it may hold. The situation for high dimensional datasets is discussed in Section 4.5. □

Although the above theorem holds under the ideal condition of average partition, it can be supported by more evidence when the condition is not satisfied, for example, linear partition and multinomial partition.

Theorem 2. *Suppose a dataset is linearly partitioned into K subsets. If* $K = \sqrt{N}$, *then the time complexity is* $O(N^{1.5})$.

Proof. Let $n_1, n_2, \ldots, n_K$ be the numbers of data points of the K clusters. The K numbers form an arithmetic series, namely, $n_i - n_{i-1} = c$, where $n_1 = 0$ and c is a constant. The arithmetic series sums up to $sum = K * n_K/2 = N$, and thus, we have $n_K = 2N/K$ and $c = 2N/[K(K-1)]$. The time complexity of constructing MSTs of the subsets is then:

$$\begin{aligned} T_2 &= n_1^2 + n_2^2 + \cdots + n_{K-1}^2 = c^2 + (2c)^2 + \cdots + [(K-1)c]^2 = c^2 \times \frac{(K-1)K(2K-1)}{6} \\ &= \left[\frac{2N}{(K-1)K}\right]^2 \times \frac{(K-1)K(2K-1)}{6} = \frac{2}{3} \times \frac{(2K-1)N^2}{K(K-1)} \end{aligned} \tag{4}$$

If $K = \sqrt{N}$, then $T_2 = \frac{4}{3}N^{1.5} + \frac{2}{3}\frac{N^{1.5}}{N^{0.5}-1} = O(N^{1.5})$. Therefore, $T = T_1 + T_2 = O(N^{1.5})$ holds. □

Theorem 3. *Suppose a dataset is partitioned into K subsets, and the sizes of the K subsets follow a multinomial distribution. If* $K = \sqrt{N}$, *then the time complexity is* $O(N^{1.5})$.

Proof. Let $n_1, n_2, \ldots, n_K$ be the numbers of data points of the K clusters. Suppose the data points are randomly assigned into the K clusters, and $n_1, n_2, \ldots, n_K \sim Multinomial(N, \frac{1}{K}, \ldots, \frac{1}{K})$. We have $Ex(n_i) = N/K$ and $Var(n_i) = (N/K) * (1 - 1/K)$. Since $Ex(n_i^2) = [Ex(n_i)]^2 + Var(n_i) = N^2/K^2 + N * (K-1)/K^2$, the expected complexity of constructing MSTs is $T_2 = \sum_{i=1}^{K} n_i^2 = K * Ex(n_i^2) = N^2/K + N * (K-1)/K$, if $K = \sqrt{N}$, then $T_2 = O(N^{1.5})$. Therefore $T = T_1 + T_2 = O(N^{1.5})$ holds. □

According to the above theorems, we have $T_{DAC} = O(N^{1.5})$.

In **CA**, the time complexity of computing the mean points of the subsets is $O(N)$, as one scan of the dataset is enough. Constructing MST of the K mean points by an exact MST algorithm takes only $O(N)$ time. In Step 3, the number of subset

pairs is $K-1$, and for each pair, determining the connecting edge by **DCE** requires one scan on the two subsets, respectively. Thus, the time complexity of Step 3 is $O(2N \times (K-1)/K)$, which equals $O(N)$. The total computational cost of **CA** is therefore $O(N)$.

In **SAM**, Step 1 computes $K-1$ midpoints, which takes $O(N^{0.5})$ time. Step 2 takes $O(N \times (K-1))$ to partition the dataset. The running time of Step 3 is $O((K-1) \times N^2/(K-1)^2) = O(N^2/(K-1))$. Step 4 is to call **CA** and has the time complexity of $O(N)$. Therefore, the time complexity of **SAM** is $O(N^{1.5})$.

The number of edges in the graph that is formed by combining MST_1 and MST_2 is at most $2(N-1)$. The time complexity of applying an exact MST algorithm to this graph is only $O(2(N-1)\log N)$. Thus, $T_{COM} = O(N \log N)$.

To sum up, the time cost of the proposed algorithm is $(c_1 N^{1.5} + c_2 N \log N + c_3 N + N^{0.5}) = O(N^1.5)$. The hidden constants are not remarkable; according to our experiments we estimate them as $c1 = 3 + d * I, c2 = 2, c3 = 5$. The space complexity of the algorithm is the same as that of K-means and Prim, which are $O(N)$ if a Fibonacci heap is used within Prim's algorithm.

3.2. Accuracy analysis

Most inaccuracies originate from points that are in the boundary regions of the partitions of K-means. The secondary partition is generated in order to capture these problematic points into the same clusters. Inaccuracies after the refinement stage can, therefore, originate only if two points should be connected by the exact MST, but are partitioned into different clusters both in the primary and in the secondary partition, and neither of the two conquer stages will be able to connect these points. In Fig. 6, few such pair of points are shown that belong to different clusters in both partitions. For example, point a and b belong to different clusters of the first partition, but are in the same cluster of the second.

Since partitions generated by K-means form a Voronoi graph [16], the analysis of the inaccuracy can be related to the degree by which the secondary Voronoi edges overlap that of the Voronoi edges of the primary partition. Let $|E|$ denote the number of edges of a Voronoi graph, in two-dimensional space, $|E|$ is bounded by $K-1 \leqslant |E| \leq 3K-6$, where K is the number of clusters (the Voronoi regions). In a higher dimensional case it is more difficult to analyze.

