Applied Computing and Intelligence

http://www.aimsciences.org/journal/aimsaci

Applied Intelligence and Computing
3(1): 93–115
DOI: 10.3934/aci.2023006
Received: 09 December 2022
Revised: 14 March 2023
Accepted: 19 April 2023
Published: 15 May 2023

Research article

All-pairwise squared distances lead to more balanced clustering

Mikko I. Malinen*and Pasi Fränti

Machine Learning Unit, School of Computing, University of Eastern Finland, Box 111, FIN-80101 Joensuu, FINLAND; mmali@cs.uef.fi, franti@cs.uef.fi

* **Correspondence:** Email: mmali@cs.uef.fi.

Academic Editor: Chih-Cheng Hung

Abstract: In clustering, the cost function that is commonly used involves calculating all-pairwise squared distances. In this paper, we formulate the cost function using mean squared error and show that this leads to more balanced clustering compared to centroid-based distance functions, like the sum of squared distances in *k*-means. The clustering method has been formulated as a cut-based approach, more intuitively called *Squared cut (Scut)*. We introduce an algorithm for the problem which is faster than the existing one based on the Stirling approximation. Our algorithm is a sequential variant of a local search algorithm. We show by experiments that the proposed approach provides better overall optimization of both mean squared error and cluster balance compared to existing methods.

Keywords: clustering; balanced clustering; squared cut; scut; max *k*-cut problem

1. Introduction

Agglomerative clustering is the second most popular clustering algorithm after k-means. It operates by a series of local optimal merge operations until the desired number of clusters is reached. Classical choices for the merge cost function include *single link, complete link* and *average link* but none of these are particularly good for most data. *Ward's method* [1] is a much better choice, and it has been used to minimize the same *total squared error* (TSE) cost function as *k*-means but with better optimization of the cost function.

Agglomerative clustering also has another advantage over *k*-means as there is a variant that does not require calculating the centroids: *all pairwise distances* (APD) within the clusters. This can be very handy in cases where the mean of the data cannot be easily calculated. These can include strings [2], time series [3] or GPS trajectories [4]. Furthermore, it has been shown that APD is just a scaled variant of TSE when using squared Euclidean distance.

Our first contribution is to show that APD will lead to more balanced clustering than TSE. This is important because, while the balance property can be a desired property in some applications, it can also cause the detection of real clusters to fail when some of the clusters are much larger (or smaller) than others. In [5], they wanted all clusters to be roughly the same size to make results easier to analyze but in [6] the balanced tendency caused a smaller country (Iceland) to merge into a larger country, thereby failing to detect the real clusters.

Our second contribution is to show that APD can also be formulated as a cut-based clustering. We give alternate proof for Hyugens' theorem and show that the APD of a cluster is equal to its TSE multiplied by its size (APD = n·TSE). We also propose a cut-based variant called *Squared Cut* (Scut) and present a fast sequential variant of k-means to minimize the cost function. This algorithm is significant due to the slowness of agglomerative algorithms, which are lower bound by $O(n^2)$ in their exact form.

The rest of the paper is organized as follows. In Section 2, we review the existing cost functions for traditional and balanced clustering. In Section 3, we present how clustering and cut-based methods are connected and present an alternative proof for Huygens' theorem. This theorem is important because it demonstrates the similarity between different representations of the APD cost function. In Section 4, we present a k-means-based algorithm for minimizing APD. In Section 5, Scut is extended to l_2^p k-clustering. Experiments and conclusions are then given in Sections 6 and 7, respectively.

2. Cost functions for traditional and balanced clustering

In this section, we introduce different cost functions and algorithms for traditional and balanced clustering. The choice of algorithm depends on the selected cost function, which is an important factor. By far the most common cost function in clustering is the total squared error (*TSE*), or equivalently, the mean squared error (*MSE*):

$$TSE = \sum_{i=1}^{n} \|x_i - C_{P_i}\|_2^2 \tag{1}$$

$$MSE = \frac{TSE}{n} \tag{2}$$

where x_i is a data point, C_j is the centroid of cluster j and P_i is the label of the cluster which x_i is assigned to. These can be generalized to other p-norms, even to infinite norms ($p = \infty$). Reference [7] discussed how the infinity norm criterion could be used in practice. Instead of taking the maximum, the infinity norm was approximated by a p-norm with a high value of p. This is because a p-norm has an analytic formulation that can then be optimized.

A balancing constraint can be given along with the cost function [8],

$$n_j = \lfloor \frac{n}{k} \rfloor \text{ or } n_j = \lceil \frac{n}{k} \rceil \quad \forall j = 1..k, \tag{3}$$

where n_j denotes the size of the cluster j.

2.1. k-Means clustering

The Euclidean sum-of-squares clustering (k-means clustering) aims at finding the best grouping of n points in k clusters and it has been shown to be an NP-hard problem [9]. The clusters are represented

by *center points (centroids)*, and the aim is to minimize the *mean squared error* (*MSE*), calculated as the mean distance of the points from their nearest centroid, or, equivalently, to minimize the *total squared error (TSE)*, calculated as the sum of the squared distances of the points from their centroids. *k*-Means clustering minimizes

$$TSE = TSE_1 + TSE_2 + ... + TSE_k, \tag{4}$$

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

$$\text{TSE} = n_1 \cdot MSE_1 + n_2 \cdot MSE_2 + ... + n_k \cdot MSE_k, \tag{5}$$

where n_j is the number of points in cluster j and MSE_j is the mean squared error of the jth cluster. The *TSE* and *MSE* for a single cluster j are calculated as

$$TSE_j = \sum_{x_i \in P_j} ||x_i - C_j||^2 \tag{6}$$

$$MSE_j = \frac{TSE_j}{n_j}. \tag{7}$$

When k and dimensionality d are constant, the k-means clustering problem (minimizing *TSE*) can be solved in polynomial $O(n^{kd+1})$ time [10]. Although polynomial, this is still slow, and suboptimal algorithms are therefore used. The k-means algorithm [11] is fast and simple, although its worst-case number of iterations is $O(n^{kd})$. The advantage of k-means is that it finds a local optimum starting from any initial centroid location by a simple iterative two-step procedure. A drawback of k-means is that it cannot always find the global optimum. This is one of the reasons why slower agglomerative clustering [12–14], or more complex variants of k-means [15–17], is used.

Gaussian mixture models (EM algorithm) [18, 19] and cut-based methods [20, 21] have also been used. Recent research has considered clustering using analytic functions [7] and fitting the data into the model before fitting the model to the data [22].

2.2. Balanced clustering

Balanced clustering is defined as a type of clustering where the points are evenly distributed between the clusters. In other words, every cluster includes either $\lfloor n/k \rfloor$ or $\lceil n/k \rceil$ points. We define a balanced clustering problem as a problem that aims at minimizing the imbalance, while also minimizing some other cost function, such as *MSE*. The imbalance is measured by

$$\text{imbalance} = \sum_j \max(n_j - \lceil \frac{n}{k} \rceil, \lfloor \frac{n}{k} \rfloor - n_j). \tag{8}$$

There is a need for balance, it is useful in workload balancing, wireless sensor networks and sightseeing tour design. In traveling salesman problem a shortest route for the salesman through all the cities. For example, in [8] a clustering algorithm is applied to the multiple traveling salesman problem [23]. The algorithm clusters the cities so that each cluster can be solved by one salesman. The goal is to achieve an equal workload among salesmen. The following is an example of partitioning a dataset for 4-TSP, that is, there are four salesmen in Figure 1 [24].

