HACAM: Hierarchical Agglomerative Clustering Around Medoids – and its Limitations

Erich Schubert

TU Dortmund University, Dortmund, Germany

Abstract

Partitioning Around Medoids (PAM) is a popular and flexible clustering method. Also known by the name k-Medoids clustering, and originally conceived for the L_1-norm, it can be used to cluster data into k partitions with respect to an arbitrary distance or similarity measure. The ability to work with any distance makes this method more widely applicable than, for example, k-means clustering. Similar to k-means, a challenge when using PAM is the need to choose the number of clusters, k, before running the algorithm. In many cases, the "optimal" k will not be known beforehand, and the user may need to run PAM several times with different k and rely on additional heuristics to pick the "best" result. We introduce the algorithm Hierarchical Agglomerative Clustering Around Medoids (HACAM), a combination of ideas from classic hierarchical agglomerative clustering (HAC), but where points are clustered around medoids. In our approach, each subtree of the dendrogram has a representative point, which is the medoid: the point with the smallest average distance to all others. In contrast to the arithmetic mean, this does not make assumptions on the data representation or distance function used. Unfortunately, we also show that the requirement to produce a hierarchical result is a limiting factor to the cluster quality, as the optimum result for a particular number of clusters k does not have to be consistent with the optimum result with $k{+}1$ clusters. Hence, if a range of interesting values of k is known beforehand, existing algorithms such as FasterPAM remain favorable.

Keywords

Cluster Analysis, Paritional Clustering, PAM, k-Medoids, Hierarchical Agglomerative Clustering

1. Introduction

In the unsupervised machine learning task of clustering, the unlabeled data set has to be automatically partitioned into "clusters", where objects of high similarity are supposed to be in the same cluster, whereas objects of low similarity should be separated into different clusters. This task is more difficult than this description suggests, because "high" similarity is often not transitive, i.e., even if pairs (a, b) and (b, c) are each of "high" similarity, a and c are not necessarily so. Then we get the conflicting requirement of having a and c in separate partitions, that both should contain b. And since the data is unlabeled, we cannot reliably determine which objects should be in the same partition: the similarity measure (commonly a distance) is only a heuristic, which depends heavily on the preprocessing chosen by the user, such as the scaling and weighting of features, feature selection, projections, and the choice of distance function, which make both the clustering result and the notion of clustering quality highly subjective.

LWDA'21: Lernen, Wissen, Daten, Analysen September 01–03, 2021, Munich, Germany

erich.schubert@tu-dortmund.de (E. Schubert)
0000-0001-9143-4880 (E. Schubert)

Attempts to formally define clusters have had limited success, as none of the more specific definitions is “best” in a general sense [1]. Instead, “clusters are, in large part, in the eye of the beholder” [2], i.e., different users will have different needs to ask for a slightly different definition. Over many years of research, hundreds of clustering algorithms have been proposed, involving different definitions of clusters, objectives, algorithmic structures, merits, and drawbacks. Many of these cannot be found in any toolkit, and seem to have never been used since.

Many seminal methods have been published and receive widespread use, including hierarchical clustering (which can be traced back to numerical taxonomy in the 1950s and 1960s [3, 4]), k-means (its complicated history is studied in [5]), Partitioning Around Medoids [PAM, 6, 7], DBSCAN [8], and OPTICS [9]. And even after 60 to 20 years we still see new publications trying to explain these algorithms better (e.g., [10]), improving their efficiency (e.g., [11, 12]), parallelizing, approximating and scaling them to larger data sets (e.g., [13]), or trying to better understand the similarities and relationships among the published methods (e.g., [14]). In this paper, we attempt to bridge the gap between two of these most classic methods: HAC and PAM. This puts into question a popular, but quite dated, taxonomy of clustering algorithms found in many textbooks, where clustering algorithms are often divided into “partitioning”, “hierarchical”, “density-based”, and “grid-based” [15, Fig. 10.1]. Our method falls into the first two categories, while methods such as OPTICS [9] are both hierarchical and density-based.

2. Related Work

We will first review hierarchical agglomerative clustering (HAC) and partitioning around medoids (PAM, k-medoids), before introducing our combination of these ideas.

2.1. Hierarchical Agglomerative Clustering

In hierarchical agglomerative clustering (HAC), each object initially is a separate cluster. The closest clusters are then repeatedly merged to build a cluster tree called the dendrogram. Because of this algorithmic scheme, this is also referred to as SAHN, for “sequential, agglomerative, hierarchic, nonoverlapping”. HAC is a very flexible method: it can be used with any distance or (dis-)similarity, and it allows for different rules of aggregating the object distances into cluster distances, such as the minimum (“single linkage”), average, or maximum (“complete linkage”). The linkage is an essential parameter of the algorithm, and many such strategies have been proposed and used. Some example linkages are given in Table 1. Single linkage corresponds to the minimum spanning tree of the distance graph; and while this linkage is best understood theoretically, it is known to not work well in practical applications because of the “single-link chaining effect”. Ward linkage, which measures the increase in squared errors, is closely related to k-means clustering. We also include the more exotic MiniMax linkage [23], because of its similarity to our approach, and two previous hierarchical approaches that use medoids [24, 25]. In this table, we can observe some special cases. Certain linkages (Ward, Centroid, Median) are defined using squared distances. Squared Euclidean distance allows for certain transformations (the König-Huygens theorem) that allow for a simpler, and more efficient, computation, where we do not need to compute the cluster means μ to compute these clusterings. The use of the mean is also most meaningful for squared Euclidean, so these linkages are closely tied