A favorable case is demonstrated in Fig. 7. The first row is a dataset which consists of 400 points and is randomly distributed. In the second row, the dataset is partitioned into six clusters by K-means, and a collinear Voronoi graph is achieved. In the third row, the secondary partition has five clusters, each of which completely cover one boundary region in the second row. An exact MST is produced in the last row.

4. Experiments

In this section, experimental results are presented to illustrate the efficiency and the accuracy of the proposed fast approximate MST algorithm. The accuracy of FMST is tested with both synthetic datasets and real applications. As a framework, the proposed algorithm can be incorporated with any exact or even approximate MST algorithm, of which the running time is definitely reduced. Here we only take into account Kruskal's and Prim's algorithms because of their popularity. As in Kruskal's algorithm, all the edges need to be sorted into nondecreasing order, it is difficult to apply the algorithm to large datasets. Furthermore, as Prim's algorithm may employ a Fibonacci heap to reduce the running time, we therefore use it rather than Kruskal's algorithm in our experiments as the exact MST algorithm.

Experiments were conducted on a PC with an Intel Core2 2.4 GHz CPU and 4 GB memory running Windows 7. The algorithm for testing the running time is implemented in C++, while the other tests are performed in Matlab (R2009b).

4.1. Running time

4.1.1. Running time on different datasets

We first perform experiments on four typical datasets with different sizes and dimensions to test the running time. The four datasets are described as Table 1.

Dataset t4.8k[1] is designed to test the CHAMELEON clustering algorithm in [28]. MNIST[2] is a dataset of ten handwriting digits and contains 60,000 training patterns and 10,000 test patterns of 784 dimensions, we use just the test set. The last two sets are from the UCI machine learning repository.[3] ConfLongDemo has eight attributes, of which only three numerical attributes are used here.

From each dataset, subsets with different sizes are randomly selected to test the running time as a function of data size. The subset sizes of the first two datasets gradually increase with step 20, the third with step 100 and the last with step 1000.

In general, the running time for constructing an MST of a dataset depends on the size of the dataset but not on the underlying structure of the dataset. In our FMST method, K-means is employed to partition a dataset, and the size of the subsets depends on the initialization of K-means and the distributions of the datasets, which leads to different time costs. We therefore perform FMST ten times on each dataset to alleviate the effects of the random initialization of K-means.

[1] http://glaros.dtc.umn.edu/gkhome/cluto/cluto/download.
[2] http://yann.lecun.com/exdb/mnist.
[3] http://archive.ics.uci.edu/ml/.

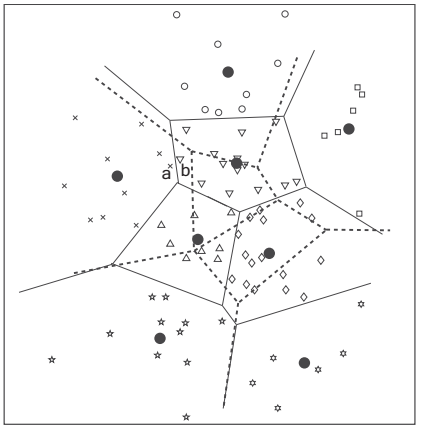


Fig. 6. Merge of two Voronoi graphs. Voronoi graph in solid line is corresponding to the first partition, and that in dashed line corresponding to the secondary partition. Only the first partition is illustrated.

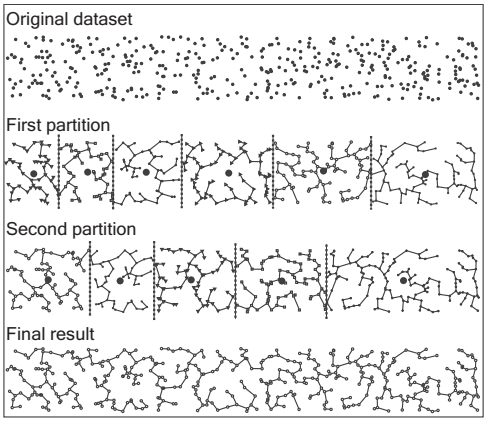


Fig. 7. The collinear Voronoi graph case.

Table 1
The description of four datasets.

	t4.8k	MNIST	ConfLongDemo	MiniBooNE
Data size	8000	10,000	164,860	130,065
Dimension	2	784	3	50

The running time of FMST and Prim's algorithm on the four datasets is illustrated in the first row of Fig. 8. From the results, we can see that FMST is computationally more efficient than Prim's algorithm, especially for the large datasets ConfLongDemo and MiniBooNE. The efficiency for MiniBooNE shown in the rightmost of the second and third row in Fig. 8, however, deteriorates because of the high dimensionality.

Although the complexity analysis indicates that the time complexity of the proposed FMST is $O(N^{1.5})$, the actual running time can be different. We analyzed the actual processing time by fitting an exponential function $T = aN^b$, where T is the running time and N is the number of data points. The results are shown in Table 2.

