Applied Computing and Intelligence

https://www.aimspress.com/journal/aci

Applied Computing and Intelligence
2 (1): 63–82
DOI: 10.3934/aci.2022004
Received: 18 March 2022
Accepted: 22 March 2022
Published: 31 March 2022

Research article

Optimal clustering by merge-based branch-and-bound

Pasi Fränti* **and Olli Virmajoki**

School of Computing, University of Eastern Finland, Joensuu, Finland
*** Correspondence:** franti@cs.uef.fi

Academic Editor: Chih-Cheng Hung

Abstract: We present a method to construct optimal clustering via a sequence of merge steps. We formulate the merge-based clustering as a minimum redundancy search tree, and then search the optimal clustering by a branch-and-bound technique. Optimal clustering is found regardless of the objective function used. We also consider two suboptimal polynomial time variants based on the proposed branch-and-bound technique. However, all variants are slow and has merely theoretical interest. We discuss the reasons for the results.

Keywords: Clustering; algorithms; optimal; agglomeration; branch-and-bound

1. Introduction

Clustering is a fundamental problem that must often be solved as a part of more complicated tasks in pattern recognition, image analysis and other fields of science and engineering [1−4]. Clustering is also needed for designing a *codebook* in vector quantization [4]. The clustering problem is defined here as follows. Given a set of N data points $X = \{x_1, x_2, \ldots, x_N\}$, partition the data set into M clusters so that a given objective function f is minimized.

The clustering problem in its combinatorial form has been shown to be *NP-hard* [5]. No polynomial time algorithm is known to find the globally optimal solution, and sub-optimal solutions are usually obtained by heuristic algorithms. Despite the known limitations implicated by the NP-completeness, solving the optimal clustering problem has theoretical interest that can provide insight to the problem itself. It might also have practical implications in the case of problem instances of limited size.

Agglomerative clustering is an approach for generating the clustering hierarchically. The clustering starts by initializing each data point as its own cluster. Two clusters are merged at each step and the process is repeated until the desired number of clusters is obtained. *Ward's method* [6] is

a method that selects the cluster pair to be merged so that it increases mean square error least. In the vector quantization context, this is known as the *pairwise nearest neighbor* (*PNN*) method due to [7]. In the rest of this paper, we denote it as the *PNN method*.

The *PNN* method is interesting here because of its conceptual simplicity and because of the optimality of the single merge step. This step reduces a given clustering from m clusters to m-1 clusters by minimizing the optimization function value. Even though the step is optimal, there is no guarantee of optimality of the final clustering resulting from a series of locally optimal merge steps. The main idea of the *PNN* method, however, can be generalized so that we do not optimize only a single merge but over multiple merge steps.

In this paper, we present an optimal clustering algorithm derived from the *PNN* method. It is easy to see that any clustering can be produced by a series of merge operations. Every merge reduces the number of clusters by one. It therefore takes exactly N-M steps to generate a clustering with M groups from the set of N points. Optimal clustering can be found by considering all the possible merge sequences and finding the one that minimizes the objective function. The idea can be implemented as a *branch-and-bound technique* that uses a search tree for finding the optimal clustering, and a suitable bounding criterion to cut out non-optimal branches of the tree.

The relation of the proposed method to the *PNN* method is demonstrated in Fig. 1. At the first step, all possible merges are generated. At the second step, the *PNN* method would continue from the locally optimal result whereas branch-and-bound technique will study all branches. The root of the search tree represents the case where all data points are assigned to their own clusters. At the level N-m, there are all possible solutions to m clusters. The final clustering with M groups is located at the level N-M. All branches of the tree must be generated to find the optimal clustering.

We consider also two sub-optimal variants that compromise the optimality but work in polynomial time. The first algorithm generates the search tree only down to a fixed limit. The best result of the fixed depth is then taken as the new starting point, and the same procedure is repeated until the desired number of clusters is reached. The second variant proceeds only one level in the tree along the path towards to the direction of the best solution within the search range. After this, a completely new search tree is generated for one level further, and the process is then repeated.

The rest of the paper is organized as follows. In Section 2, we give formal description of the clustering problem, and review previous literature of optimal clustering. In Section 3 we recall the *PNN* method. The branch-and-bound technique is introduced in Section 4. We first study the redundancy of the search tree in Section 4.1, and then present a method to generate minimum redundancy tree in Sections 4.2 and 4.3. A bounding criterion is formalized in Section 4.4 to cut irrelevant branches of the tree. Two suboptimal polynomial time variants are introduced in Sections 5. Experimental tests are made in Section 6, and conclusions drawn in Section 7. The proposed branch-and-bound method was first reported in a conference [8], and the polynomial time variants later in [9].