Table 1: Selected linkage strategies

Names	**Definition** (for distances)
Single [3] Minimum	$d(A,B) := \min_{a\in A} \min_{b\in B} d(a,b)$
Complete [16, 17, 18] Maximum	$d(A,B) := \max_{a\in A} \max_{b\in B} d(a,b)$
(Group) Average [19] UPGMA	$d(A,B) := \frac{1}{\lvert A\rvert\cdot\lvert B\rvert} \sum_{a\in A} \sum_{b\in B} d(a,b)$
Weighted Average [19] WPGMA, McQuitty [20]	$d(A\cup A',B) := \frac{1}{2}\Big(d(A,B)+d(A',B)\Big)$
Centroid [21] UPGMC	$d^2(A,B) := \lVert\mu_A-\mu_B\rVert_2^2$
Median [21] WPGMC	$d^2(A\cup A',B) := \left\lVert\frac{1}{2}\left(\mu_A+\mu_{A'}\right)-\mu_B\right\rVert_2^2$
Ward [22]	$d^2(A,B) := \frac{2\lvert A\rvert\lvert B\rvert}{\lvert A\cup B\rvert}\cdot\lVert\mu_A-\mu_B\rVert_2^2$
Uncommon:	
Mini-Max [23]	$d(A,B) := \min_{r\in A\cup B} \max_{p\in A\cup B} d(r,p)$
Medoid [24, 25]	$d(A,B) := d(\text{medoid}(A),\text{medoid}(B))$
New proposals in this publication:	
Min-sum	$d(A,B) := \min_{m\in A\cup B} \sum_{p\in A\cup B} d(m,p)$
Min-sum-increase	$d(A,B) := \min\limits_{m\in A\cup B} \sum\limits_{p\in A\cup B} d(m,p) - \min\limits_{m\in A} \sum\limits_{p\in A} d(m,p) - \min\limits_{m\in B} \sum\limits_{p\in B} d(m,p)$

to the 2-norm (although it could be the 2-norm in a "kernel" vector space). The definition of McQuitty's linkage (WPGMA) and median linkage (WPGMC) also comes with a subtle hidden problem: it depends on the previous partitions A and A', i.e., it is a recursive definition that we cannot compute without knowing the merge history that led to these partitions. Such recursive definitions also exist for other linkages and have been generalized to what is called Lance-Williams equations [26]. In Table 2 we give such recursive definitions for several linkages, and it is easy to see that these can be computed in $O(1)$ from the known previous distances $d(A,B)$ and $d(A',B)$, whereas the equations in Table 1 can require up to $O(|A|\cdot|B|)=O(n^2)$ operations to compute. Because of this, the preferred way of performing hierarchical clustering is by using such recursive forms (and as mentioned, McQuitty and Median linkage can only be defined recursively). Unfortunately, for some linkages such as Mini-Max and our proposed linkage based on the minimum sum, such a simple recurrence *cannot* exist. This has implications on the computational cost to perform such a clustering.

While the name "medoid linkage" for our approach would have been appropriate because of the relationship to PAM (partitioning around medoids), this name has been previously used for a simple heuristic variation of median linkage, using the distances of the old medoids to choose the next merge and independently proposed by multiple groups [24, 25]. Our approach will produce better results than this naive definition because the closeness of two medoids is only a rough proxy for the quality of the resulting medoid. A problem of medoid linkage becomes

Table 2: Recursive definitions of selected linkages

Name	**Recurrence** (for distances)
Single	$d(A \cup A', B) := \min\{d(A,B), d(A',B)\}$
Complete	$d(A \cup A', B) := \max\{d(A,B), d(A',B)\}$
Average	$d(A \cup A', B) := \frac{\lvert A\rvert}{\lvert A \cup A'\rvert} d(A,B) + \frac{\lvert A'\rvert}{\lvert A \cup A'\rvert} d(A',B)$
McQuitty	$d(A \cup A', B) := \frac{1}{2} d(A,B) + \frac{1}{2} d(A',B)$
Centroid	$d^2(A \cup A', B) := \frac{\lvert A\rvert}{\lvert A \cup A'\rvert} d^2(A,B) + \frac{\lvert A'\rvert}{\lvert A \cup A'\rvert} d^2(A',B) - \frac{\lvert A\rvert \cdot \lvert A'\rvert}{\lvert A \cup A'\rvert^2} d^2(A,A')$
Median	$d^2(A \cup A', B) := \frac{1}{2}\left(d(A,B) + d^2(A',B) - \frac{1}{2} d^2(A,A')\right)$
Ward	$d^2(A \cup A', B) := \frac{\lvert A\rvert + \lvert B\rvert}{\lvert A \cup A' \cup B\rvert} d^2(A,B) + \frac{\lvert A'\rvert + \lvert B\rvert}{\lvert A \cup A' \cup B\rvert} d^2(A',B) - \frac{\lvert B\rvert}{\lvert A \cup A' \cup B\rvert} d^2(A,A')$

evident when considering the first merge: both points are equally good medoids, but this choice has a substantial impact on the subsequent merges.

2.2. Algorithms for Hierarchical Agglomerative Clustering

HAC methods will usually require a full distance matrix to be computed and stored, hence they have a lower bound of $O(n^2)$ for time and memory for clustering n objects. The standard algorithm discussed below requires $O(n^3)$ time. For the special case of single-linkage, $O(n)$ memory is sufficient when using a variant of Prim's algorithm known as SLINK [27]. For complete-linkage, the linear-memory approximation CLINK has been proposed [28], but it tends to produce worse results and the results depend on the ordering of points [28]. In Figure 1 we give the basic idea of hierarchical clustering: find the best merge, greedily execute the merge, then update the distance matrix. It is easy to see that line 3 will be executed exactly $n-1$ times. Finding the best merge in line 4 with brute force requires $O(n^2)$ operations (we will discuss better strategies below). Merging clusters in line 6 is $O(n)$, and therefore not much of a concern. Updating the dissimilarity matrix in line 7 is where we need to be most careful. A naive implementation that recomputes the entire matrix each iteration performs $O(n^2)$ work here while using the Lance-Williams equations allows computing this in just $O(n)$ from the previous values in the matrix. Because of the outer loop, the overall complexity is then $O(n^3)$. Because the bottleneck with Lance-Williams is the $\arg\min$ operation in line 4, several techniques have been proposed for this. Several authors have proposed to use heaps here, which reduces the runtime to $O(n^2 \log n)$, but with fairly high constant factors (because the heap is of size $O(n^2)$, and needs to be updateable). An interesting alternative, which we can recommend because of its simplicity, is found in a small note of the book by Anderberg [29], who suggested caching the location of the minimum in each row. According to his analysis, this will typically reduce the effort to n^2, but the theoretical worst-case remains $O(n^3)$. In our experiments with real data, this practical result is much more useful than the theoretical analysis; and this low-overhead approach outperforms heap-based approaches, as previously observed by Kriegel et al. [30].