4.1.2. Running time with different Ks

We have discussed the number of clusters K and set it to $\sqrt{N}$ in Section 2.2.1, and have also presented some supporting theorems in Section 3. In practical applications, however, the value is slightly small. Some experiments were performed on

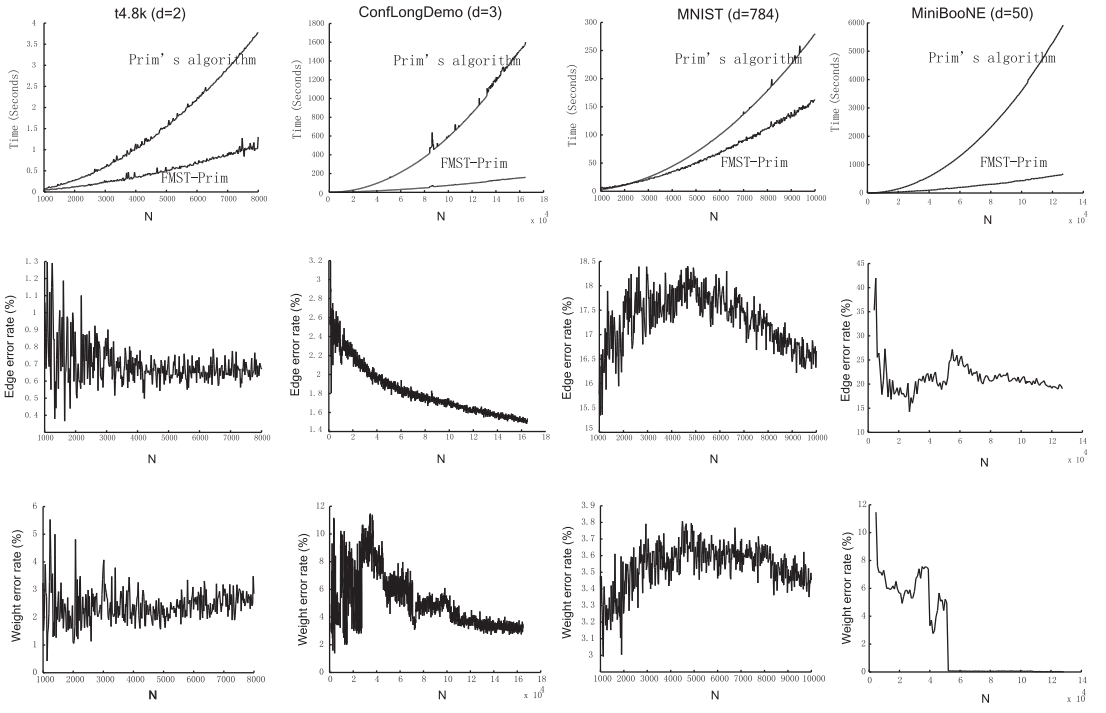


Fig. 8. The results of the test on the four datasets. FMST-Prime denotes the proposed method based on Prim's algorithm. The first row shows the running time of t4.8k, ConfLongDemo, MNIST and MiniBooNE, respectively. The second row shows corresponding edge error rates. The third row shows corresponding weight error rates.

dataset t4.8k and ConfLongDemo to study the effect of different Ks on running time. The experimental results are illustrated in Fig. 9, from which we find that if K is set to 38 for t4.8k and 120 for ConfLongDemo, the running time will be minimum. But according to the previous analysis, K would be set to $\sqrt{N}$, namely 89 and 406 for the two datasets, respectively. Therefore, K is practically set to $\frac{\sqrt{N}}{C}$, where $C > 1$. For dataset t4.8k and ConfLongDemo, C is approximately 3. The phenomenon is explained as follows.

From the analysis of the time complexity in Section 3, we can see that the main computational cost comes from K-means, in which a large K leads to a high cost. If partitions produced by K-means have the same size, when K is set to $\sqrt{N}$, the time complexity is minimized. However, the partitions practically have unbalanced sizes. From the viewpoint of divide-and-conquer, the proposed method with a large K will have a small time cost for constructing the meta-MSTs, but the unbalanced partitions can reduce this gain, and the large K only increases the time cost of K-means. Therefore, before K is increased to $\sqrt{N}$, the minimum time cost can be achieved.

4.2. Accuracy on synthetic datasets

4.2.1. Measures by edge error rate and weight error rate

The accuracy is another important aspect of FMST. Two accuracy rates are defined: edge error rate ER_{edge} and weight error rate ER_{weight}. Before ER_{edge} is defined, we present the notation of an equivalent edge of an MST, because the MST may not be unique. The equivalence property is described as:

Equivalence Property. Let T and T' be the two different MSTs of a dataset. For any edge $e \in (T \setminus T')$, there must exist another edge $e' \in (T' \setminus T)$ such that $(T' \setminus \{e'\}) \cup \{e\}$ is also an MST. We call e and e' a pair of equivalent edges.

Proof. The equivalency property can be operationally restated as: Let T and T' be the two different MSTs of a dataset, for any edge $e \in (T \setminus T')$, there must exist another edge $e' \in (T' \setminus T)$ such that $w(e) = w(e')$ and e connects T'_1 and T'_2, where T'_1 and T'_2 are the two subtrees generated by removing e' from T', $w(e)$ is the weight of e.

Let G be the cycle formed by $\{e\} \cup T'$, we have:

$$\forall e' \in (G \setminus \{e\} \setminus (T \cap T')), w(e) \geqslant w(e') \tag{5}$$

Otherwise, an edge in $G \setminus \{e\} \setminus (T \cap T')$ should be replaced by e when constructing T'.

