DISSERTATIONS IN
FORESTRY AND
NATURAL SCIENCES

Mikko Malinen

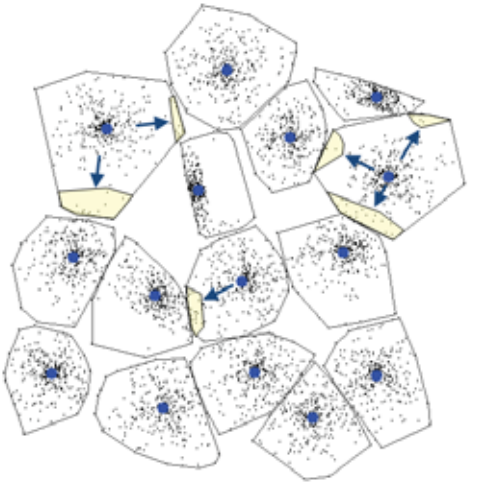
New Alternatives for k-Means Clustering

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

UNIVERSITY OF
EASTERN FINLAND

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.

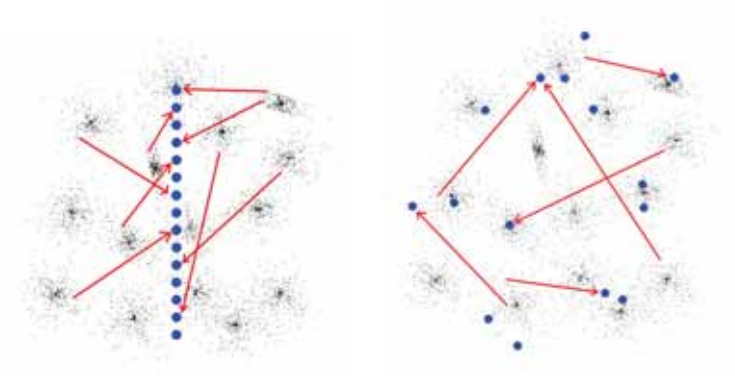


Figure 4.1: Original dataset and line init (left) or random init (right) with sample mappings shown by arrows.

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.