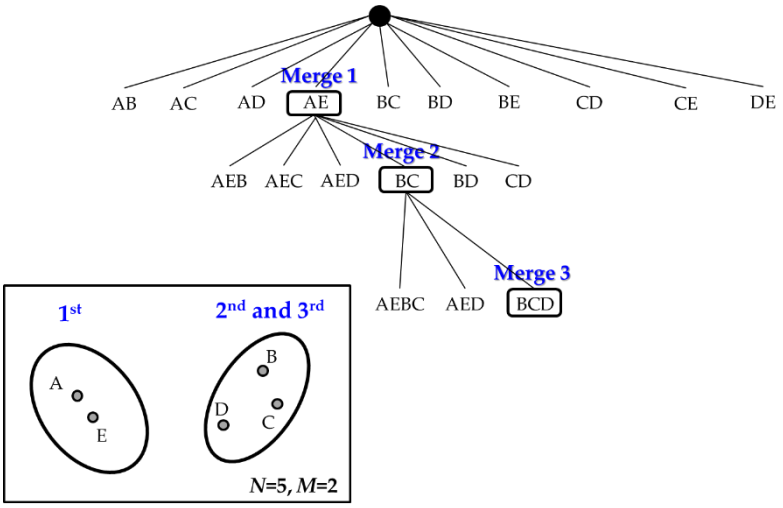


Figure 1. Illustration of the PNN method as a search tree.

2. Optimal clustering

Given a set of N data points $X=\{x_1, x_2, \ldots, x_N\}$, clustering aims at solving the partition $P = \{p_1, p_2, \ldots, p_N\}$, which defines for each data point the index of the cluster where it belongs to. *Cluster* s_a is defined as the set of data points that belong to the same partition a:

$$s_a = \left\{ x_i \middle| p_i = a \right\}. \tag{1}$$

Clustering is represented as the set $S=\{s_1, s_2, ..., s_M\}$, and a set of cluster centroids $C=\{c_1, c_2, \ldots, c_M\}$.

The most important choice in clustering is the objective function f for evaluating the goodness of clustering. When the data points belong to the Euclidean vector space, a commonly used function is the mean square error between the data points and their cluster centroids. Given a partition P and the cluster centroid C, it is calculated as:

$$MSE(C,P) = \frac{1}{N} \cdot \sum_{i=1}^{N} \left\| x_i - c_{p_i} \right\|^2. \tag{2}$$

The choice of the function depends on the application and there is no general solution of which measure should be used. However, once the objective function is decided the clustering problem can be formulated as a combinatorial optimization problem.

Optimal solution can be solved by constructing all $M^N/M!$ possible groupings of N data points into M groups and selecting the optimal one. This can be implemented by *brute force* by permuting all possible partitions of the data points. We refer this as *partition-based* approach.

This approach has been used for developing branch-and-bound technique by Koontz et al. [10]. They construct the partition by assigning data points to the clusters one by one, and by using a partial error for bounding non-optimal solutions. They have reported to solve problems of size N = 40, and of size N = 120 by compromising the optimality with a stronger bouncing criterion.

Cheng [11] finds clusters in a binary matrix where the values correspond to files/transactions relationships, and the goal is to organize the database for minimizing disc access. Problem size of 45×20 matrix has been solved by the method.

In [12], the clustering problem was formulated as a partitioning on edge-weighted graph, in which the goal is to minimize the sum of weights of the edges within the clusters [12]. Problem size of $N = 145$ has been considered.

Iyer and Aronson [13] proposed a parallel implementation with a linear speed-up with respect to the number of processors available. This increases the size of problems that is possible to solve by the algorithm, but only by a logarithmic factor.

Brusco [14] added data points to the solution one by one until all points have been added. Each point has M possible clusters to join, and branch-and-bound technique is used to prune the search tree. They managed to provide optimal solutions for problems with up to $N = 240$ objects in case of $M = 8$ well-separated clusters. For randomly distributed data optimal solution was found up to $N = 60$.

Since then, optimality of clustering has received very little attention. Guns et al. extended Brusco's approach using constrained programming [15]. Optimal clustering was studied in cutting and packing problem in 2-d space [16]. Optimal graph structure was presented in [17] for graph-based clustering. Missing-value problem was shown to fit nicely into the optimal clustering framework in [18]. All these results have mainly theoretical interest and do not generalize beyond small-scale instances.