Table 2
The exponent *b*s obtained by fitting $T = aN^b$. FMST denotes the proposed method.

	b			
	t4.8k	MNIST	ConfLongDemo	MiniBooNE
FMST	1.57	1.62	1.54	1.44
Prim's alg.	1.88	2.01	1.99	2.00

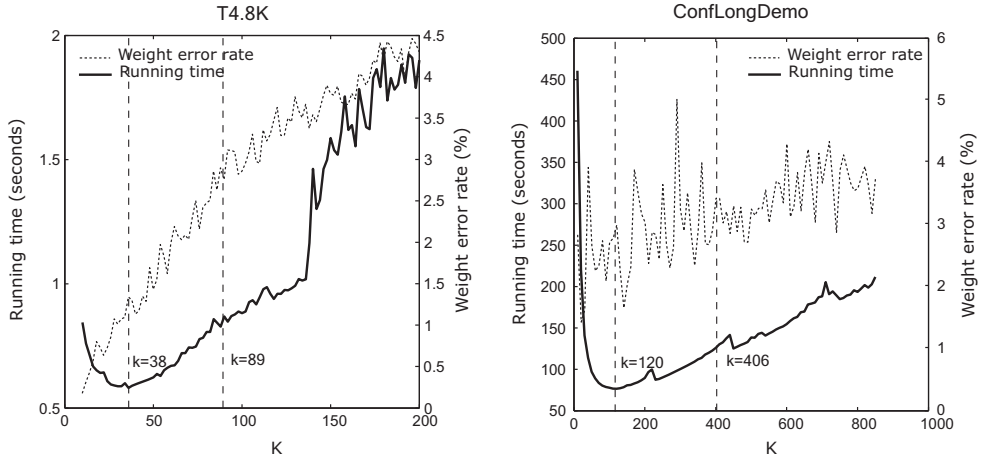


Fig. 9. Performances (running time and weight error rate) as a function of *K*. The left shows the running time and weight error rate of FMST on t4.8k, and the right on ConfLongDemo.

Furthermore, the following claim holds: there must exist at least one edge $e' \in (G \setminus \{e\} \setminus (T \cap T'))$, such that the cycle formed by $\{e'\} \cup T$ contains e. We prove this claim by contradiction.

Assuming that all the cycles G'_j formed by $\left\{e'_j\right\} \cup T$ do not contain e, where $e'_j \in (G \setminus \{e\} \setminus (T \cap T'))$, $1 \leqslant j \leqslant |G \setminus \{e\} \setminus (T \cap T')|$, let $G_{union} = (G'_1 \setminus \{e'_1\}) \cup \cdots \cup (G'_l \setminus \{e'_l\})$, where $l = |G \setminus \{e\} \setminus (T \cap T')|$. G can be expressed as $\{e\} \cup \{e'_1\} \cup \cdots \cup \{e'_l\} \cup G_{delta}$, where $G_{delta} \subset (T \cap T')$. As G is a cycle, $G_{union} \cup \{e\} \cup G_{delta}$ must also be a cycle, this is contradictory because $G_{union} \subset T, G_{delta} \subset T$ and $e \in T$. Therefore the claim is correct.

As a result, there must exist at least one edge $e' \in (G \setminus \{e\} \setminus (T \cap T'))$ such that $w(e') \geqslant w(e)$.

Combining this result with (5), we have the following: for $e \in (T \setminus T')$, there must exist an edge $e' \in (T' \setminus T)$ such that $w(e) = w(e')$. Furthermore, as e and e' are in the same cycle G, $(T' \setminus \{e'\}) \cup \{e\}$ is still an MST. □

According to the equivalency property, we define a criterion to determine whether an edge belongs to an MST:

Let T be an MST and e be an edge of a graph. If there exists an edge $e' \in T$ such that $|e| = |e'|$ and e connects T_1 and T_2, where T_1 and T_2 are the two subtrees achieved by removing e' from T, then e is a *correct edge*, i.e., belongs to an MST.

Suppose E_{appr} is the set of the correct edges in an approximate MST, the edge error rate ER_{edge} is defined as:

$$ER_{edge} = \frac{N - |E_{appr}| - 1}{N - 1} \tag{6}$$

The second measure is defined as the difference of the sum of the weights in FMST and the exact MST, which is called the weight error rate ER_{weight}:

$$ER_{weight} = \frac{W_{appr} - W_{exact}}{W_{exact}} \tag{7}$$

where W_{exact} and W_{appr} are the sum of the weights of the exact MST and FMST, respectively.

The edge error rates and weight error rates of the four datasets are shown in the third row of Fig. 8. We can see that both the edge error rate and the weight error rate decrease with the increase in data size. For datasets with high dimensions, the edge error rates are greater, for example, the maximum edge error rates of MNIST are approximately 18.5%, while those of t4.8k and ConfLongDemo are less than 3.2%. In contrast, the weight error rates decrease when the dimensionality increases. For instance, the weight error rates of MNIST are less than 3.9%. This is the phenomenon of the curse of dimensionality. The high dimensional case will be discussed further in Section 4.5.

4.2.2. Accuracy with different Ks

Globally, the edge and weight error rates increase with K. This is because the greater the K, the greater the number of split boundaries, from which the error edges come. But when K is small, the error rates increase slowly with K. In Fig. 9, we can see that the weight error rates are still low when K is set to approximate $\frac{\sqrt{N}}{3}$.

4.2.3. Comparison to other approaches

We first compare the proposed FMST with the approach in [34]. The approach in [34] is designed to detect the clusters efficiently by removing the longer edges of the MST, and an approximate MST is generated in the first stage.

The accuracy of the approximate MST produced in [34] is relevant to a parameter: the number of the nearest neighbors of a data point. This parameter is used to update the priority queue when an algorithm like Prim's is employed to construct an MST. In general, the larger the number, the more accurate the approximate MST. However, this parameter is also relevant to the computational cost of the approximate MST, which is $O(dN(b+k+k\log N))$, where k is the number of nearest neighbors and b is the number bits of a Hilbert number. Here we only focus on the accuracy of the method, and the number of nearest neighbors is set to $N*0.05, N*0.10, N*0.15$, respectively. The accuracy is tested on t4k.8k, and the result is shown in Fig. 10. From the result, the edge error rates are more than 22%, and much higher than that of FMST, even if the number of nearest neighbors is set to $N*0.15$, which leads to a loss in the computational efficiency of the method.