Figure 1: Abstract Hierarchical Agglomerative Clustering

```
$D \leftarrow$ pairwise dissimilarity matrix
$C \leftarrow \{\{x_1\}, \ldots, \{x_N\}\}$ singleton clusters
while $|C| > 1$ do
    $i, j \leftarrow \arg\min D$ (use $\arg\max$ with a similarity matrix)        // find best merge
    add $(i, j, d_{ij})$ to dendrogram
    $C \leftarrow$ merge clusters $i$ and $j$ into $i$, remove $j$
    $D \leftarrow$ update column and row $i$
```

2.3. k-means Clustering

Probably the most widely taught clustering algorithm is k-means. The standard algorithm uses alternating minimization: first, each point is assigned to the nearest cluster mean (by least squared error), then the cluster mean of each cluster is updated. Both steps minimize the sum-of-squared errors (SSQ, SSE, equivalent to minimizing WCSS):

$$SSQ := \sum\nolimits_{i=1}^{k} \sum\nolimits_{x_j \in C_i} \|x_j - \mu_i\|_2^2 \quad . \tag{1}$$

In hierarchical clustering, this closely matches the objective of Ward linkage, which minimizes the increase in SSQ each step. It is a consequence of the König-Huygens-Steiner theorem that we can compute this directly from the means (Table 1) or recursively (Table 2).

Because k-means is closely tied to least-squares and hence squared Euclidean distance, it is desirable to extend this to other distance functions. However, the arithmetic mean only minimizes the sum of squares, hence we need to choose an appropriate replacement. Manhattan distance (also known as the city-block metric, or L_1) is minimized by the component-wise median [31, k-medians]; which can be computed by partial sorting. Minimizing Euclidean distances is already harder (k-means does *not* minimize Euclidean distances) and known as the Weber problem [32], which has no closed-form solution [31] (for a recent survey of algorithms for the Weber point see Fritz et al. 33). Each distance function yields a new optimization problem, but the medoid discussed next is a discrete and easy-to-understand approximation to all of them.

2.4. k-medoids Clustering

A generic replacement is the *medoid* of a set C (also called the discrete median in some literature, although this can easily be confused with the component-wise median):

$$\text{medoid}(C) := \arg\min_{x_i \in C} \sum\nolimits_{x_j \in C} d(x_i, x_j) \tag{2}$$

It can easily be seen that this is usable with an arbitrary dissimilarity function d, and can even be extended to similarities by maximizing instead of minimizing. Hence, this approach is also not limited to vector spaces $\mathbb{R}^d$, but could be used to cluster, e.g., phonemes by Levenshtein distance. On the other hand, in the general non-metric case, we can only solve it by enumeration.

Figure 2: Abstract Partitioning Around Medoids (PAM) BUILD algorithm

```
$m_1 \leftarrow \arg\min_{x_i \in X} \sum_{x_j \in X} d(x_i, x_j)$                // medoid of $X$
$M \leftarrow \{m_1\}$
while $|M| < k$ do
    $j \leftarrow \arg\min_{x_j \in X \setminus M} \Delta(M, \emptyset, x_j)$          // best reduction in TD
    $M \leftarrow M \cup \{x_j\}$                                  // add to the chosen medoids
```

In k-medoids, we model the data using k medoids m_i as "prototypes" for the clusters, and attempt to optimize the absolute error criterion ("total deviation"):

$$TD := \sum\nolimits_{i=1}^{k} \sum\nolimits_{x_j \in C_i} d(x_j, m_i) \quad , \tag{3}$$

i.e., the sum of distances of each point $x_j \in C_i$ to the medoid m_i of its cluster. This closely resembles the SSQ objective of k-means if $d(x, m) = \|x - m\|_2^2$, except that here m_i is chosen "discretely" from the input data, instead of "continuously" from the vector space. Because of this restriction, k-medoids will usually score worse than k-means when used with this objective, and will also take considerably longer because finding the medoids is more expensive than finding the arithmetic mean. Because this clustering problem is NP-hard [34], we have to focus on efficient approximations to this problem.

2.5. Algorithms for k-medoids Clustering

Alternating optimization similar to k-means is an obvious approach to try (and has been proposed several times, e.g., [35]), but it was repeatedly found to produce much worse results than the swapping technique discussed next [36, 37, 38, 12]. Schubert and Rousseeuw [12] provide a simple example where this approach gets stuck in a bad local optimum, which PAM can easily escape. Partitioning Around Medoids (PAM [6, 7]) is a standard algorithm to find a good partitioning using medoids, with respect to TD (Eq. 3). PAM consists of two algorithms for this problem: BUILD is a greedy algorithm to find a good initial result, while SWAP is a local search algorithm that replaces the medoid with the non-medoid which reduces TD best (if no further reduction is possible, the algorithm stops). Figure 2 and Figure 3 outline the general idea of BUILD and SWAP, but we omit the details of how to efficiently compute the function $\Delta(M, m_i, x_j)$ here that computes the change in TD when removing medoid m_i and adding medoid x_j. The naive approach of computing TD for each combination of m_i and x_j takes $O(n^2k^2)$ time, but by caching the distances to the nearest and second nearest, PAM reduced this to $O(n^2k)$, and FastPAM [11, 12] recently further reduced this to $O(n^2)$ by processing all m_i at once. This removes the need for BUILD, but allows starting with a random choice of medoids.