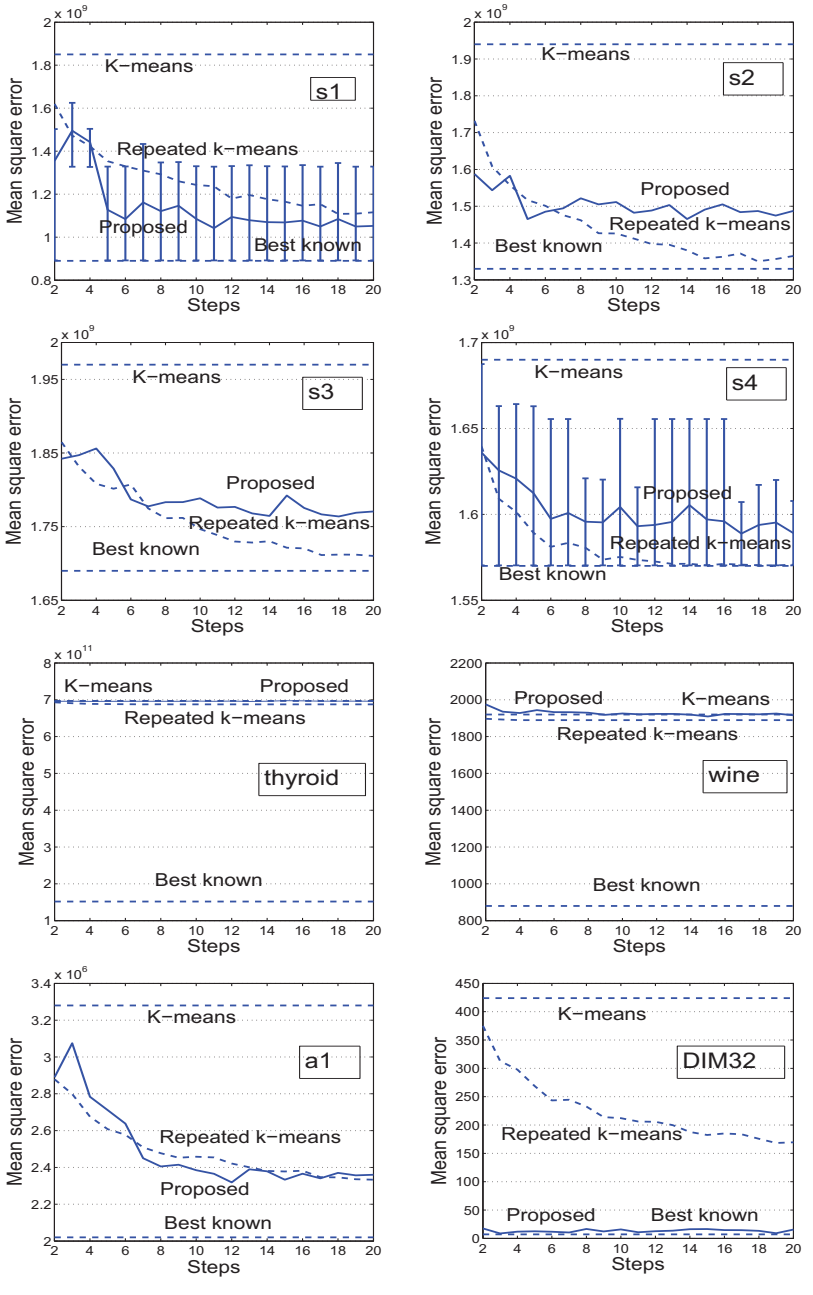


Figure 4.2: Results of k-means (average over 200 runs) for datasets s1, s2, s3, s4, thyroid, wine, a1 and DIM32 with different numbers of steps. For repeated k-means there are an equal number of repeats as there are steps in the proposed algorithm. For s1 and s4, the 75% error bounds are also shown.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

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

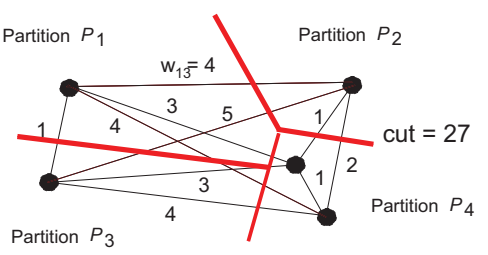
and

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

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.

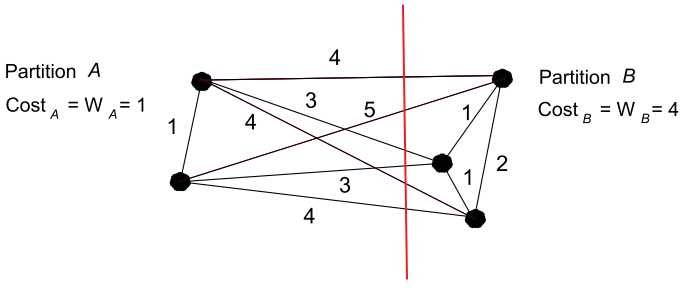


Figure 5.3: Calculation of the cost. Edge weights are squared Euclidean distances.

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

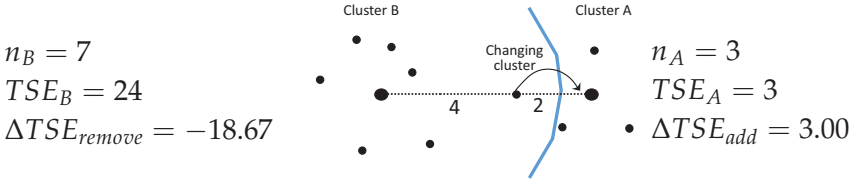


Figure 5.4: Changing point from cluster B to A decreasing cost by 121.02.

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

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

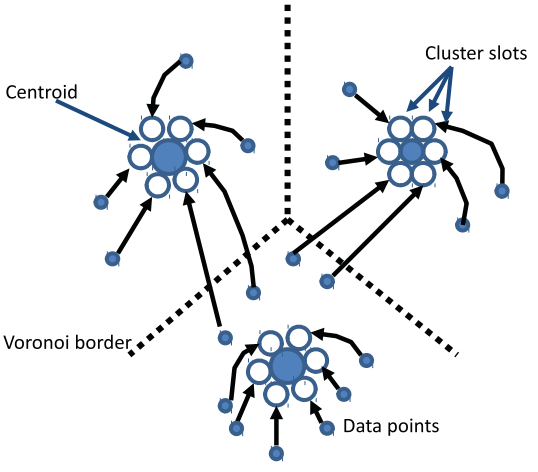


Figure 6.1: Assigning points to centroids via cluster slots.