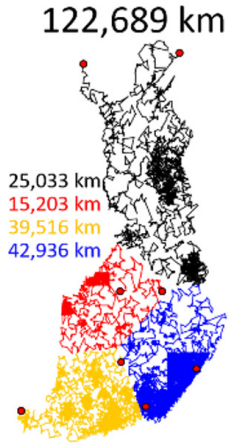


Figure 1. Balanced clustering [24] leads to better parallel efficiency when solving large-scale TSP problems. Subsolutions of a 4-TSP problem of the Santa Claus data in [24] are shown.

Clustering is used in wireless sensor networks for the energy efficiency of nodes and the lifetime of the network [25, 26]. Targeting certain sizes of clusters is also useful [27, 28].

Balanced clustering, in general, is a two-objective optimization problem, in which the two aims can be in conflict with each other: minimize a cost function such as *MSE*, and balance the cluster sizes at the same time. Traditional clustering aims at minimizing the *MSE* completely without considering cluster size balance. Balancing, on the other hand, would be trivial if we did not care about *MSE*, which involves dividing the vectors into equal size clusters randomly. To optimize 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 some applications, cluster size balance is not a mandatory requirement. It is enough that cluster sizes are more balanced. In balance-driven clustering, balanced clustering is an aim, but it is not mandatory. It is a compromise between the two goals, namely the balance and the *MSE*. The solution is a weighted cost function between the *MSE* and the imbalance, or it is a heuristic, which aims at minimizing *MSE* but indirectly creates a more balanced result than optimizing *MSE* alone.

Next, we review the existing methods that aim to achieve balanced clustering. Bradley et al. [29] presented a constrained k-means algorithm where the assignment step of k-means is implemented as a linear programming problem in which a minimum number of points in a cluster is set as a constraint. In [8], a balanced k-means algorithm using the Hungarian algorithm and fixed-size clusters is presented. It solves the k-means assignment step as an assignment problem. It is a balance-constrained algorithm that forces balanced clusters.

The method in [30] tried to find a partition close to a given partition, but so that the cluster size constraints were fulfilled. Banerjee and Ghosh [31] presented an algorithm based on frequency sensitive competitive learning (FSCL) where the centroids competed for the points. It multiplicatively scaled the error (the distance from the data point to the centroid) by the number of times that a centroid had won in the past. This meant that larger clusters were less likely to gain points in the future. Althoff et al. [32] used FSCL, but their solution incorporated an additive distance bias instead of a multiplicative distance bias. They reported that their algorithm was more stable for high-dimensional

feature spaces.

Banerjee and Ghosh [33] introduced a fast ($O(kn \log n)$) algorithm for balanced clustering that used 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 [34] is defined as the sum of the inter-cluster similarity and a regularization term measuring the relative size of two clusters. In [35], there was a term included in the cost function, which aimed at facilitating the balancing. In [36], the size of the maximum-size cluster was minimized in a k-means-type algorithm.

There are also application-based solutions in networking [27] that aim to balance network loads. With these solutions, the clustering is done through self-organization without central control. In [28], energy-balanced routing between sensors was the goal; thus, only the most suitably balanced number of nodes were cluster members. The classification of some algorithms into these two classes and the types of the algorithms can be found in Table 1.

Table 1. Classification of some balanced clustering algorithms.

Balance-constrained	Type of algorithm
Balanced k-means [8]	k-means
Balanced size constraints [37]	linear programming
Constrained k-means [29]	k-means
Size constrained [30]	integer linear programming
Balance-driven	**Type of algorithm**
FSCL [31]	assignment
FSCL additive bias [32]	assignment
Cluster sampled data [33]	k-means
Ratio cut [38] (spectral)	divisive
Ncut [20] (spectral)	divisive
Mcut [21]	divisive
SRcut [34]	divisive
Submodular fractional programming [35]	submodular fractional programming
MinMax k-means [36]	k-means

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. The partitioning of a dataset into two parts A and B can be indicated as cut(A, B), while the value of cut(A, B) is the total weight of all pairs of points where one is from part A and the other is from part B:

$$\text{cut}(A, B) = \sum_{x_l \in A, x_s \in B} w_{ls}. \tag{9}$$

The weights w can be defined either as distances or similarities between the two points (see a summary of their use in Figure 2). Unless otherwise noted, we use (squared) Euclidean distances in this paper.

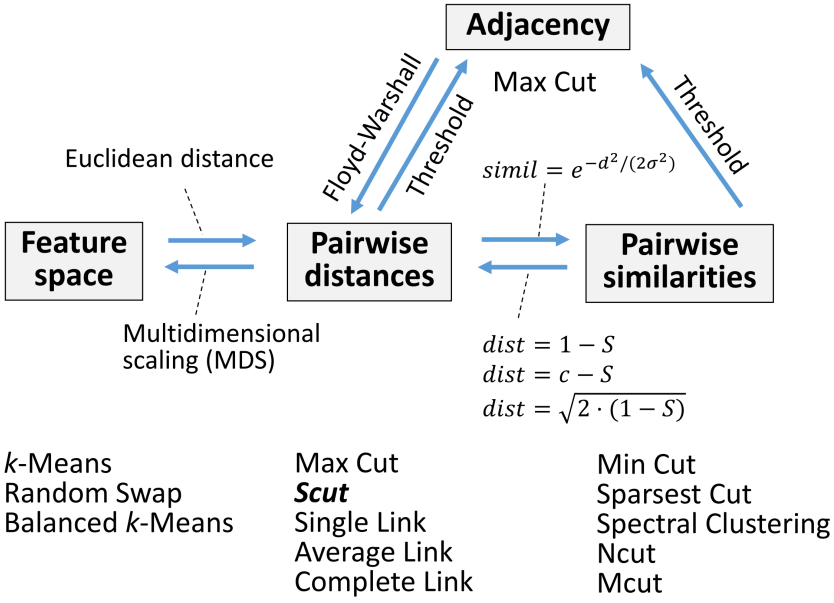


Figure 2. Data representation types, conversions between them and methods and algorithms to cluster each of them.

The cut(A, B) equals the total pairwise weights of $A \cup B$ subtracted by the pairwise weights within the parts A and B:

$$\text{cut}(A, B) = W - W(A) - W(B), \tag{10}$$

where W is the sum of all pairwise weights (here we assume that arc lengths are symmetric).

$$W = \frac{1}{2} \sum_{l \neq s} w_{ls}, \tag{11}$$

where $W(A)$ is the sum of the pairwise weights within part A:

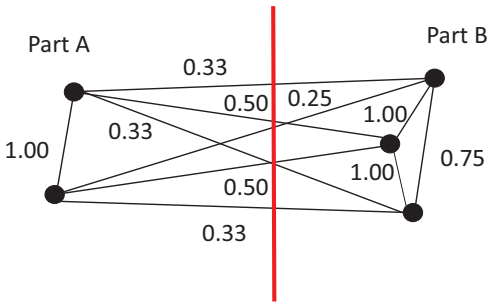
$$W(A) = \frac{1}{2} \sum_{x_l \in A, x_s \in A, l \neq s} w_{ls}, \tag{12}$$

where $W(B)$ is defined similarly.

In cut-based clustering, the most common objective (cost or utility) functions are *Minimum cut* [39], *Maximum cut* [40], *Sparsest cut* [41], *Ratio cut* [38] and *Normalized cut, Ncut,* [20]. In Minimum cut, the points are partitioned into two parts, so that the cut is minimized. In Maximum cut, the cut is maximized. In Sparsest cut, the value of the cut is divided by the minimum of the number of points in the resulting parts. In Ratio cut, the cost of a cut is normalized by the number of points n_A or n_B, while in Ncut, the cut is normalized by the similarities to all the other points in the dataset. These normalizations favor balanced cuts [42], p. 401. In *Mcut* [21], the cost function aims at minimizing cut(A,B), the similarity of A and B, while maximizing the similarities within the parts ($W(A)$ and $W(B)$) at the same time. Mcut tends to make balanced clusters, even when compared to Ratio cut and Mcut [21]. A summary of these methods is in Table 2 and an example in Figure 3.