We then compare FMST with two other methods: MST using cover-tree by March et al. [35] and the divide-and-conquer approach by Wang et al. [45] on the following datasets: MNIST, ConfLongDemo, MiniBooNE and ConfLongDemo $\times$ 6. To compare the performances on a large data set, ConfLongDemo $\times$ 6 is generated. It has 989,160 data points, and is achieved as follows: Move two copies of ConfLongDemo to the right of the dataset along the first coordinate axis, and then copy the whole data and move the copy to the right along the second coordinate axis.

The results measured by running time (RT) and weight error rate in Table 3 confirm that Wang's approach is faster due to the recursive dividing of the data, but suffers from lower quality results, especially with the ConfLongDemo dataset, this is because the approach focuses on finding the longest edges of an MST in the early stage for efficient clustering but does not focus on constructing a high quality approximate MST. The method by March et al. is different and produces exact MSTs. It works very fast on lower dimensional datasets, but inefficiently on high dimensional data such as MNIST and MiniBooNE. FMST is slower than Wang's approach on all of the tested datasets, but has better quality. In [35], kd-tree and similar structures are used, which are known to work well with low-dimensional data. The proposed method is slower than March's method for lower dimensional datasets, but faster for the higher dimensional.

4.3. Accuracy on clustering

In this subsection, the accuracy of FMST is tested on a clustering application. Path-based clustering employs the minimax distance metric to measure the dissimilarities of data points [17,18]. For a pair of data points x_i, x_j, the minimax distance D_{ij} is defined as:

$$D_{ij} = \min_{\mathcal{P}_{ij}^k} \left\{ \max_{(x_p, x_{p+1}) \in \mathcal{P}_{ij}^k} d(x_p, x_{p+1}) \right\} \tag{8}$$

where $\mathcal{P}_{ij}^k$ denotes all possible paths between x_i and x_j and k is an index to enumerate the paths, and $d(x_p, x_{p+1})$ is the Euclidean distance between x_p and x_{p+1}.

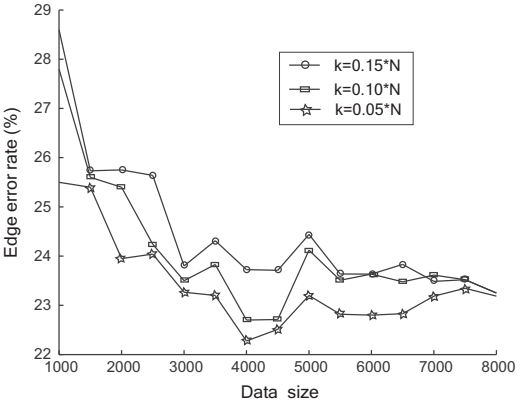


Fig. 10. The edge error rate of Lai's method on t4.8k.

Table 3
The proposed method FMST is compared to MST-Wang [45] and MST-March [35] methods.

Methods	MNIST		MiniBooNE		ConfLongDemo		ConfLong Demo × 6	
	RT(S)	ER_{weight} (%)	RT(S)	ER_{weight} (%)	RT(S)	ER_{weight} (%)	RT(S)	ER_{weight} (%)
FMST	164	3.3	781	0.3	174	0.5	16,201	0.2
MST-Wang	26	43.4	64	40.5	51	38.2	5262	46.8
MST-March	1135	0	2181	0	18	0	133	0

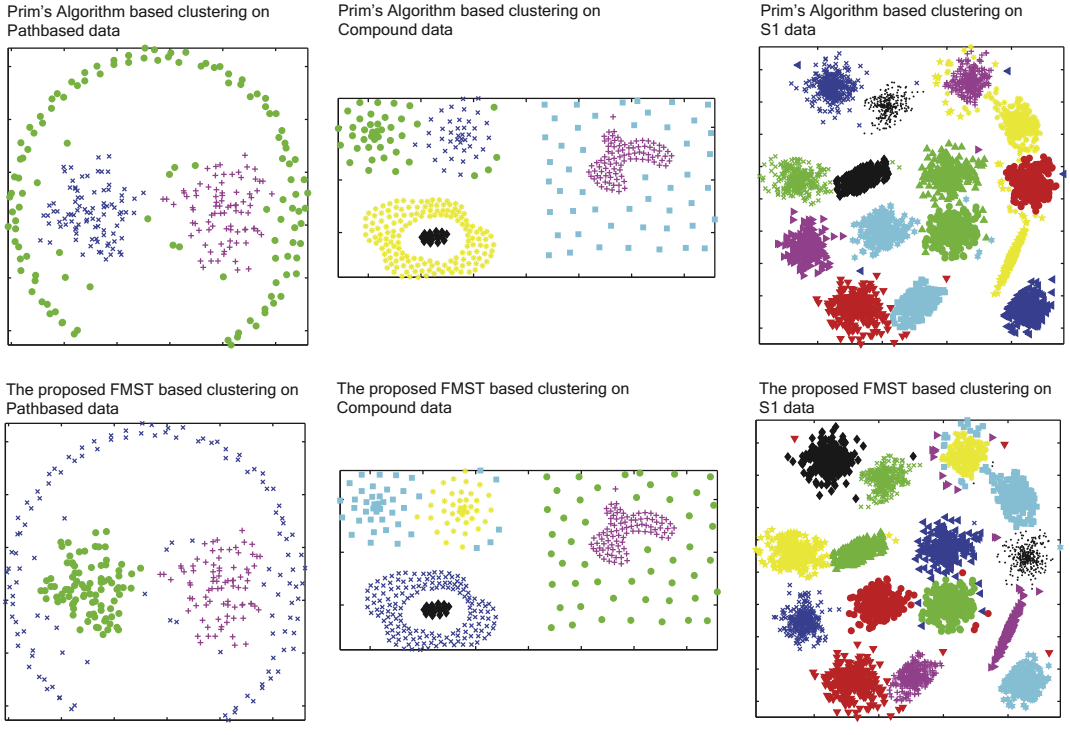


Fig. 11. Prim's algorithm and the proposed FMST based clustering results.