Approximation algorithms have also been considered but with a limited success. Feder and Greene have shown that one cannot approximate optimal cluster size for fixed number of clusters in polynomial time size within a factor close to 2, unless $P = NP$ [19]. A solution with the time complexity of $O(N \log M)$ was then proposed. Mettu and Plaxton have proposed a randomized O(1)-approximation algorithm that works in $O(NM)$ time [20].

Polynomial time solutions are known but only for some special cases. For example, the clustering has been formulated as a graph problem in [21] using the assumption that the data set can be adequately represented by an adjacency graph. The method finds the partition that minimizes the maximum flow between the sub graphs (clusters) in the given partition. Efficient polynomial time solutions are known for this problem. The clustering problem involved in parallel scheduling [22] has been formulated as finding exact minimum *makespan* constrained by a tree structure.

It is also noted that efficient algorithm exists for the 1-dimensional special case. An example is the *scalar quantization* problem, for which $O(NM)$ solution is known [23].

3. Pairwise nearest neighbor method

The *pairwise nearest neighbor* (*PNN*) method [6,7] generates the clustering hierarchically using a sequence of merge operations as described in Fig. 2. In each step of the algorithm, the number of the clusters is reduced by merging two nearby clusters:

$$s_a \leftarrow s_a \cup s_b \,. \tag{3}$$

The cost of merging two clusters s_a and s_b is the increase in the MSE-value caused by the merge. It can be calculated using the following formula [7]:

$$d_{a,b} = \frac{n_a n_b}{n_a + n_b} \cdot \left\| c_a - c_b \right\|^2 \quad (4)$$

where n_a and n_b denote to the sizes of the corresponding clusters s_a and s_b.

PNN applies local optimization strategy: all possible cluster pairs are considered and the one increasing the objective function least is chosen:

$$a, b = \arg \min_{\substack{i,j \in [1,N] \\ i \neq j}} d_{i,j} \quad (5)$$

A single merge step of PNN is optimal but there is no guarantee of optimality of the final clustering resulting from a series of locally optimal merge steps. The time complexity of PNN varies from O(N^2) to O(N^3) depending on the implementation and data set [24].

```
PNN(X, M) → S
    FOR i←1 to N DO
        s_i ← {x_i};
    REPEAT
        (s_a, s_b) ← SearchNearestClusters(S);
        Merge(s_a, s_b);
    UNTIL |S|=M;
```

Figure 2. Structure of the PNN method.

4. Merge-based branch-and-bound technique

We described next a branch-and-bound technique that generates optimal clustering by a sequence of merge operations. It is easy to see that any clustering can be produced by merging the data points into the groups one by one. Every merge operation reduces the number of clusters by one. It therefore takes exactly *N*-*M* steps to generate a clustering with *M* clusters, independent of the order of the merge operations.

For example, consider the example shown in Fig. 1, in which we have five data points {A, B, C, D, E}. The resulting clustering can be generated by the following three merge operations:

Initial: {A} {B} {C} {D} {E}
Step 1: {AE} {B} {C} {D}
Step 2: {AE} {BC} {D}
Step 3: {AE} {BCD}

All alternative merge sequences can be represented as a *search tree*. The root of the tree represents the starting point in which every data point is assigned to its own cluster (*N* clusters), and its descendants represent all possible clusterings of *N*-1 clusters. In general, every node in the tree represents a single clustering with *m* clusters, and its children represent the clusterings that have been produced by merging any two of *m* existing clusters.

4.1. Redundancy of the search tree

The search tree includes a lot of redundant solutions (referred to *redundant search tree*) as the same partition can be constructed with different orders of the merge operations. The partitions are generated from the search tree as follows. At the first step, there are $N\cdot(N\text{-}1)/2$ alternatives for the merge operation, and therefore, equally many partitions at the level ($m = N$-1). Similarly, at the second level ($m = N$-2) there are $(N\text{-}1)\cdot(N\text{-}2)/2$ alternatives for the merge operation, and they are independent of the merge operation made at the previous level.