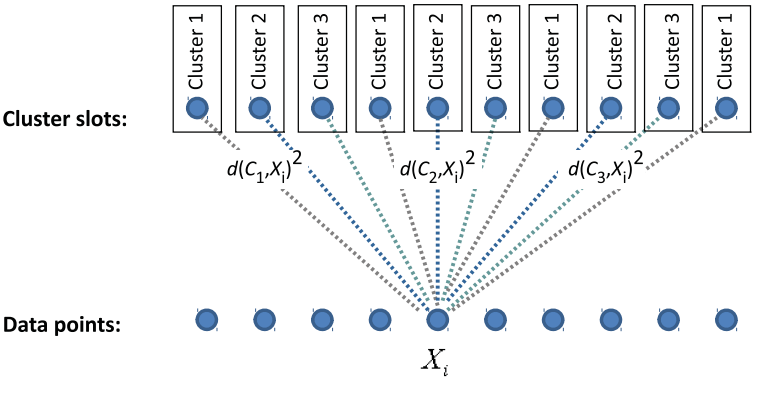


Figure 6.2: Minimum MSE calculation with balanced clusters. Modeling with bipartite graph.

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**

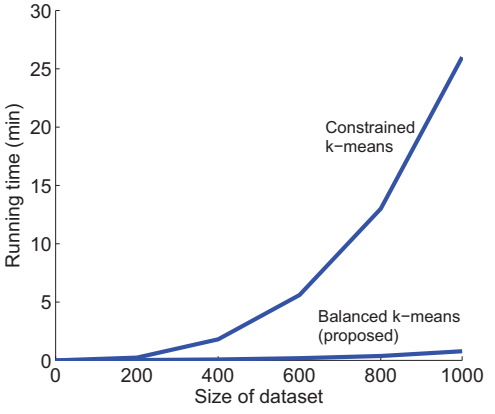


Figure 6.3: Running time with different-sized subsets of s1 dataset.

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.

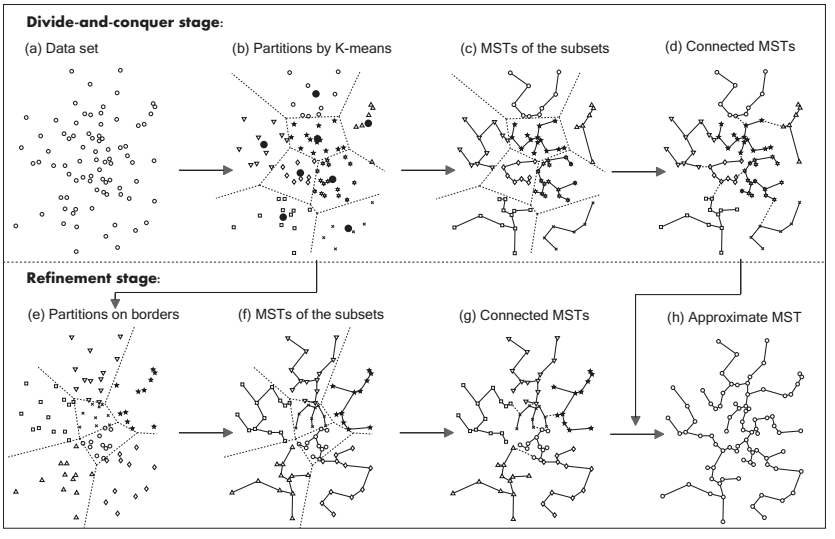


Figure 7.1: Phases of FMST algorithm.

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.