For most algorithms for k-medoids, it is beneficial to store the matrix of pairwise distances, because all distances will be used multiple times. BanditPAM [39] is a recent proposal that attempts to optimize TD without a stored matrix using multi-armed bandits. Earlier approaches that avoid this include CLARA [40] (using PAM on subsamples) and CLARANS [41] (only exploring a subset of possible swaps). These may work well on "easy" data sets, but will likely produce much worse results on challenging data sets.

Figure 3: Abstract Partitioning Around Medoids (PAM) SWAP algorithm

```
$M \leftarrow \{m_1, \ldots, m_k\}$ initial cluster centers      // initial medoids chosen via BUILD
while true do
    $i, j, \Delta' \leftarrow \arg\min_{m_i \in M, x_j \in X \setminus M} \Delta(M, m_i, x_j)$      // find the best swap
    if $\Delta' >= 0$ then break loop      // no improvement found
    $M \leftarrow (M \setminus \{m_i\}) \cup \{x_j\}$      // update set of medoids
```

In this article, we propose a new hierarchical approach to clustering around medoids (without a predefined value of k). We do not require the distance d to be a metric (i.e., we do not assume the triangle inequality), and by using $\arg\max$ instead of the $\arg\min$ it can also be applied to similarities. Similar to PAM (and the standard agglomerative hierarchical clustering algorithm), it requires a full distance matrix to be computed and stored.

3. Hierarchical Agglomerative Clustering Around Medoids

The method we propose is a straightforward combination of the ideas of hierarchical agglomerative clustering via the algorithm shown in Figure 1 with a novel linkage method based on the ideas of k-medoids clustering. We then also discuss an approach to accelerate this algorithm.

3.1. Min-Sum and Min-Sum-Increase Linkage

While our linkage proposal is directly based on the ideas of k-medoids (more specifically, on the idea of optimizing TD), other authors previously proposed a different approach named “Medoid linkage” [24, 25] using the distances of the medoids. Hence we opt for a more specific name: minimum sum. This name is inspired by the linkage variants of Podani [42] (some of which were also discussed before by Jambu and Lebeaux [43]), who would have called these variants MNSUM and MISUM, as well as the Mini-Max clustering of Ao et al. [23]. The basic idea is that we want to minimize TD, i.e., the sum of distances to the current set of medoids. Initially, each point is a medoid, and the sum hence is 0.

Minimum Sum: The first strategy is to always merge those clusters, whose medoid yields the smallest contribution to TD, i.e., choose the two clusters A, B with the “cheapest” medoid:

$$d(A,B) := \min\nolimits_{m \in A \cup B} \sum\nolimits_{p \in A \cup B} d(m,p) \tag{4}$$

Minimum Sum Increase: An alternative – and better – strategy is to instead only consider the *increase* in total deviation caused by the new merge:

$$d(A,B) := \min_{m \in A \cup B} \sum_{p \in A \cup B} d(m,p) - \min_{m \in A} \sum_{p \in A} d(m,p) - \min_{m \in B} \sum_{p \in B} d(m,p) \tag{5}$$

This equation describes the loss change when merging two clusters instead of the resulting loss. Note that the PAM algorithm also uses the loss change, and the well-known Ward linkage also uses the *increase* in variance. It needs $O(n)$ additional memory to store the previous values.

Computational Complexity Unfortunately, neither of the two methods can be expressed using a Lance-Williams equation, i.e., the loss cannot be computed just from the losses of the two sub-clusters. Instead, to compute these linkages, we need to find the medoid of the resulting partition. Without further constraints (e.g., metric properties), finding the medoid takes $O(n^2)$ time for a partition of size n. In the standard HAC algorithm (c.f., Figure 1, Line 7), we need to compute the new linkage distances after merging two clusters, i.e., one row/column of this symmetric matrix. Lance-Williams allow doing each such computation in $O(1)$, and hence a merge can be computed in $O(n)$. Naively, one may assume the cost for this linkage then is $O(n^3)$ for each iteration and $O(n^4)$ total. However, at any step, if we have many clusters they are small, and if we have large clusters we can only have few. Because of this, the cost is only quadratic per iteration even when chaining occurs (for an analogous discussion, see Bien and Tibshirani [44]), and hence the cost is cubic in total; the same as regular HAC. Furthermore, we employ Anderberg caching [29] to reduce the time to find the smallest distance, such that the method often performs not *much* worse than quadratic (although asymptotically cubic).

No Recurrent Solution Many linkage strategies for hierarchical clustering can be computed more efficiently using Lance-Williams equations as shown in Table 2. With this equation, the linkage cost of two clusters can be computed in $O(1)$ from the previous distances and a constant-size summary of the previous clusters (usually the number of objects only). Similar to Mini-Max linkage [44], our proposed linkages do not fall into this class. It is easy to see that finding the medoid of the union of two sets A and B cannot be done in $O(1)$, even if we know the medoid of both A and B, because the new medoid will usually be some other of the $n = |A|+|B|$ points in the set. It is possible to build a counterexample similar to Figure 9 of Bien and Tibshirani [44], but their proof does not rule out keeping additional constant-size summary information as used in some extensions of Lance-Williams (nevertheless we are confident that Mini-Max also cannot have a solution to compute merging costs in $O(1)$). In general, we will need to consider all points as possible medoids, which results in using all $O(n^2)$ pairwise distances. It is hence advisable to use this method only with a precomputed distance matrix. Some of the $n = |A|+|B|$ candidates can be eliminated early at least in low-dimensional Euclidean or Euclidean-embeddable spaces, where finding a solution in $O(n^{3/2})$ is possible [45].