Table 4
The quantitative measures of clustering results. FMST denotes the proposed method.

Datasets	FMST				Prim's algorithm			
	Rand	AR	Jac	FM	Rand	AR	Jac	FM
Pathbased	0.937	0.859	0.829	0.906	0.942	0.870	0.841	0.913
Compound	0.993	0.982	0.973	0.986	0.994	0.984	0.977	0.988
S1	0.995	0.964	0.936	0.967	0.995	0.964	0.935	0.966

The minimax distance can be computed by an all-pair shortest path algorithm, such as the Floyd Warshall algorithm. However, this algorithm runs in time $O(N^3)$. An MST can be used to compute the minimax distance more efficiently in [31]. To make the path-based clustering robust to outliers, Chang and Yeung [8] improved the minimax distance and incorporated it into spectral clustering. We tested the FMST within this method on three synthetic datasets (Pathbased, Compound and S1).[4]

For computing the minimax distances, Prim's algorithm and FMST are used. In Fig. 11, one can see that the clustering results on three datasets are almost the same. The quantitative measures are given in Table 4, which contains four validity

[4] http://cs.joensuu.fi/sipu/datasets/.

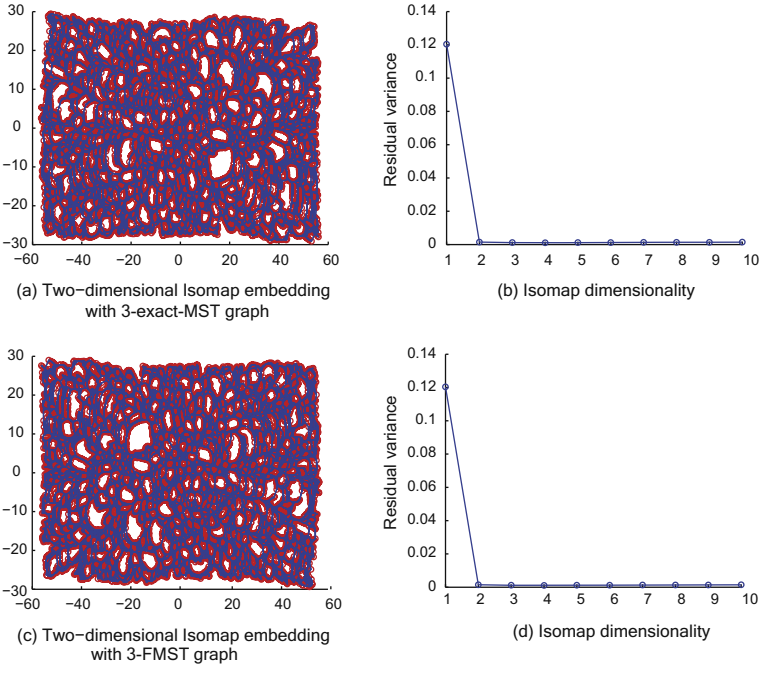


Fig. 12. Two 3-MST graph based ISOMAP results using exact MST (Prim's algorithm) and FMST, respectively. In (a) and (c), the two dimensional embedding is illustrated. (b) and (d) are corresponding resolutions.

indexes and indicates that the results on the first two datasets of Prim's algorithm-based clustering are slightly better than those of the FMST-based clustering.

4.4. Accuracy on manifold learning

MST has been used for manifold learning [48,49]. For a *K*NN based neighborhood graph, an improperly selected k may lead to a disconnected graph, and degrade the performance of manifold learning. To address this problem, Yang [48] used MSTs to construct a k-edge connected neighborhood graph. We implement the method of [48], with exact MST and FMST respectively, to reduce the dimensionality of a manifold.

The FMST-based and the exact MST-based dimensionality reduction were performed on the dataset Swiss-roll, which has 20,000 data points. In experiments, we selected the first 10,000 data points because of the memory requirement, and set $k = 3$. The accuracy of the FMST-based dimensionality reduction is compared with that of an exact MST-based dimensionality reduction in Fig. 12. The intrinsic dimensionality of Swiss-roll can be detected by the "elbow" of the curves in (b) and (d). Obviously, the MST graph based method and the FMST graph based method have almost identical residual variance, and both indicate the intrinsic dimensionality is 2. Furthermore, Fig. 12(a) and (c) shows that the two methods have similar two-dimensional embedding results.

4.5. Discussion on high dimensional datasets

As described in the experiments, the performances of both computation and accuracy of the proposed method are reduced when applied to high-dimensional datasets. Since the time complexity of FMST is $O(N^{1.5})$ under the condition of $d \ll N$, when the number of dimensions d is becoming large and even approximate to N, the computational cost will degrade to $O(N^{2.5})$. However, it is still more efficient than the corresponding Kruskal's or Prim's algorithms.