Table 2. Summary of some cut-based methods.

Method	min or max	Cost	Weights	Balanced
Minimum cut [39]	min	weight	similarity	no
Maximum cut [43]	max	weight or number of edges	dissimilarity	no
Sparsest cut [41]	min	number of edges	adjacency	no
Ratio cut [38]	min	weights	similarity	yes
Ncut [20]	min	weights	similarity	yes
Mcut [21]	min	weights	similarity	yes



$W = 5.99$
$W(A) = 1$
$W(B) = 2.75$
$\text{cut}(A, B) = 2.24$
$\text{RatioCut}(A, B) \approx 1.87$
$\text{Ncut}(A, B) \approx 1.14$
$\text{Mcut}(A, B) \approx 3.05$

Figure 3. An example of a cut.

The methods optimize the following definitions:

$$\text{MinimumCut}(A, B) = \min_{A,\bar{A}} \text{cut}(A, \bar{A}) \tag{13}$$

$$\text{MaximumCut}(A, B) = \max_{A,\bar{A}} \text{cut}(A, \bar{A}) \tag{14}$$

$$\text{SparsestCut}(A, B) = \frac{\text{cut}(A, \bar{A})}{\min(|A|, |\bar{A}|)} \tag{15}$$

$$\text{RatioCut}(A, B) = \frac{\text{cut}(A, \bar{A})}{n_A} + \frac{\text{cut}(B, \bar{B})}{n_B} \tag{16}$$

$$\text{Ncut}(A, B) = \frac{\text{cut}(A, \bar{A})}{W(A) + \text{cut}(A, \bar{A})} + \frac{\text{cut}(B, \bar{B})}{W(B) + \text{cut}(B, \bar{B})} \tag{17}$$

$$\text{Mcut}(A, B) = \frac{\text{cut}(A, B)}{W(A)} + \frac{\text{cut}(A, B)}{W(B)}. \tag{18}$$

where $\bar{A}$ is the complement point set of A, $\bar{B}$ is the complement point set of B and $W(A)$ is the total similarities between the pairs of points within cluster A. Optimizing the cost functions (16) and (17) aims at minimizing the cuts (the numerators), while at the same time maximizing the denominators. Often one approximates this problem through relaxation, i.e., solving a nearby easier problem. Relaxing Ncut leads to normalized spectral clustering, while relaxing RatioCut leads to unnormalized spectral clustering [42]. There is also a relaxation based on semidefinite programming for Ncut [44]. The methods Ncut, Ratio cut and Mcut, in their basic forms, cut the graph into two

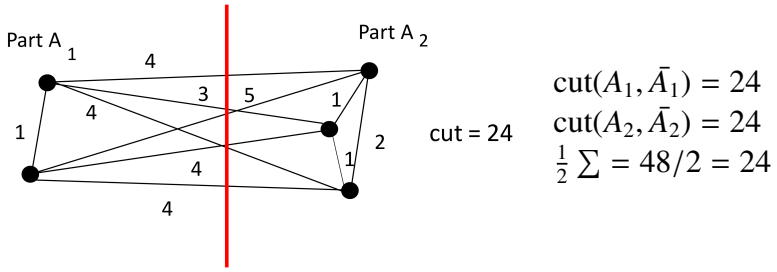


Figure 4. An example of MAX k-CUT, when $k = 2$.

parts. Although extensions exist, which recursively cut the graph into k parts, $k > 2$, these methods do not consider cutting the graph simultaneously into more than two parts. In our method, which we will introduce in the next sections, cutting into k parts is a standard procedure.

3.1. MAX k-CUT method

In the weighted MAX k-CUT problem [45] one partitions a graph into k subgraphs so that the sum of the weights of the edges between the subgraphs is maximized. The weights are the distances. MAX k-CUT aims at partitioning the data into k clusters $A_1, ..., A_k$. Following the notation of Section 3 and writing the factor 1/2 in order to avoid summing the weights twice, the MAX k-CUT problem is defined as

$$\max_{A_j, 1 \le j \le k} \frac{1}{2} \sum_{j=1}^{k} \text{cut}(A_j, \bar{A}_j). \tag{19}$$

See an example of MAX k-CUT in Figure 4. This is an NP-hard problem [46] for general weights. If we use Euclidean distance for the weights, then weighted MAX k-CUT results in the minimum intra-cluster pairwise Euclidean distances among any k-CUT. This is due to the fact that when we maximize what is taken off, we, at the same time, minimize what is left. If we use the *squared* Euclidean distances, we end up minimizing intra-cluster pairwise squared Euclidean distances, see Equation 20. With squared Euclidean distances as the weights, the problem is expected to remain NP-hard. NP-hardness is proved for two cluster cases [47].

3.2. Scut

In this paper, we formulate all-pairwise squared distances using a cut-based method called *Squared cut (Scut)*. This method is called l_2^2 k-clustering [48]. It is a special case of APD that uses squared Euclidean distances. However, we formulate it by using the *TSE*s and *MSE*s of the clusters and show that the method leads to more balanced solutions for 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 the MAX k-CUT method [49]. We present an algorithm for the problem that is more practical than the exhaustive Stirling search proposed in [50] for l_2^2 k-clustering. The algorithm is a sequential variant of the k-means algorithm, with immediate updates directly optimizing the cost function.

A general k-clustering problem in [49] defined the cost by calculating all pairwise distances within the clusters for an arbitrary weighted graph. The paper [51] studied the problem when the distances

Figure 5. Two different-sized clusters with the same MSE.

satisfied the triangle inequality. Schulman [48] gave probabilistic algorithms for l_2^2 k-clustering. The running time was linear if the dimensionality $d = o(\log n / \log\log n)$ but otherwise it was $n^{O(\log\log n)}$. De la Vega et al. [50] improved and extended Schulman's results, giving a true polynomial time approximation algorithm for arbitrary dimensions. However, even their algorithm was slow in practice. We, therefore, present a faster algorithm for the Squared cut method.

In Scut, we form the graph by assigning the 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 = $n_j \cdot TSE_j$. The generalization of this to all clusters is known as Huygens' theorem. Huygens' theorem is crucial for our method because it relates the pairwise distances to the intra-clusters *TSE*, and thus, to the Scut cost function:

$$\text{Scut} = \sum_{j=1}^{k} \sum_{x_s, x_t \in P_j t > s} \|x_s - x_t\|_2^2 = n_1 \cdot TSE_1 + n_2 \cdot TSE_2 + ... + n_k \cdot TSE_k, \tag{20}$$

where k is the number of clusters, x_s and x_t are point numbers s and t in part P_j, n_j is the number of points and TSE_j is the total squared error of the jth cluster. Note that this Scut cost tells what there is inside the clusters, not what there is between the clusters. Based on (6), this may also be written as

$$\text{Scut} = n_1^2 \cdot MSE_1 + n_2^2 \cdot MSE_2 + ... + n_k^2 \cdot MSE_k, \tag{21}$$