In general, every node has $(m)\cdot(m\text{-}1)/2$ children at the level with m clusters. We can therefore derive the total number of merge sequences, which lead to a partition of M clusters as follows:

$$Sequences(N,M)=\prod_{i=M+1}^{N}\frac{i\cdot(i-1)}{2}=\frac{1}{2^{N-M}}\frac{N!\cdot(N-1)!}{M!\cdot(M-1)!}\ . \tag{6}$$

At the same time, we know from [25] that the total number of different partitions equals to *Stirling's number of second kind* [26]:

$$Clusterings(N,M)=\left\{\begin{matrix}N\\M\end{matrix}\right\}=\frac{1}{M!}\sum_{i=1}^{M}(-1)^{M-i}\binom{M}{i}i^{N}\ . \tag{7}$$

We can calculate the average number of sequences per different partitions as:

$$\begin{aligned}\frac{Sequences(N,M)}{Clusterings(N,M)}&=\frac{\frac{1}{2^{N-M}}\frac{N!\cdot(N-1)!}{M!\cdot(M-1)!}}{\frac{1}{M!}\sum_{i=1}^{M}(-1)^{M-i}\binom{M}{i}i^{N}}\\&=\frac{\frac{N!\cdot(N-1)!}{2^{N-M}\cdot(M-1)!}}{\sum_{i=1}^{M}(-1)^{M-i}\binom{M}{i}i^{N}}\geq\frac{\frac{N!\cdot(N-1)!}{2^{N-M}\cdot(M-1)!}}{\sum_{i=1}^{M}\binom{M}{i}i^{N}}\\&\geq\frac{\frac{N!\cdot(N-1)!}{2^{N-M}\cdot(M-1)!}}{\sum_{i=1}^{M}\binom{M}{i}M^{N}}=\frac{\frac{N!\cdot(N-1)!}{2^{N-M}\cdot(M-1)!}}{M^{N}\cdot\sum_{i=1}^{M}\binom{M}{i}}\\&=\frac{N!\cdot(N-1)!}{2^{N-M}\cdot(M-1)!}\cdot\frac{1}{M^{N}\cdot(2^{M}-1)}\\&\geq\frac{N!\cdot(N-1)!}{2^{N}\cdot(M-1)!}\cdot\frac{1}{M^{N}}\geq\frac{N!}{2^{N}\cdot M^{N}}\\&\geq\frac{N!}{(2M)^{N}}\to\infty\ as\ N\to\infty\end{aligned}\ , \tag{8}$$

where N and M are nonnegative, and $N\geq M$. In other words, the search tree contains significant amount of redundant solutions.

4.2. Permutating non-redundant clusters

We consider next a single cluster represented as a list of the data points, and merge operation as the catenation of the two lists. For example, the clustering in Fig. 1 is represented as the pair of lists (AE) (BCD), and their merge as (AEBCD). Using this representation, we can see that the same cluster has several different representations. The cluster (BCD), for example, has the following representations:

(BCD)
(BDC)
(CBD)
(CDB)
(DBC)
(DCB)

The data points x_i can be ordered by their index i in the data set. We therefore use the following condition to prevent redundant representations of the same cluster.

Condition 1: *The only valid representation for a cluster* $s_j = \{x_1, x_2, ..., x_{nj}\}$ *is the ordered sequence* $(x_1\ x_2\ ...\ x_{nj})$.

For example, the only valid representation for {BCD} in the previous example is then (BCD). When using this condition, we can still represent all possible clusterings but without redundant representations for a single cluster.

The condition 1 can be applied in the *Branch-and-bound* algorithm using the following merge condition.

Condition 2: *Clusters* s_a *and* s_b *can be merged iff:* $i < j \forall x_i \in s_a, x_j \in s_b$.

Because of this *merge rule*, the order of the data points will be automatically retained. Yet, every possible cluster can be obtained by merging the data points starting to merge from the smallest index one by one.

For example, the cluster (ABE) is possible to obtain but only using the sequence that merges first (A) + (B), and then (AB) + (E). On the other hand, the cluster pair (AE) (B) cannot be merged because the resulting cluster (AE) + (B) = (AEB) is not a valid representation as the data points are not sorted. Furthermore, if the current clustering were (AE) (B) (C) (D), we could not merge the cluster (AE) with any other cluster anymore and (AE) would inevitably remain as such in the final clustering.

4.3. Permutating minimum redundancy search tree

The condition 2 removes the redundancy in the case of representing a single cluster but it is still possible to construct the same cluster via different *paths* (merge sequences) in the search tree. For example, the cluster (BCD) can be constructed using two different paths:

Sequence 1: (B) (C) (D) → (BC) (D) → (BCD)
Sequence 2: (B) (C) (D) → (B) (CD) → (BCD)

Furthermore, the clustering (AE) (BCD) can be reached by six different merge sequences, of which two are shown in Table 1.

Table 1: Example of generating clustering (AE) (BCD) via two different merge sequences.