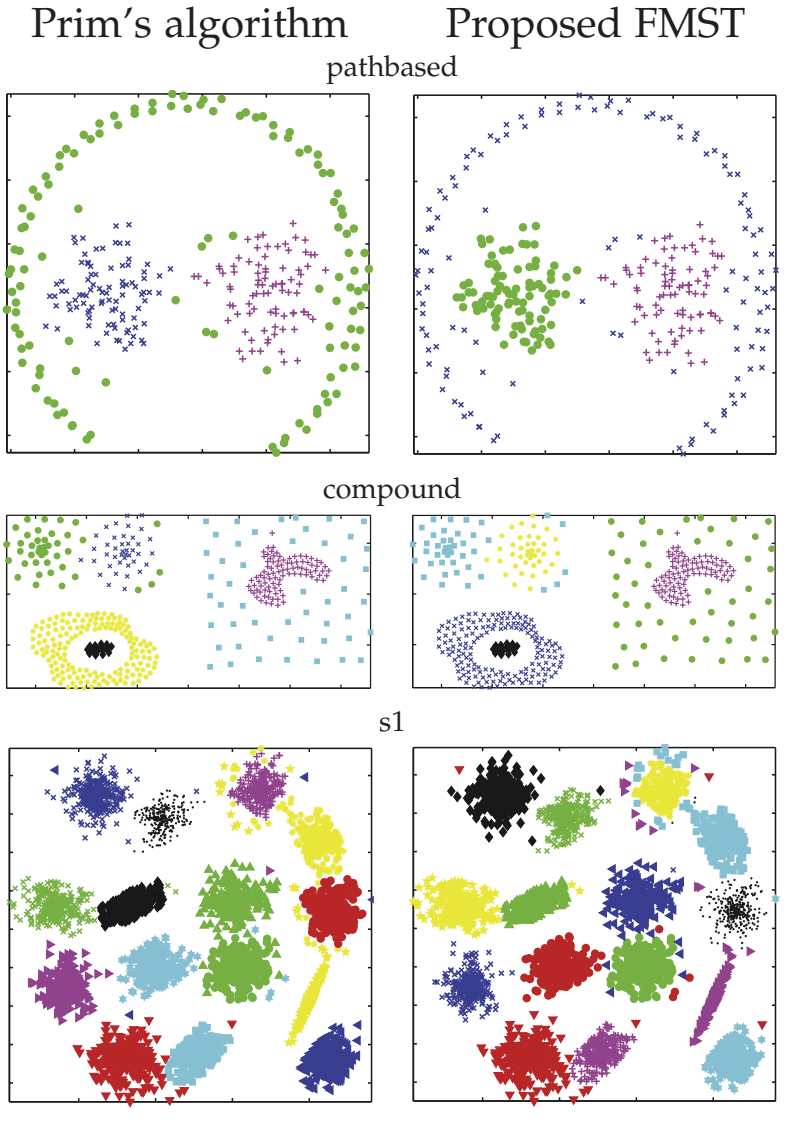


Figure 7.2: Prim's algorithm (left) and the proposed FMST based (right) clustering results for datasets pathbased (top), compound (middle) and s1 (bottom).