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

$$\text{Scut} = W - \max_{A_j, 1 \le j \le k} \frac{1}{2} \sum_{j=1}^{k} \text{cut}(A_j, \bar{A}_j), \tag{22}$$

where the A_js are subsets of the dataset. From this, we conclude that using squared Euclidean distances as weights and optimizing MAX k-CUT results in optimization of the Scut cost function (20). The difference between the MAX k-CUT used in calculating Scut cost and MAX k-CUT in general is the choice of the weights. Our cut-based method has an *MSE*-based cost function and tends to balance clusters because of the n_j^2 factors in (21). This can be seen by the following simple example, where we assume that the two clusters have the same mean squared error: $MSE_1 = MSE_2 = MSE$ (Figure 5). Their total errors are $2^2 \cdot MSE_1 = 4 \cdot MSE$ and $10^2 \cdot MSE_2 = 100 \cdot MSE$. Adding one more point increases the error by $(n+1)^2 \cdot MSE - n^2 \cdot MSE = (2n+1) \cdot MSE$. In Figure 5, the cost would increase by $5 \cdot MSE$ if the point is added into cluster 1 and $21 \cdot MSE$ if into cluster 2. The

cost function therefore always favors putting points into a smaller cluster which tends to make more balanced clusters. The cost function leads to more balanced clustering in general. This is supported by logical argument (Figure 5) and experimental evidence. We do not have proof that this will happen in all cases. Due to the suboptimality of k-means, it is possible that different cost functions would lead to different suboptimal solutions. The suboptimal solution obtained by TSE might be more balanced in certain situations. However, we expect Scut to provide more balanced results than TSE in most cases. Demonstration of the calculation of the cost can be found in Figures 6 and 7.

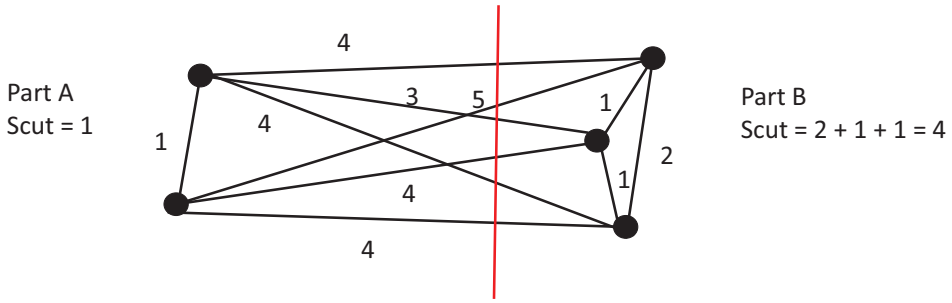


Figure 6. Calculation of the cost. Edge weights are squared Euclidean distances.

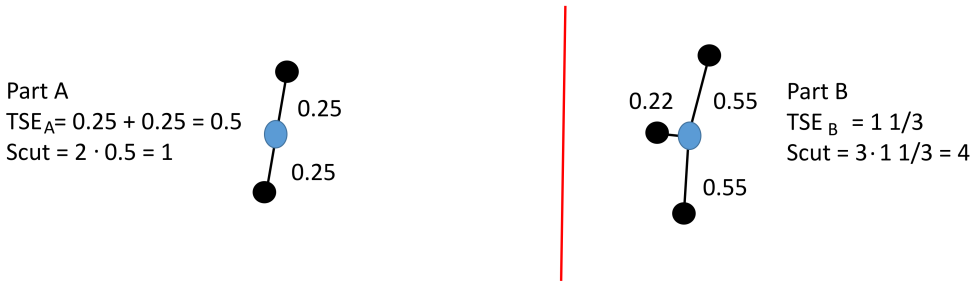


Figure 7. Calculation of the cost by TSEs. Edge weights are squared Euclidean distances. The blue dots are centroids.

3.3. Proof of Huygens' theorem

We provide here an alternative, simpler and new proof of Huygens' theorem. An earlier proof can be found in Späth's book [52], p. 52. The name Huygens is mentioned in connection with this theorem in [9]. Our proof is based on differentiation:

By x_{vt} we denote the point v, feature t and by Scut_i we denote the pairwise intra-cluster squared distances. For a single cluster i,

$$TSE_i = \sum_u (\sum_t ((x_{ut} - \frac{\sum_v x_{vt}}{n_i})^2)),$$

$$\text{Scut}_i = \sum_{u=1}^{n_i} \sum_{v=u+1}^{n_i} \sum_t (x_{ut} - x_{vt})^2.$$

Huygens' theorem for one cluster i is

$$\text{Scut}_i = n_i \cdot TSE_i.$$

$\Leftrightarrow$

$$\sum_{u=1}^{n_i} \sum_{v=u+1}^{n_i} \sum_{t} (x_{ut} - x_{vt})^2$$

$$= n_i \cdot \sum_{u} (\sum_{t} ((x_{ut} - \frac{\sum_v x_{vt}}{n_i})^2)).$$

Our proof of Huygens' theorem is achieved by differentiating TSE_i and Scut_i with respect to x_{st}, the feature t of point s and by showing that these differ only by the factor n_i. Both of the differentiations use the chain rule:

$$\frac{\partial TSE_i}{\partial x_{st}} = 2 \sum_{u \neq s} (x_{ut} - \frac{\sum_v x_{vt}}{n_i}) \cdot -\frac{1}{n_i} + 2(x_{st} - \frac{\sum_v x_{vt}}{n_i}) \cdot (1 - \frac{1}{n_i})$$

$$= 2 \sum_{u} (x_{ut} - \frac{\sum_v x_{vt}}{n_i}) \cdot -\frac{1}{n_i} + 2(x_{st} - \frac{\sum_v x_{vt}}{n_i})$$

$$= 2 \sum_{u} (x_{ut}) \cdot -\frac{1}{n_i} + 2 \cdot \frac{\sum_v x_{vt}}{n_i} + 2(x_{st} - \frac{\sum_v x_{vt}}{n_i})$$

$$= 2 \sum_{u} (x_{ut} \cdot -\frac{1}{n_i}) + 2x_{st}.$$

$$= 2 \cdot (x_{st} - \sum_{u} (x_{ut} \cdot \frac{1}{n_i}))$$

Next, we calculate the other derivative:

$$\frac{\partial \text{Scut}_i}{\partial x_{st}} = 2 \cdot \sum_{v=s+1}^{n_i} (x_{st} - x_{vt}) \cdot 1 + 2 \cdot \sum_{u=1}^{s-1} (x_{ut} - x_{st}) \cdot -1$$

$$= 2 \cdot ((n_i - (s+1) + 1) \cdot x_{st} + \sum_{v=s+1}^{n_i} (-x_{vt}) + \sum_{u=1}^{s-1} (-x_{ut}) + (s-1) \cdot x_{st})$$

$$= 2 \cdot ((n_i - s + s - 1) \cdot x_{st} + \sum_{u=1}^{n_i} (-x_{ut}) + x_{st})$$

$$= 2 \cdot (n_i \cdot x_{st} - \sum_{u} x_{ut})$$

Thus

$$\frac{\partial \text{Scut}_i}{\partial x_{st}} = n_i \cdot \frac{\partial TSE_i}{\partial x_{st}}. \tag{23}$$

In addition, we know that TSE_i and Scut_i are both equal to 0 when the points are at the same location as the centroid. Based on this and (23), we conclude that Huygens' theorem holds. □