4. Limitations

At first sight, this appears to help to solve a key problem of k-medoids clustering: choosing k. Computing a dendrogram for all k, then choosing the appropriate cut based on the heights of the trees would provide a nice solution to this problem. Unfortunately, it turns out that the results of HACAM for small k can be much worse than those found by PAM-style algorithms. Only for very large $k \geq \frac{n}{2}$, where the average cluster size is less than two, HACAM has a good chance of finding reasonable results; while at the same time the results will then not differ much from single-linkage (nearest neighbor) clustering, and we are not really using medoids.

This problem can be explained on a one-dimensional data set containing the numbers $1 \ldots N$, illustrated in Figure 4. The optimum solution for k partitions consists of k approximately equal intervals. Clearly, the optimum solution for $k>1$ cannot be constructed from the optimum

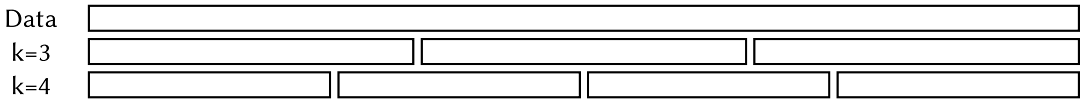


Figure 4: Illustration of why the results of HACAM cannot be optimal for both k=3 and k=4: the optimum result for k=3 cannot be constructed by agglomeration of the optimum k=4 result.

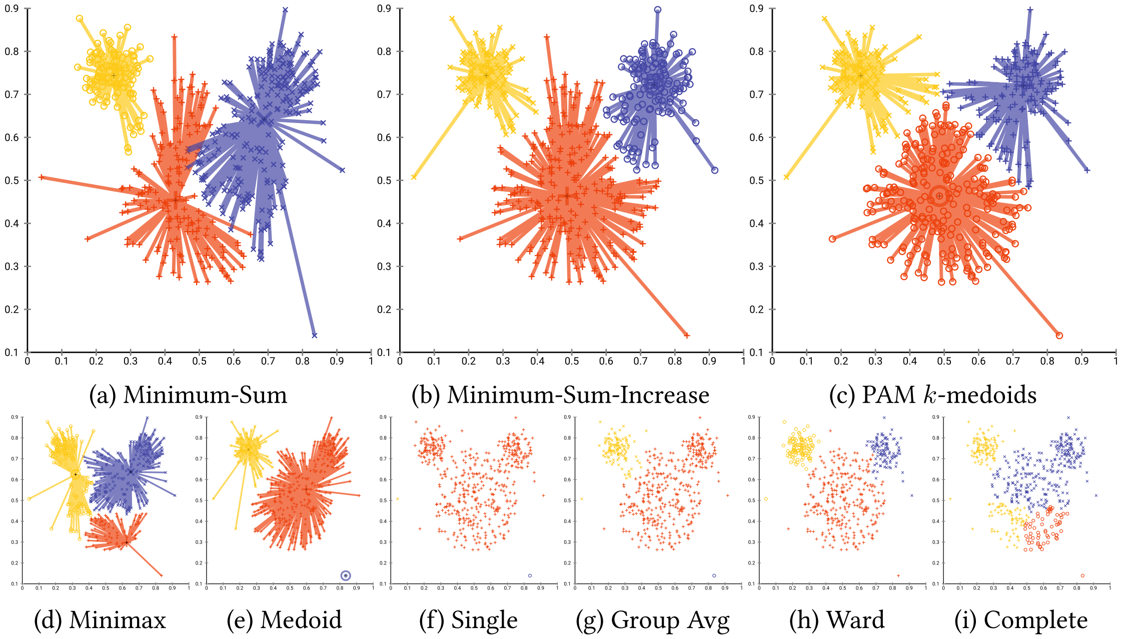
solution for k+1 *by aggregation*. It is this constraint to nestable, hierarchical clusters that prevents this approach from finding the optimum result for all k: *any* hierarchical clustering for which these equally sized intervals are optimal cannot contain the optimum solution for both k and k+1 at the same time, for any k. While we showcase this for an idealized case only, it is clear that these situations will be very common, and hence we are more likely than not to obtain an inferior result than when using k-medoids with the desired k directly. It is worth noting that this negative result likely transfers to most other linkages *except* single-linkage clustering. The strength of such approaches – including the presented approach – lies in producing a hierarchical (nesting) result, not the ability to cut this hierarchy into k flat partitions. Hence, such methods are best used if we intend to further use the hierarchy in one way or another.

A second problem that limits the practical applicability, e.g., for facility location, is that medoids change during merges. I.e., the medoid of $A \cup B$ will usually be *neither* the medoid of A nor B. Consider the facility location problem, where a company tries to optimally cover its customers using k facilities. It would be desirable to have the medoids for the solution with k facilities be a subset of the solution with k+1 facilities to be able to open and close facilities. Enforcing such consistency bottom-up would be possible, but will likely yield even worse results, as globally best medoids will likely be deleted too early. Instead, a top-down approach is more likely useful – and that is exactly what the greedy BUILD heuristic of PAM does: choose the best solution for k=1, then find the best solution with k+1 that contains the k-solution as a subset of medoids (note that this approach does not yield a tree). Alternatively, we could repeatedly run k-medoids with k=2 to split the data recursively.

If we know a particular k of central interest, we can also experiment with hybrid combinations: first, obtain a good solution for the k of interest using k-medoids, then construct the hierarchy above using the proposed method, while constructing the hierarchy below up to desired granularity using k-medoids with k=2.