The accuracy of FMST is reduced because of the curse of dimensionality, which includes distance concentration phenomenon and the hubness phenomenon [40]. The distance concentration phenomenon is that the distances between all pairs of data points from a high dimensional dataset are almost equal, in other words, the traditional distance measures become ineffective, and the distances computed with the measures become unstable [25]. For constructing an MST in terms of these distances, the results of Kruskal's or Prim's algorithm are meaningless, so is the accuracy of the proposed FMST. Furthermore, the hubness phenomenon in a high-dimensional dataset, which implies some data points may appear in many more *K*NN

lists than other data points, shows that the nearest neighbors also become meaningless. Obviously, hubness affects the construction of an MST in the same way.

The intuitive way to address the above problems caused by the curse of dimensionality is to employ dimensionality reduction methods, such as ISOMAP, LLE, or subspace based methods for a concrete task in machine learning, such as subspace based clustering. Similarly, for constructing an MST of a high dimensional dataset, one may preprocess the dataset with dimensionality reduction or subspace based methods for the purpose of getting more meaningful MSTs.

5. Conclusion

In this paper, we have proposed a fast MST algorithm with a divide-and-conquer scheme. Under the assumption that the dataset is partitioned into equal sized subsets in the divide step, the time complexity of the proposed algorithm is theoretically $O(N^{1.5})$. Although this assumption may not hold practically, the complexity is still approximately $O(N^{1.5})$. The accuracy of the FMST was analyzed experimentally using edge error rate and weight error rate. Furthermore, two practical applications were considered, and the experiments indicate that the proposed FMST can be applied to large datasets.

Acknowledgments

This work was partially supported by the Natural Science Foundation of China (No. 61175054), the Center for International Mobility (CIMO), and sponsored by K.C. Wong Magna Fund in Ningbo University.

References

[1] H. Abdel-Wahab, I. Stoica, F. Sultan, K. Wilson, A simple algorithm for computing minimum spanning trees in the internet, Inform. Sci. 101 (1997) 47–69.
[2] L. An, Q.S. Xiang, S. Chavez, A fast implementation of the minimum spanning tree method for phase unwrapping, IEEE Trans. Med. Imag. 19 (2000) 805–808.
[3] B. Awerbuch, Optimal distributed algorithms for minimum weight spanning tree, counting, leader election, and related problems, in: Proceedings of the 19th ACM Symposium on Theory of Computing, 1987.
[4] D.A. Bader, G. Cong, Fast shared-memory algorithms for computing the minimum spanning forest of sparse graphs, J. Paral. Distrib. Comput. 66 (2006) 1366–1378.
[5] J.C. Bezdek, N.R. Pal, Some new indexes of cluster validity, IEEE Trans. Syst., Man Cybernet., Part B 28 (1998) 301–315.
[6] O. Borůvka, O jistém problému minimálním (About a Certain Minimal Problem), Práce moravské přírodovědecké společnosti v Brně III (1926) 37–58 (in Czech with German summary).
[7] P.B. Callahan, S.R. Kosaraju, Faster algorithms for some geometric graph problems in higher dimensions, in: Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete algorithms, 1993.
[8] H. Chang, D.Y. Yeung, Robust path-based spectral clustering, Patt. Recog. 41 (2008) 191–203.
[9] B. Chazelle, A minimum spanning tree algorithm with inverse-Ackermann type complexity, J. ACM 47 (2000) 1028–1047.
[10] G. Chen et al, The multi-criteria minimum spanning tree problem based genetic algorithm, Inform. Sci. 177 (2007) 5050–5063.
[11] D. Cheriton, R.E. Tarjan, Finding minimum spanning trees, SIAM J. Comput. 5 (1976) 24–742.
[12] K.W. Chong, Y. Han, T.W. Lam, Concurrent threads and optimal parallel minimum spanning trees algorithm, J. ACM 48 (2001) 297–323.
[13] G. Choquet, Etude de certains réseaux de routes, Comptesrendus de l'Acadmie des Sciences 206 (1938) 310 (in French).
[14] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, second ed., The MIT Press, 2001.
[15] E.W. Dijkstra, A note on two problems in connexion with graphs, Numer. Math. 1 (1959) 269–271.
[16] Q. Du, V. Faber, M. Gunzburger, Centroidal Voronoi tessellations: applications and algorithms, SIAM Rev. 41 (1999) 637–676.
[17] B. Fischer, J.M. Buhmann, Path-based clustering for grouping of smooth curves and texture segmentation, IEEE Trans. Pattern Anal. Mach. Intell. 25 (2003) 513–518.
[18] B. Fischer, J.M. Buhmann, Bagging for path-based clustering, IEEE Trans. Pattern Anal. Mach. Intell. 25 (2003) 1411–1415.
[19] K. Florek, J. Łkaszewicz, H. Perkal, H. Steinhaus, S. Zubrzycki, Sur la liaison et la division des points d'un ensemble fini, Colloq. Mathemat. 2 (1951) 282–285.
[20] M.L. Fredman, R.E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, J. ACM 34 (1987) 596–615.
[21] H.N. Gabow, Z. Galil, T.H. Spencer, R.E. Tarjan, Efficient algorithms for finding minimum spanning trees in undirected and directed graphs, Combinatorica 6 (1986) 109–122.
[22] H.N. Gabow, Z. Galil, T.H. Spencer, Efficient implementation of graph algorithms using contraction, J. ACM 36 (1989) 540–572.
[23] R.G. Gallager, P.A. Humblet, P.M. Spira, A distributed algorithm for minimum-weight spanning trees, ACM Trans. Program. Lang. Syst. 5 (1983) 66–77.
[24] J.C. Gower, G.J.S. Ross, Minimum spanning trees and single linkage cluster analysis, J. R. Statist. Soc., Ser. C (Appl. Statist.) 18 (1969) 54–64.
[25] C.M. Hsu, M.S. Chen, On the design and applicability of distance functions in high-dimensional data space, IEEE Trans. Knowl. Data Eng. 21 (2009) 523–536.
[26] V. Jarník, O jistém problému minimálním (About a certain minimal problem), Práce moravské přírodovědecké společnosti v Brně VI (1930) 57–63 (in Czech).
[27] P. Juszczak, D.M.J. Tax, E. Pękalska, R.P.W. Duin, Minimum spanning tree based one-class classifier, Neurocomputing 72 (2009) 1859–1869.
[28] G. Karypis, E.H. Han, V. Kumar, CHAMELEON: a hierarchical clustering algorithm using dynamic modeling, IEEE Trans. Comput. 32 (1999) 68–75.
[29] M. Khan, G. Pandurangan, A fast distributed approximation algorithm for minimum spanning trees, Distrib. Comput. 20 (2008) 391–402.
[30] K. Li, S. Kwong, J. Cao, M. Li, J. Zheng, R. Shen, Achieving balance between proximity and diversity in multi-objective evolutionary algorithm, Inform. Sci. 182 (2012) 220–242.
[31] K.H. Kim, S. Choi, Neighbor search with global geometry: a minimax message passing algorithm, in: Proceedings of the 24th International Conference on Machine Learning, 2007, pp. 401–408.
[32] J.B. Kruskal, On the shortest spanning subtree of a graph and the traveling salesman problem, Proc. Am. Math. Soc. 7 (1956) 48–50.
[33] B. Lacevic, E. Amaldi, Ectropy of diversity measures for populations in Euclidean space, Inform. Sci. 181 (2011) 2316–2339.
[34] C. Lai, T. Rafa, D.E. Nelson, Approximate minimum spanning tree clustering in high-dimensional space, Intell. Data Anal. 13 (2009) 575–597.
[35] W.B. March, P. Ram, A.G. Gray, Fast euclidean minimum spanning tree: algorithm, analysis, and applications, in: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2010.
[36] T. Öncan, Design of capacitated minimum spanning tree with uncertain cost and demand parameters, Inform. Sci. 177 (2007) 4354–4367.
[37] D. Peleg, V. Rubinovich, A near tight lower bound on the time complexity of distributed minimum spanning tree construction, SIAM J. Comput. 30 (2000) 1427–1442.
[38] S. Pettie, V. Ramachandran, A randomized time-work optimal parallel algorithm for finding a minimum spanning forest, SIAM J. Comput. 31 (2000) 1879–1895.
[39] R.C. Prim, Shortest connection networks and some generalizations, Bell Syst. Tech. J. 36 (1957) 567–574.