Algorithm 1 Sequential k-means 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 ΔScut < 0 then
        move point i to cluster l
        update centroids and TSEs of previous cluster and cluster l
        changed ← TRUE
      end if
    end for
  end for
end while
Output partitioning of points P.
```

4. Approximating Scut

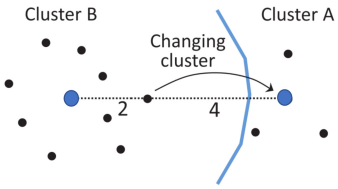
4.1. k-means variant for Scut

We next define the sequential k-means variant for the Scut method. In the algorithm, points are repeatedly re-partitioned to the cluster that provides the lowest value for the Scut cost function. The partitioning of the points is done one by one, and a change of cluster will cause an immediate update of the two affected 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 (20). We next derive a fast $O(1)$ update formula which calculates how much the value of the cost function changes when one 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 1. This may require repeating the process of going through the points multiple times before stopping. The approximation ratio is the same as in the subsequent Equation (30), where $\alpha_k > 1 - k^{-1}$, which is derived in [46]. The update formula follows the merged cost in the agglomerative clustering algorithm [12]. It includes the change of *TSE* when adding a point, the change of *TSE* when removing a point and the change of the overall cost with respect to the cost function (20).

$$\Delta TSE_{add} = \frac{n_A}{n_A + 1} \cdot \|C_A - x_i\|^2 \tag{24}$$

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

$n_B = 9$
$TSE_B = 25$
$\Delta TSE_{remove} = \frac{9}{9-1} \cdot 2^2 = 4.5$



$n_A = 3$
$TSE_A = 3$
$\Delta TSE_{add} = \frac{3}{3+1} \cdot 4^2 = 12$

Figure 8. Changing point from cluster B to A decreases Scut cost by 10. TSE increases, but because cluster A has many points fewer than cluster B, the Scut cost decreases.

$$= \frac{n_B}{n_B - 1} \cdot \|C_B - x_i\|^2 \tag{25}$$

The total cost before the move with respect to the two clusters is

$$\text{Scut}_{before} = n_A \cdot TSE_A + n_B \cdot TSE_B, \tag{26}$$

where n_A and n_B are the number of points in the clusters A and B. C_A and C_B are the centroid locations, and x_i is the data point involved in the operation. The total cost after the move is

$$\text{Scut}_{after} = (n_A + 1) \cdot (TSE_A + \Delta TSE_{add}) + (n_B - 1) \cdot (TSE_B - \Delta TSE_{remove}) \tag{27}$$

From these we get the change in cost:

$$\Delta\text{Scut} = \text{Scut}_{after} - \text{Scut}_{before} \tag{28}$$

$$= TSE_A - TSE_B + (n_A + 1) \cdot \Delta TSE_{add} - (n_B - 1) \cdot \Delta TSE_{remove}. \tag{29}$$

See an example of a point changing its cluster in Figure 8, where the changes in the *TSE*s are $\Delta TSE_{add} = 12$ and $\Delta TSE_{remove} = 4.5$. The change in the cost function $\Delta\text{Scut} = -10$.

4.2. Approximation ratio

We can provide an approximation ratio ϵ_k for Scut using the known approximation ratio $\alpha_k = 1 - k^{-1}$ for MAX k-CUT [46]. The approximation ratio of MAX k-CUT is calculated using the sum of pairwise weights that were cut off (MAX k-CUT) while Scut uses the sum of remaining weights, see Figure 9. We can derive ϵ_k from α_k for the expected case as follows:

$$\epsilon_k = \frac{\text{Scut}}{\text{Scut}^*} \tag{30}$$

$$= \frac{W - \texttt{MAX} \quad k - \texttt{CUT}}{W - \texttt{MAX} \quad k - \texttt{CUT}^*} \tag{31}$$

$$= \frac{W - \texttt{MAX} \quad k - \texttt{CUT}}{W - \frac{1}{\alpha_k} \cdot \texttt{MAX} \quad k - \texttt{CUT}} \tag{32}$$

$$< \frac{W - \texttt{MAX} \quad k - \texttt{CUT}}{W - \frac{1}{1-k^{-1}} \cdot \texttt{MAX} \quad k - \texttt{CUT}} \tag{33}$$

Since we have a lower bound for α_k, it is possible to get an upper bound for ϵ_k. However, this bound is dataset-specific and also depends on the number of clusters (k). In practice, the denominator of (30) becomes very small in all of the cases we tested, so the upper bound does not have much practical relevance.

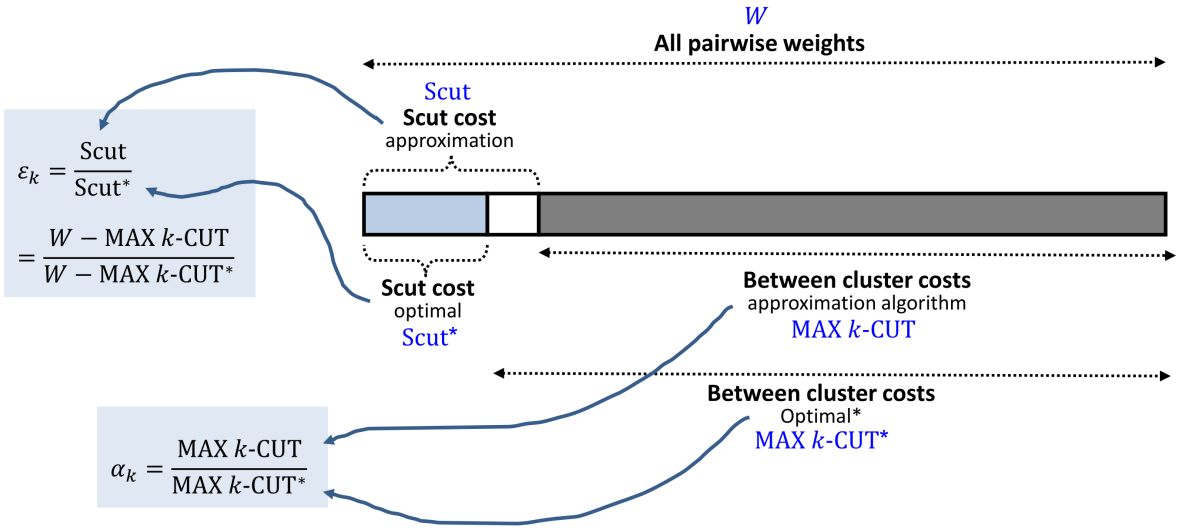


Figure 9. Derivation of the approximation ratio.

5. l_2^p k-Clustering

A generalization of the Scut cost function (20) is l_2^p k-Clustering, i.e., all-pairwise powered distances as cost. So far, in this article, we have dealt with the case where $p = 2$, but p can have other values, like $p = 1$ or $p = 3$. It is a known fact that, in centroid-based clustering, when using absolute values in distance calculations (this corresponds to $p = 1$ here), the outlier points tend not to be put in one-point clusters compared to when squared distances are used. Thus we believe that here $p = 1$ tends to make less-balanced clusters and $p > 2$ tends to make more balanced clusters than the more common $p = 2$. As an extreme, if we use $p = \infty$, we will get clusters where the points of point pairs with the largest pairwise distances are in different clusters. We show that the relation corresponding to Huygens' theorem does not hold for this generalized cost function:

Theorem. Huygens' theorem does not hold for powers higher than two because there exists a dataset X for which the following holds:

$$\sum_{x_s, x_t \in P_j} \|x_s - x_t\|^p \neq n_j \cdot \Big(\sum_{x_s \in P_j} \|x_s - \frac{\sum_{x_v \in P_j} x_v}{n_j}\|^p\Big), \qquad p > 2. \tag{34}$$

Proof. Consider a dataset of two points x_s and x_t, located at a distance of 1 from each other. Then

$$\|x_s - x_t\|^p > \|x_s - \frac{x_s + x_t}{2}\|^p + \|x_t - \frac{x_s + x_t}{2}\|^p, \qquad \text{when} \qquad p > 2,$$

i.e.,

$$1 > 2 \cdot (0.5^p + 0.5^p), \qquad \text{when} \qquad p > 2.$$

So we have found an X for Equation 34. □

6. Experiments

We use datasets from the Machine Learning Unit in the School of Computing, University of Eastern Finland [53]. The s-sets are synthetic, two-dimensional (2-d) and have increasingly overlapping clusters with the postfix number, iris is a real dataset.