5. Experiments

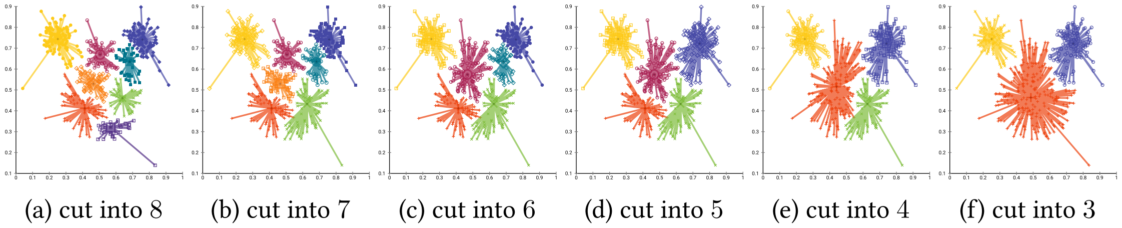
We first experiment with the toy “mouse” example, which consists of three clusters and background noise. This data set is not well solvable neither by k-means nor by k-medoids with Euclidean distance, because of the different cluster diameters. It is easy to solve with Gaussian Mixture Modeling. Figure 5 depicts the clustering result using Min-Sum clustering (Figure 5a), Min-Sum-Increase clustering (Figure 5b), and standard k-medoids with PAM (Figure 5c). It can be seen that Min-Sum-Increase worked better than regular Min-Sum, but also that PAM produces a better result, with convex clusters. Medoid linkage (Figure 5e) has problems with outliers in the data that are merged very late (its solution with $k = 5$ was much better). Other



(a) Minimum-Sum (b) Minimum-Sum-Increase (c) PAM k-medoids

(d) Minimax (e) Medoid (f) Single (g) Group Avg (h) Ward (i) Complete

Figure 5: Clustering the mouse data set into 3 clusters (Fig. 5f to Fig. 5i do not have prototypes).



(a) cut into 8 (b) cut into 7 (c) cut into 6 (d) cut into 5 (e) cut into 4 (f) cut into 3

Figure 6: Clustering the mouse data set with Min-Sum-Increase into $8 \dots 3$ clusters.

classic linkages such as complete linkage (Figure 5i) also exhibit non-convex clusters, and sensitivity to outliers (group-average linkage, Figure 5g). The non-convexity of the HACAM clusters is caused by the hierarchy: the union of two clusters is not necessarily convex. To understand the results, we have to look at the merge history that led to the result. This is shown in Figure 6, which begins with eight clusters, the closest two of which are then merged repeatedly until only three clusters remain.

In Table 3 we study the result quality as well as runtime on a collection of 59 benchmark problems from operations research (ORLib, for details, see [12]). We use 10 restarts with different permutations of the data sets; because of duplicate distances in this data, this has a major effect on the results. While FasterPAM found the optimum solution for 30 problems with 10 restarts, MinSumInc only found 21 optimal solutions; the other approaches performed much worse. On average, FasterPAM results were 1.7% worse than the optimum, while MinSumInc was 35.1% worse – a non-negligible difference. Again, the alternatives worked much worse. The run-time was about three times that of FasterPAM.

Table 3: Evaluation on the ORLib benchmark data sets (10 shuffles, 59 data sets, c.f., [12])

Method	average extra loss	minimum extra loss	optimum found	run time
FasterPAM	1.7%	0.6%	30	111 ms
MinSumInc	35.1%	2.3%	21	330 ms
MinSum	74.3%	64.3%	0	309 ms
Medoid	129.2%	84.2%	4	272 ms
MiniMax	52.3%	34.0%	9	269 ms

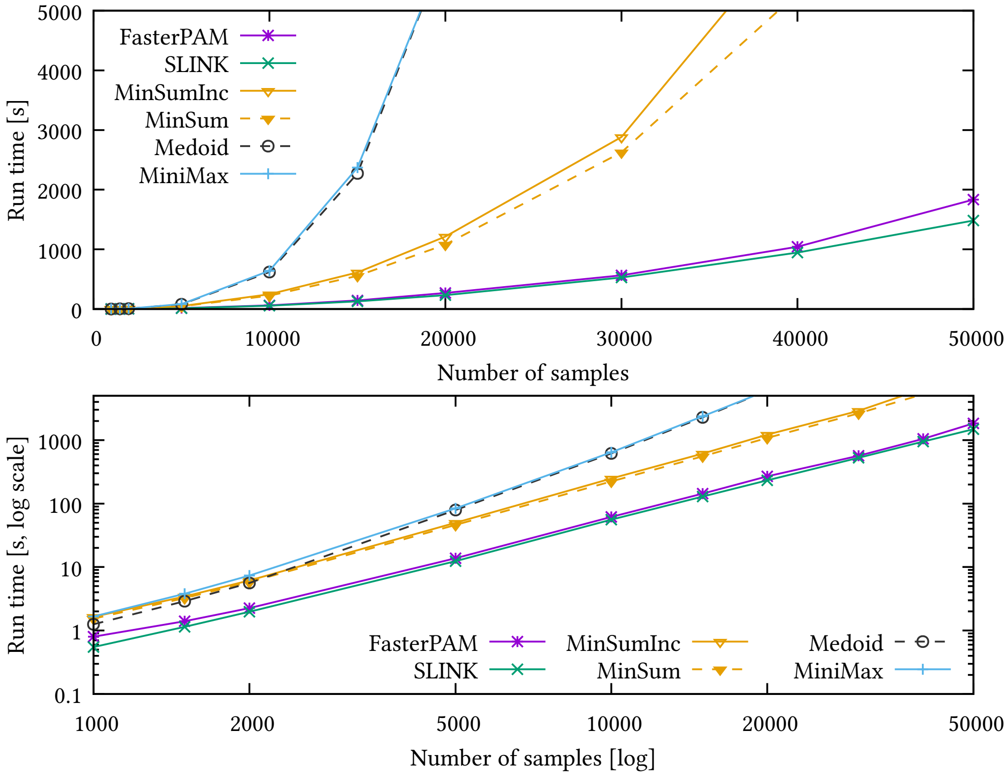


Figure 7: Runtime on subsets of the MNIST data set (linear and log-log scale)

In Figure 7 we explore the scalability using subsets of the MNIST data set. SLINK is an efficient $O(n^2)$ method, while the Medoid and MiniMax implementations are based on AGNES and in $O(n^3)$. For MinSum and MinSumInc we use Anderberg [29] caching, which often performs approximately quadratic, but the worst case is $O(n^3)$. Neither scales to big data sets.