[40] M. Radovanovic, A. Nanopoulos, M. Ivanovic, Hubs in space: popular nearest neighbors in high-dimensional data, J. Mach. Learn. Res. 11 (2010) 2487–2531.
[41] M.R. Rezaee, B.P.F. Lelieveldt, J.H.C. Reiber, A new cluster validity index for the fuzzy c-mean, Patt. Recog. Lett. 19 (1998) 237–246.
[42] M. Sollin, Le trace de canalisation, in: C. Berge, A. Ghouilla-Houri (Eds.), Programming, Games, and Transportation Networks, Wiley, New York, 1965 (in French).
[43] S. Sundar, A. Singh, A swarm intelligence approach to the quadratic minimum spanning tree problem, Inform. Sci. 180 (2010) 3182–3191.
[44] P.M. Vaidya, Minimum spanning trees in k-dimensional space, SIAM J. Comput. 17 (1988) 572–582.
[45] X. Wang, X. Wang, D.M. Wilkes, A divide-and-conquer approach for minimum spanning tree-based clustering, IEEE Trans. Knowl. Data Eng. 21 (2009) 945–958.
[46] Y. Xu, V. Olman, D. Xu, Clustering gene expression data using a graph-theoretic approach: an application of minimum spanning trees, Bioinformatics 18 (2002) 536–545.
[47] Y. Xu, E.C. Uberbacher, 2D image segmentation using minimum spanning trees, Image Vis. Comput. 15 (1997) 47–57.
[48] L. Yang, k-Edge Connected neighborhood graph for geodesic distance estimation and nonlinear data projection, in: Proceedings of the 17th International Conference on Pattern Recognition, ICPR'04, 2004.
[49] L. Yang, Building k edge-disjoint spanning trees of minimum total length for isometric data embedding, IEEE Trans. Patt. Anal. Mach. Intell. 27 (2005) 1680–1683.
[50] A.C. Yao, An $O(|E|\log\log|V|)$ algorithm for finding minimum spanning trees, Inform. Process. Lett. 4 (1975) 21–23.
[51] C.T. Zahn, Graph-theoretical methods for detecting and describing gestalt clusters, IEEE Trans. Comp. C20 (1971) 68–86.
[52] C. Zhong, D. Miao, R. Wang, A graph-theoretical clustering method based on two rounds of minimum spanning trees, Patt. Recog. 43 (2010) 752–766.
[53] C. Zhong, D. Miao, P. Fränti, Minimum spanning tree based split-and-merge: a hierarchical clustering method, Inform. Sci. 181 (2011) 3397–3410.
[54] C. Zhong, M. Malinen, D. Miao, P. Fränti, Fast approximate minimum spanning tree algorithm based on K-means, in: 15th International Conference on Computer Analysis of Images and Patterns, York, UK, 2013.