The benefits of balancing clustering algorithms are best seen when clustering is done for non-balanced datasets. To compare how close the obtained clustering is to balance-constrained clustering having equal distribution of sizes $\lceil n/k \rceil$ and $\lfloor n/k \rfloor$, we measure the imbalance by calculating the difference in the cluster sizes and a balanced n/k distribution using Equation (8) with 30 repeats.

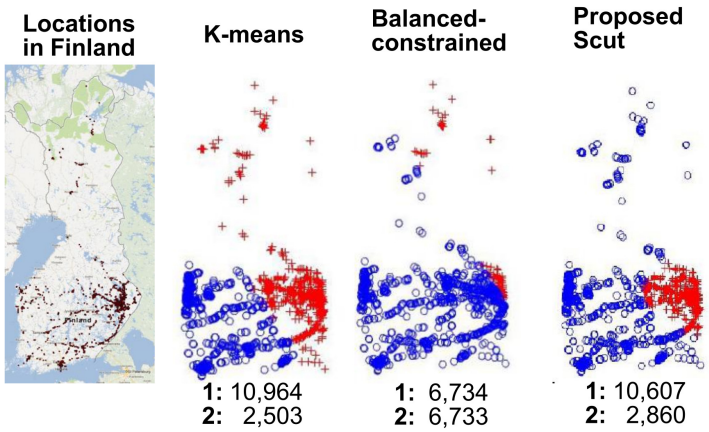


Figure 10. k-Means leads to a large imbalance in cluster sizes whereas balance-constrained clustering optimizes the distances poorly. Scut provides a good compromise between these two objectives.

The results in Table 3 show that the fast Scut algorithm outperforms k-means when compared to the Scut cost function (20) even with the balanced datasets, for which one could have expected k-means to perform equally well. We use the centroid index (CI) to measure the success at cluster level [54, 55] (S-sets). Given a ground truth solution (G) and clustering solution (C), the centroid index counts how many real clusters are missing a center, or alternatively, how many clusters have too many centers. The CI-value is the higher of the two numbers [54]. When CI = 0, it means that the clustering is correct. The Tables 3 and 4 have a line indicating how many times the numbers in columns are best. Of course, these indications do not have statistical meaning (not used in a formal statistical test), but these are just indications.

Table 3. Means of Scut cost, means of MSE and means of CI validity indices of the proposed Scut and k-means clustering for 30 runs. 'MSE' and 'Scut' and 'CI' on the top of the table refer to cost functions and 'k-means' and 'Scut' on the next level refer to algorithms.

Dataset	MSE		Scut		CI	
	K-means	Scut	K-means	Scut	K-means	Scut
s1	2.73	**2.39**	6.24	**4.62**	0.93	**0.50**
s2	3.27	**3.11**	6.41	**5.50**	0.87	**0.57**
s3	3.83	**3.59**	7.23	**6.09**	1.1	**0.43**
s4	3.38	**3.19**	5.96	**6.09**	0.87	**0.17**
a1	**5.28**	5.70	2.99	**2.98**	**1.4**	1.6
DIM32	10.0	**2.27**	11.7	**1.49**	0.20	**0.0**
unbalance	**0.59**	2.97	**2.49**	9.49	**0.667**	3.87
iris	9.83	**9.34**	8.25	**6.99**		
thyroid	**3.48**	4.39	12.2	**6.26**		
wine	2.77	**2.65**	3.10	**2.57**		
breast	**28.2**	28.4	5.64	**5.59**		
yeast·100	**3.27**	3.57	8.96	**6.56**		
glass	**1.51**	1.78	14.7	**7.56**		
wdbc	**8.11**	**8.11**	**2.56**	**2.56**		
times best	**7**	**8**	**2**	**13**	**2**	**5**

The results in Table 4 show Scut yields more balanced results than k-means. See an example clustering result in Figure 10 for the locations in the Finland dataset. A sightseeing designer would perhaps like to divide the locations into two clusters so that in both clusters there would be a well-balanced number of locations and that route lengths in clusters would be short. To interpret the figures note that in Eastern Finland the density of locations is high in this dataset. From Figure 10 we see that Scut balances better than k-means. The Scut cost is better in Scut than in k-means, because of the balancing property of Scut. k-Means gives better MSE than Scut for this dataset. Figure 10 demonstrates an application area for the Scut method that justifies the usefulness of the method.

The results in Table 4 show that 85% of the clustering results are more balanced with the proposed method than with the k-means method. They were equally balanced in 8% of the cases, and in the remaining 8% of the cases, the k-means result was more balanced. We tried also with a semidefinite programming-based algorithm, but it turned out to be too slow in practice: with 50 points, the computing time is approximately 20 s, but with 150 points it is already about 7 hours. The memory requirement for 150 points would also be very high: 4.4 GB. The results in Table 3 are therefore only for the fast approximation algorithm because it is able to deal with considerably large datasets. For calculations, we used a PC and Octave software.

We next compare the resulting partition sizes of the Scut method and the k-means method, when we have two groups of points of sizes 1 and 100. We assume that all pairwise distances are 1 within each group, and we change the between groups distance from 5 to 10, see Figure 11. Imbalance is calculated using (8). It starts to become lower with Scut, when the inter-partition distance is decreased from 10, whereas with k-means it remains the same, see Figure 12.

Table 4. Mean imbalances and mean execution times of the proposed Scut and k-means clustering for 30 runs. 'k-Means' and 'Scut' refer to algorithms.

Dataset	Points	Clusters	Imbalance		Time	
	n	k	k-means	Scut	k-means	Scut
s1	5000	15	749	**478**	**1.16s**	180s
s2	5000	15	675	**453**	**1.54s**	236s
s3	5000	15	931	**469**	**1.26s**	227s
s4	5000	15	830	**441**	**1.47s**	244s
a1	3000	20	**441**	489	**0.99s**	220s
DIM32	1024	16	26	**0**	**0.33s**	33s
unbalance	6500	8	6146	**2858**	**1.15s**	127s
iris	150	3	21	**4**	**0.12s**	0.50s
thyroid	215	2	173	**126**	**0.12s**	0.66s
wine	178	3	50	**23**	**0.13s**	0.41s
breast	699	2	230	**216**	**0.19s**	0.76s
yeast·100	1484	10	727	**300**	**0.66s**	66s
glass	215	7	183	**114**	**0.17s**	2.96s
wdbc	569	2	**546**	**546**	**0.17s**	0.45s
times best			**2**	**13**	**14**	**0**

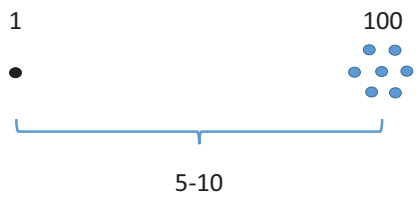


Figure 11. We change the distance of two groups of points between 5 and 10.