A fair quality evaluation would require a hierarchy-based evaluation, for which no standard methodology is established. The source code will be made available in ELKI [46].

6. Conclusions

In this article, we propose two new linkage strategies based on the ideas of k-medoids clustering, i.e., of optimizing the sum of distances to a cluster representative chosen from the data. This method is closely related to the earlier proposal of Mini-Max clustering of Ao et al. [23]. While the new approach has its merits, it is rather slow (in particular, slower than running FasterPAM for multiple small k), and because of the hierarchy constraint, the results tend to be much worse. We give a theoretical argument why this happens, and why we cannot expect a hierarchical result to contain the optimum solution for every k. Hence this linkage will be only of occasional use in applications where hierarchical results are desired, as opposed to flat partitionings with a fixed k where FasterPAM [12] will be able to find better results faster. One example of such a use case is to first partition the data into k partitions with PAM, then construct a hierarchy only above this to explore how these clusters relate to each other, e.g., to allow for interactive exploration and to explain the clustering based on the medoids as representative objects. But even then the discussed limitations prevail: the medoid of an aggregation of two well-separated clusters will often be an object at the border of one cluster, close to the other cluster, in order to explain both, while neither of the previous medoids may be good at explaining both clusters.

Hence further research is needed to obtain clusterings that are both easy to understand (e.g., by the use of cluster representatives such as medoids or means) as well as hierarchical navigable to allow to “drill-down” into or “roll-up” clusters for exploration and explanation.

References

[1] R. E. Bonner, On some clustering techniques, IBM Journal of Research and Development 8 (1964) 22–32. doi:10.1147/rd.81.0022.

[2] V. Estivill-Castro, Why so many clustering algorithms: a position paper, SIGKDD Explorations 4 (2002) 65–75. doi:10.1145/568574.568575.

[3] P. H. A. Sneath, The application of computers to taxonomy, Microbiology 17 (1957) 201–226. doi:10.1099/00221287-17-1-201.

[4] R. R. Sokal, P. H. A. Sneath, Principles of Numerical Taxonomy, Books in Biology, W. H. Freeman, 1963.

[5] H. Bock, Clustering methods: A history of k-means algorithms, in: P. Brito, G. Cucumel, P. Bertrand, F. Carvalho (Eds.), Selected Contributions in Data Analysis and Classification, Springer, 2007, pp. 161–172. doi:10.1007/978-3-540-73560-1_15.

[6] L. Kaufman, P. J. Rousseeuw, Clustering by means of medoids, in: Y. Dodge (Ed.), Statistical Data Analysis Based on the L_1 Norm and Related Methods, 1987, pp. 405–416.

[7] L. Kaufman, P. J. Rousseeuw, Finding Groups in Data: An Introduction to Cluster Analysis, John Wiley&Sons, 1990. doi:10.1002/9780470316801.

[8] M. Ester, H. Kriegel, J. Sander, X. Xu, A density-based algorithm for discovering clusters in large spatial databases with noise, in: Knowledge Discovery and Data Mining, KDD, 1996, pp. 226–231.

[9] M. Ankerst, M. M. Breunig, H. Kriegel, J. Sander, OPTICS: ordering points to identify the clustering structure, in: SIGMOD, 1999, pp. 49–60. doi:10.1145/304182.304187.

[10] E. Schubert, J. Sander, M. Ester, H. Kriegel, X. Xu, DBSCAN revisited, revisited: Why and how you should (still) use DBSCAN, ACM Trans. Database Syst. 42 (2017) 19:1–19:21. doi:10.1145/3068335.

[11] E. Schubert, P. J. Rousseeuw, Faster k-medoids clustering: Improving the PAM, CLARA, and CLARANS algorithms, in: 12th Int. Conf. Similarity Search and Applications, SISAP 2019, 2019, pp. 171–187. doi:10.1007/978-3-030-32047-8_16.

[12] E. Schubert, P. J. Rousseeuw, Fast and eager k-medoids clustering: O(k) runtime improvement of the PAM, CLARA, and CLARANS algorithms, Information Systems 101 (2021) 101804. doi:10.1016/j.is.2021.101804.

[13] A. Lang, E. Schubert, BETULA: numerically stable cf-trees for BIRCH clustering, in: SISAP, 2020, pp. 281–296. doi:10.1007/978-3-030-60936-8_22.

[14] E. Schubert, S. Hess, K. Morik, The relationship of DBSCAN to matrix factorization and spectral clustering, in: Lernen, Wissen, Daten, Analysen, LWDA, 2018, pp. 330–334. URL: http://ceur-ws.org/Vol-2191/paper38.pdf.

[15] J. Han, M. Kamber, J. Pei, Data Mining: Concepts and Techniques, 3rd edition, Morgan Kaufmann, 2011.

[16] T. Sørensen, A method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on danish commons, Kongelige Danske Videnskabernes Selskab 5 (1948).

[17] P. Macnaughton-Smith, Some statistical and other numerical techniques for classifying individuals, Technical Report Home Office Res. Rpt. No. 6, H.M.S.O., London, 1965.

[18] S. C. Johnson, Hierarchical clustering schemes, Psychometrika 32 (1967) 241–254. doi:10.1007/BF02289588.

[19] R. R. Sokal, C. D. Michener, A statistical method for evaluating systematic relationship, The University of Kansas Science Bulletin 38 (1902) 1409–1438.