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

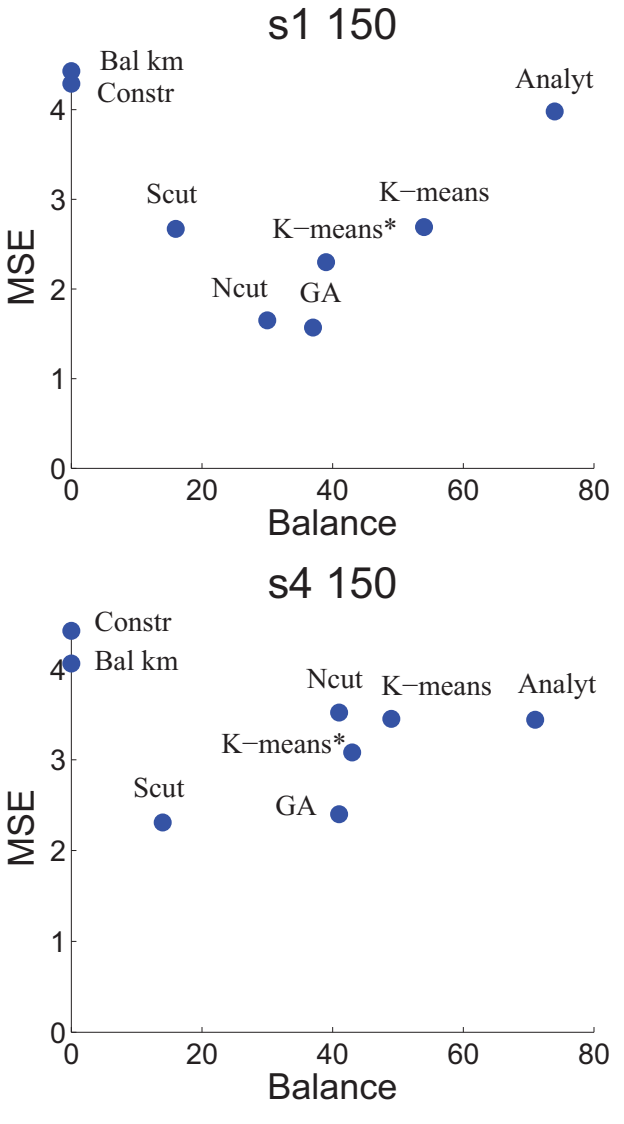


Figure 9.1: MSE *vs. balance for different methods. Means of 100 runs.*

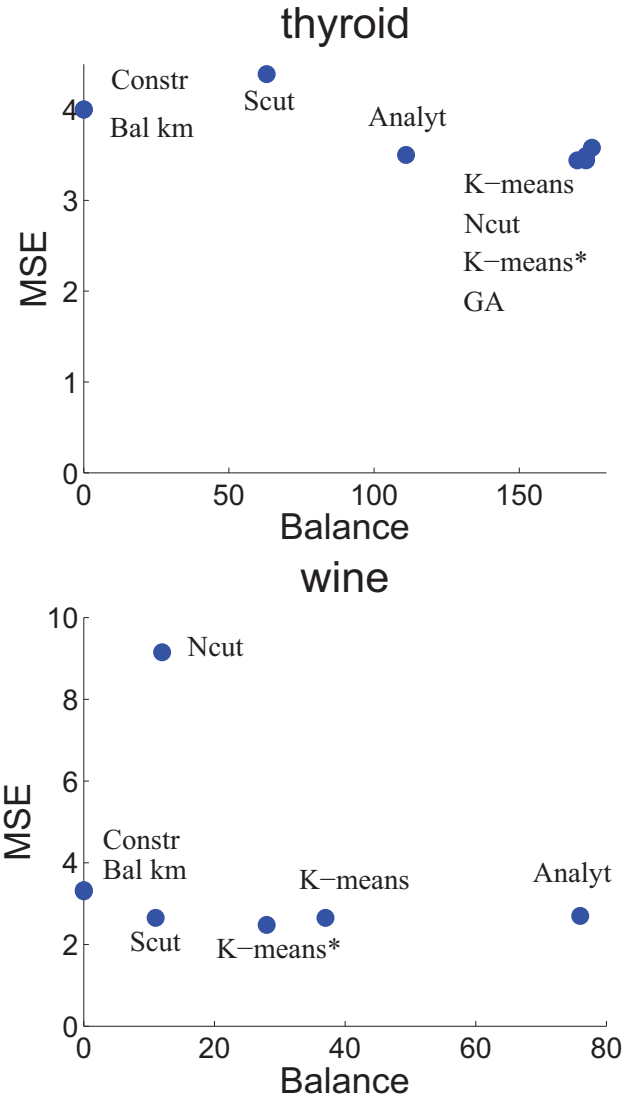


Figure 9.2: MSE *vs. balance for different methods. Means of 100 runs.*

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).

Mikko Malinen

New Alternatives for k-Means Clustering

This thesis presents mean square error of clustering as an analytic function and introduces a novel k-means based clustering algorithm. This thesis shows that all-pairwise squared distances as cost function tends to balance clusters and gives new algorithms for it. It introduces balanced k-means algorithm, which gives exactly balanced clusters while minimizing mean square error. Overall, this thesis introduces new clustering algorithms, which are comparable to the k-means algorithm or which serve some special purpose like balanced clustering.

UNIVERSITY OF
EASTERN FINLAND

Publications of the University of Eastern Finland
Dissertations in Forestry and Natural Sciences

ISBN 978-952-61-1788-1