We made also an experiment where we plotted MSEs and imbalances for several algorithms and datasets. The rationale for this is that we see which algorithms give both low MSEs and low imbalances. The genetic algorithm [56] combines the properties of several clusterings in one generation to make a better clustering for the next generation. It is the best representative for optimizing *MSE*. In Scut, k-means* and repeated k-means, we chose the results with the best balance. In constrained k-means, the cluster size parameters were set to balance=0, and *MSE* was then optimized. In Ncut we used the implementation [57] by Cour, Yu and Shi from the University of Pennsylvania. We used 100 repetitions for all algorithms and chose the best results, see Figure 13. The genetic algorithm optimizes *MSE* best, but the result is less balanced. Scut always provides a good balance, while balanced k-means and constrained k-means always gave 0 imbalance. k-Means* and Ncut performed well with one dataset. Overall, Scut performs well in both balance and *MSE*.

We made also an experiment to see how much edge weights are inter-partitional and how much are intra-partitional. 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 the

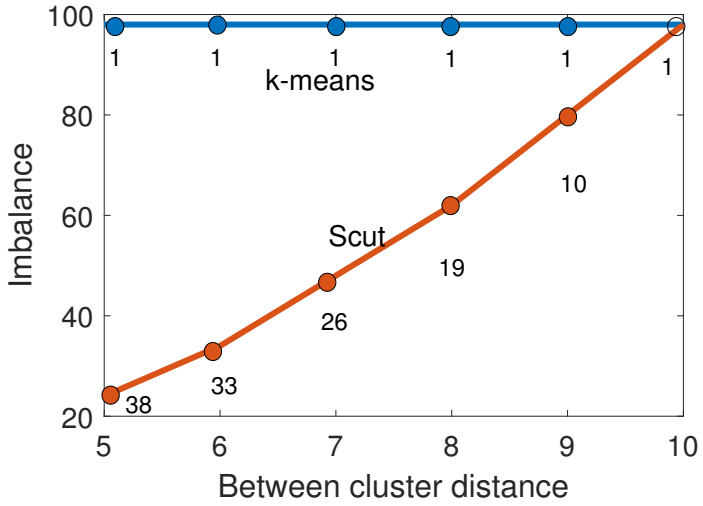


Figure 12. Imbalances for *k*-means and Scut with different inter-partition distance. The number shown is the size of the smaller cluster.

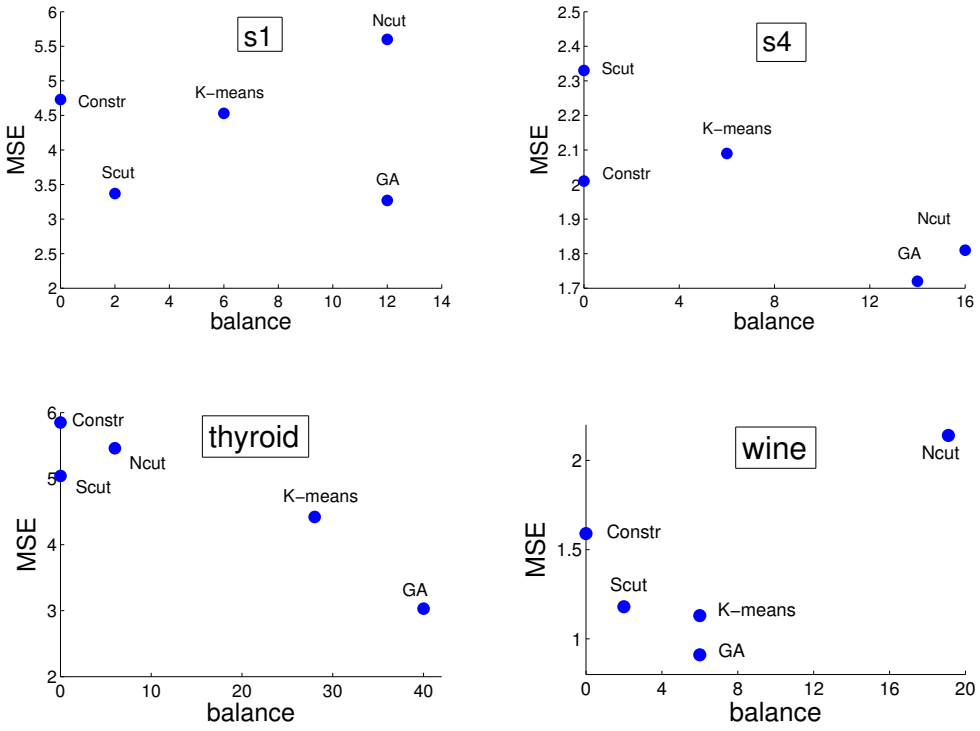


Figure 13. Joint comparison of imbalance and MSE.

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

Dataset	number of clusters	W	MAX k-CUT	**MAX k-CUT/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%**

surprising fact that, in most cases, the cut contains over 90% of all the edge weights. This yields a high value for the upper bound of the approximation factor ϵ_k for the tested sets. This means that a good guaranteed approximation cannot be made for many datasets in the way presented.

7. Conclusions

We have formulated the all-pairwise squared distances cost function as a cut-based method called Squared cut (Scut) using *TSE*, *MSE* and cluster sizes. We showed that this method leads to more balanced clustering. We used the solution of the MAX k-CUT problem to minimize the pairwise intra-cluster squared distances and used Huygens' theorem to show that this corresponds to minimizing the cost function. We gave an alternate proof of Huygens' theorem. Since Scut is expected to be an NP-hard problem, it could not be solved optimally for practical-sized datasets. We, therefore, introduced a sequential k-means algorithm to minimize the cost function directly. We showed through experiments that the proposed approach provides better overall joint optimization of *MSE* and cluster balance compared to other methods. We also treated the case of raising the distances to the pth power, by showing that the relation corresponding to Huygens' theorem does not hold in this case.

Conflict of interest

All authors declare no conflicts of interest in this paper.

References

1. J. H. Ward Jr, Hierarchical grouping to optimize an objective function, *J. Am. Stat. Assoc.,* **58** (1963), 236–244. https://doi.org/10.1080/01621459.1963.10500845
2. T. Kohonen, Median strings, *Pattern Recogn. Lett.,* **3** (1985), 309–313. https://doi.org/10.1016/0167-8655(85)90061-3
3. V. Hautamäki, P. Nykänen, P. Fränti, Time-series clustering by approximate prototypes, *19th International conference on pattern recognition*, (2008), 1–4. IEEE. https://doi.org/10.1109/ICPR.2008.4761105

4. P. Fränti, R. Mariescu-Istodor, Averaging gps segments: competition 2019, *Pattern Recogn.*, **112** (2021), 107730. https://doi.org/10.1016/j.patcog.2020.107730

5. P. Fränti, S. Sieranoja, K. Wikström, T. Laatikainen, *Clustering diagnoses from 58m patient visits in Finland 2015-2018*, 2022.

6. M. Fatemi, P. Fränti, *Clustering nordic twitter users based on their connections*, 2023.

7. M. I. Malinen, P. Fränti, Clustering by analytic functions, *Inform. Sciences*, **217** (2012), 31–38. https://doi.org/10.1016/j.ins.2012.06.018

8. 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.

9. D. Aloise, A. Deshpande, P. Hansen, P. Popat, NP-hardness of Euclidean sum-of-squares clustering, *Mach. Learn.*, **75** (2009), 245–248. https://doi.org/10.1007/s10994-009-5103-0

10. M. Inaba, N. Katoh, H. Imai, Applications of Weighted Voronoi Diagrams and Randomization to Variance-Based *k*-Clustering, *ACM symposium on computational geometry (SCG 1994)*, (1994), 332–339. https://doi.org/10.1145/177424.178042

11. J. MacQueen, Some methods of classification and analysis of multivariate observations, *Berkeley Symp. Mathemat. Statist. Probab.*, **1** (1967), 281–297.

12. W. H. Equitz, A New Vector Quantization Clustering Algorithm, *IEEE Trans. Acoust., Speech, Signal Processing*, **37** (1989), 1568–1575. https://doi.org/10.1109/29.35395

13. P. Fränti, O. Virmajoki, V. Hautamäki, Fast agglomerative clustering using a k-nearest neighbor graph, *IEEE T. Pattern Anal.*, **28** (2006), 1875–1881. https://doi.org/10.1109/TPAMI.2006.227

14. P. Fränti, O. Virmajoki, Iterative shrinking method for clustering problems, *Pattern Recogn.*, **39** (2006), 761–765. https://doi.org/10.1016/j.patcog.2005.09.012

15. P. Fränti, Efficiency of random swap clustering, *Journal of Big Data*, **5** (2018), 1–29. https://doi.org/10.1186/s40537-018-0122-y

16. B. Fritzke, *Breathing k-means*, arXiv:2006.15666.

17. C. Baldassi, Recombinator-k-means:an evolutionary algorithm that exploits k-means++ for recombination, *IEEE T. Evolut. Comput.*, **26** (2022), 991–1003.

18. 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. https://doi.org/10.1111/j.2517-6161.1977.tb01600.x

19. Q. Zhao, V. Hautamäki, I. Kärkkäinen, P. Fränti, Random swap EM algorithm for finite mixture models in image segmentation, *IEEE International Conference on Image Processing (ICIP)*, (2009), 2397–2400. https://doi.org/10.1109/ICIP.2009.5414459

20. J. Shi, J. Malik, Normalized cuts and image segmentation, *IEEE T. Pattern Anal.*, **22** (2000), 888–905. https://doi.org/10.1109/34.868688