[20] L. L. McQuitty, Elementary linkage analysis for isolating orthogonal and oblique types and typal relevancies, Educational and Psychological Measurement 17 (1957) 207–229. doi:10.1177/001316445701700204.

[21] J. C. Gower, A comparison of some methods of cluster analysis, Biometrics 23 (1967) 623. doi:10.2307/2528417.

[22] J. H. Ward, Hierarchical grouping to optimize an objective function, Journal of the American Statistical Association 58 (1963) 236–244. doi:10.1080/01621459.1963.10500845.

[23] S. I. Ao, K. Y. Yip, M. K. Ng, D. W. Cheung, P. Fong, I. Melhado, P. C. Sham, CLUSTAG: hierarchical clustering and graph methods for selecting tag SNPs, Bioinformatics 21 (2005) 1735–1736. doi:10.1093/bioinformatics/bti201.

[24] D. Herr, Q. Han, S. Lohmann, T. Ertl, Visual clutter reduction through hierarchy-based projection of high-dimensional labeled data, in: Graphics Interface Conference, 2016, pp. 109–116. doi:10.20380/GI2016.14.

[25] S. Miyamoto, Y. Kaizu, Y. Endo, Hierarchical and non-hierarchical medoid clustering using asymmetric similarity measures, in: SCIS/ISIS, 2016, pp. 400–403. doi:10.1109/SCIS-ISIS.2016.0091.

[26] G. N. Lance, W. T. Williams, Mixed-data classificatory programs I - agglomerative systems, Australian Computer Journal 1 (1967) 15–20.

[27] R. Sibson, SLINK: An optimally efficient algorithm for the single-link cluster method, The

Computer Journal 16 (1973) 30–34. doi:10.1093/comjnl/16.1.30.

[28] D. Defays, An efficient algorithm for the complete link cluster method, The Computer Journal 20 (1977) 364–366. doi:10.1093/comjnl/20.4.364.

[29] M. R. Anderberg, Hierarchical Clustering Methods, Probability and mathematical statistics, Academic Press, 1973, pp. 131–155. doi:10.1016/B978-0-12-057650-0.50012-0.

[30] H. Kriegel, E. Schubert, A. Zimek, The (black) art of runtime evaluation: Are we comparing algorithms or implementations?, Knowl. Inf. Syst. 52 (2017) 341–378. doi:10.1007/s10115-016-1004-2.

[31] P. S. Bradley, O. L. Mangasarian, W. N. Street, Clustering via concave minimization, in: Adv. Neural Information Processing Systems 9, NIPS, 1996, pp. 368–374. URL: http://papers.nips.cc/paper/1260-clustering-via-concave-minimization.

[32] M. L. Overton, A quadratically convergent method for minimizing a sum of euclidean norms, Math. Program. 27 (1983) 34–63. doi:10.1007/BF02591963.

[33] H. Fritz, P. Filzmoser, C. Croux, A comparison of algorithms for the multivariate l_1-median, Computational Statistics 27 (2012) 393–410. doi:10.1007/s00180-011-0262-4.

[34] O. Kariv, S. Hakimi, An algorithmic approach to network location problems. II: The p-medians, SIAM J. Appl. Math. 37 (1979) 539–560. doi:10.1137/0137041.

[35] F. E. Maranzana, On the location of supply points to minimize transportation costs, IBM Systems Journal 2 (1963) 129–135. doi:10.1147/sj.22.0129.

[36] M. B. Teitz, P. Bart, Heuristic methods for estimating the generalized vertex median of a weighted graph, Operations Research 16 (1968) 955–961. doi:10.1287/opre.16.5.955.

[37] K. E. Rosing, E. L. Hillsman, H. Rosing-Vogelaar, A note comparing optimal and heuristic solutions to the p-median problem, Geographical Analysis 11 (1979) 86–89. doi:10.1111/j.1538-4632.1979.tb00674.x.

[38] A. P. Reynolds, G. Richards, B. de la Iglesia, V. J. Rayward-Smith, Clustering rules: A comparison of partitioning and hierarchical clustering algorithms, J. Math. Model. Algorithms 5 (2006) 475–504. doi:10.1007/s10852-005-9022-1.

[39] M. Tiwari, M. J. Zhang, J. Mayclin, S. Thrun, C. Piech, I. Shomorony, Banditpam: Almost linear time k-medoids clustering via multi-armed bandits, in: NeurIPS, 2020.

[40] L. Kaufman, P. J. Rousseeuw, Clustering large data sets, in: Pattern Recognition in Practice, Elsevier, 1986, pp. 425–437. doi:10.1016/b978-0-444-87877-9.50039-x.

[41] R. T. Ng, J. Han, Efficient and effective clustering methods for spatial data mining, in: Proc. 20th Int. Conf. Very Large Data Bases (VLDB'94), 1994, pp. 144–155.

[42] J. Podani, New combinatorial clustering methods, Vegetatio 81 (1989) 61–77. doi:10.1007/978-94-009-2432-1_5.

[43] M. Jambu, M.-O. Lebeaux, Cluster Analysis and Data Analysis, North-Holland Publishing Company, Amsterdam, 1983.

[44] J. Bien, R. Tibshirani, Hierarchical clustering with prototypes via minimax linkage, J. Am. Stat. Assoc. 106 (2011) 1075–1084. doi:10.1198/jasa.2011.tm10183.

[45] J. Newling, F. Fleuret, A sub-quadratic exact medoid algorithm, in: Proc. Int. Conf. Artificial Intelligence and Statistics, AISTATS, 2017, pp. 185–193. URL: http://proceedings.mlr.press/v54/newling17a.html.

[46] E. Schubert, A. Zimek, ELKI: A large open-source library for data analysis - ELKI release 0.7.5 "heidelberg", 2019. URL: https://elki-project.github.io/. arXiv:1902.03616.