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

22. M. I. Malinen, P. Fränti, K-means*: Clustering by gradual data transformation, *Pattern Recogn.*, **47** (2014), 3376–3386. https://doi.org/10.1016/j.patcog.2014.03.034

23. 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.

24. R. Mariescu-Istodor, P. Fränti, Solving the large-scale tsp problem in 1 h: Santa claus challenge 2020, *Front. Robot. AI*, (2021), 1–20. https://doi.org/10.3389/frobt.2021.689908

25. D. W. Sambo, B. O. Yenke, A. Förster, P. Dayang, Optimized clustering algorithms for large wireless sensor networks: A review, *Sensors*, **19** (2019), 322.

26. J. Singh, R. Kumar, A. K. Mishra, Clustering algorithms for wireless sensor networks: A review, *International Conference on Computing for Sustainable Global Development (INDIACom)*, (2015), 637–642.

27. Y. Liao, H. Qi, W. Li, Load-Balanced Clustering Algorithm With Distributed Self-Organization for Wireless Sensor Networks, *IEEE Sens. J.,* **13** (2013), 1498–1506. https://doi.org/10.1109/JSEN.2012.2227704

28. L. Yao, X. Cui, M. Wang, An energy-balanced clustering routing algorithm for wireless sensor networks, *IEEE World Congress on Computer Science and Information Engineering*, **3** (2009), 316–320.

29. P. S. Bradley, K. P. Bennett, A. Demiriz, *Constrained k-means clustering*, Tech. rep., MSR-TR-2000-65, Microsoft Research, 2000.

30. S. Zhu, D. Wang, T. Li, Data clustering with size constraints, *Knowledge-Based Syst.*, **23** (2010), 883–889. https://doi.org/10.1016/j.knosys.2010.06.003

31. A. Banerjee, J. Ghosh, Frequency sensitive competitive learning for balanced clustering on high-dimensional hyperspheres, *IEEE Transactions on Neural Networks*, **15** (2004), 702–719. https://doi.org/10.1109/TNN.2004.824416

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

33. A. Banerjee, J. Ghosh, On scaling up balanced clustering algorithms, *SIAM International Conference on Data Mining*, (2002), 333–349. https://doi.org/10.1137/1.9781611972726.20

34. Y. Chen, Y. Zhang, X. Ji, Size regularized cut for data clustering, *Advances in Neural Information Processing Systems*, 2005.

35. Y. Kawahara, K. Nagano, Y. Okamoto, Submodular fractional programming for balanced clustering, *Pattern Recogn. Lett.*, **32** (2011), 235–243. https://doi.org/10.1016/j.patrec.2010.08.008

36. G. Tzortzis, A. Likas, The minmax k-means clustering algorithm, *Pattern Recogn.*, **47** (2014), 2505–2516. https://doi.org/10.1016/j.patcog.2014.01.015

37. W. Tang, Y. Yang, L. Zeng, Y. Zhan, Optimizing mse for clustering with balanced size constraints, *Symmetry*, **11** (2019), 338. https://doi.org/10.3390/sym11030338

38. L. Hagen, A. B. Kahng, New spectrxal methods for ratio cut partitioning and clustering, *IEEE T. Computer-Aided D.*, **11** (1992), 1074–1085. https://doi.org/10.1109/43.159993

39. T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to algorithms (2nd ed.)*, MIT Press and McGraw-Hill, 2001.

40. M. X. Goemans, D. P. Williamson, Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming, *J. ACM*, **42** (1995), 1115–1145. https://doi.org/10.1145/227683.227684

41. S. Arora, S. Rao, U. Vazirani, Expander flows, geometric embeddings and graph partitioning, *J. ACM*, **56** (2009), 1–37. https://doi.org/10.1145/1502793.1502794

42. U. von Luxburg, A tutorial on spectral clustering, *Stat. Comput.*, **17** (2007), 395–416. https://doi.org/10.1007/s11222-007-9033-z

43. M. R. Garey, D. S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*, W. H. Freeman, 1979.

44. 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.

45. A. Frieze, M. Jerrum, Improved approximation algorithms for max-k-cut and max bisection, *Algorithmica*, **18** (1997), 67–81. https://doi.org/10.1007/BF02523688

46. W. Zhu, C. Guo, A local search approximation algorithm for max-k-cut of graph and hypergraph, *International Symposium on Parallel Architectures, Algorithms and Programming*, (2011), 236–240. https://doi.org/10.1109/PAAP.2011.35

47. A. V. Kel'manov, A. V. Pyatkin, On the complexity of some quadratic euclidean 2-clustering problems, *Comput. Math. Math. Phys.*, **56** (2016), 491–497. https://doi.org/10.1134/S096554251603009X

48. L. J. Schulman, Clustering for edge-cost minimization, *Ann. ACM Symp. on Theory of Computing (STOC)*, (2000), 547–555. https://doi.org/10.1145/335305.335373

49. S. Sahni, T. Gonzalez, P-complete approximation problems, *J. ACM*, **23** (1976), 555–565. https://doi.org/10.1145/321958.321975

50. W. F. de la Vega, M. Karpinski, C. Kenyon, Y. Rabani, Approximation schemes for clustering problems, *ACM symposium on Theory of computing (STOC '03)*, (2003), 50–58. https://doi.org/10.1145/780542.780550

51. N. Guttmann-Beck, R. Hassin, Approximation algorithms for min-sum p-clustering, *Discrete Appl. Math.*, **89** (1998), 125–142. https://doi.org/10.1016/S0166-218X(98)00100-0

52. H. Späth, *Cluster analysis algorithms for data reduction and classification of objects*, Wiley, New York, 1980.

53. P. Fränti, S. Sieranoja, Clustering datasets, University of Eastern Finland, 2020. Available from: `http://cs.uef.fi/sipu/datasets/`.

54. P. Fränti, M. Rezaei, Q. Zhao, Centroid index: Cluster level similarity measure, *Pattern Recogn.*, **47** (2014), 3034–3045. https://doi.org/10.1016/j.patcog.2014.03.017

55. S. Sieranoja, P. Fränti, Fast and general density peaks clustering, *Pattern Recogn. Lett.*, **128** (2019), 551–558. https://doi.org/10.1016/j.patrec.2019.10.019

56. P. Fränti, Genetic algorithm with deterministic crossover for vector quantization, *Pattern Recogn. Lett.,* **21** (2000), 61–68. https://doi.org/10.1016/S0167-8655(99)00133-6

57. T. Cour, S. Yu, J. Shi, *Normalized Cut Segmentation Code*, 2004.

AIMS Press