Sequence 1:	**Sequence 2:**
(A) (B) (C) (D) (E)	(A) (B) (C) (D) (E)
(AE) (B) (C) (D)	(A) (BC) (D) (E)
(AE) (BC) (D)	(A) (BCD) (E)
(AE) (BCD)	(AE) (BCD)

It is therefore not enough to limit only the intra cluster representation, but we must also limit the permutation of the search paths in the tree. We do this by applying the following *permutation criterion*:

***Condition 3**: Clusters s_a and s_b can be merged iff* $a \geq a_0 \wedge b > a$,

where a_0 is the index of the first cluster in the previous merge. In other words, we force the algorithm to permutate the cluster pairs in a predefined order so that the index of the first cluster is always monotonically non-decreasing during the process. So, if we have merged clusters s_{a0} and s_{b0} at the previous level, we can consider only cluster pairs s_a and s_b such that $a \geq a_0$. The second term ($b > a$) is induced by condition 2.

We can see that any individual cluster $s_j=(s_{j1}\ s_{j2}\ s_{j3}\ ...\ s_{jn})$ is constructed by the following sequence of merge operations: $(s_{j1}) + (s_{j2}) \rightarrow (s_{j1}\ s_{j2}) + (s_{j3}) \rightarrow (s_{j1}\ s_{j2}\ s_{j3}) + (s_{j4})$, and so on. This fulfills the constraint $b > a$. Any clustering $\{s_1, s_2, ..., s_m\}$ can then be generated by constructing the clusters one by one in the order from s_1 to s_m. This fulfills the constraint $a \geq a_0$.

Furthermore, there is no other merge sequence that could construct the same clustering without contradicting the conditions 2 and 3. These conditions together guarantees that every cluster can be constructed in only one order, and the condition 2 ($a \geq a_0$) that the clusters are constructed in a unique sequence from smallest index to largest. Thus, the use of the conditions 2 and 3 produces non-redundant search tree.

In practice, the condition 2 can be too complicated to be implemented in practice. We therefore introduce the following condition that simplifies it.

***Condition 4**: Clusters s_a and s_b can be merged iff* $a = a_0 \Rightarrow b \geq b_0$.

This implication says that if we merge the same cluster s_a as previously, the index of the second cluster must be greater than that of the previously merged cluster s_{b0}. In other cases ($a > a_0$), the inequality $b > a$ from condition 3 is sufficient to satisfy also the condition 2.

The non-redundant search tree for the previous example is illustrated in Fig. 3. At the first level, the permutation creates the merges: (AB) (AC) (AD) (AE) (BC) (BD) (BE). We can see that after the merge (B)+(C), the merge (A)+(C) do not appear anymore because of condition 3, and it already exists in the branch where (AC) was constructed before (BD). Furthermore, if the previous merges created clusters (AC) and (BD), the cluster (ACBD) does not appear anymore as it has already been created by the sequence (A)+(B), (AB)+(C), (ABC)+(D).

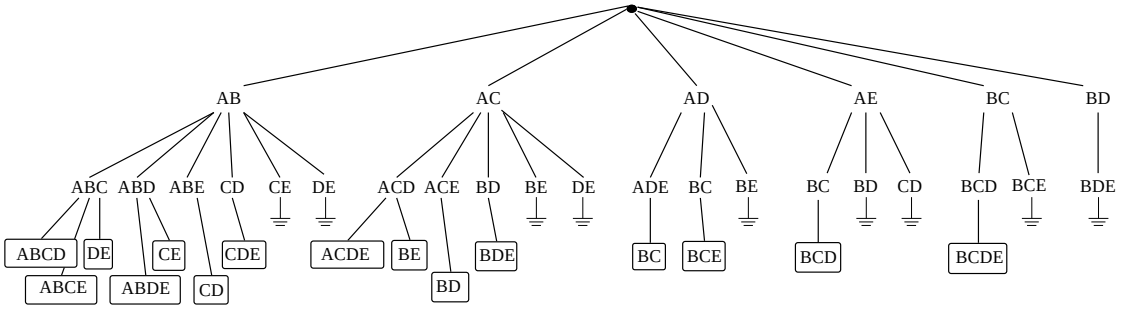


Figure 3. Example of non-redundant search tree.
Branches that do not have any valid clustering have been cut out.

The condition 3 removes redundant clusterings but there still exist partial branches that cannot be completed (resulting in too few clusters). For example, after the merge sequence (AB) and (CE), there would be no more valid merges left because the permutation criterion does not allow us to add new point in the cluster (AB) anymore, and because the merge (CE)+(D) would break the intra cluster order. Such branches can be eliminated using rather simple bounds for the permutation loop.

When we permutate new cluster pairs for merge, we always start to permute from the previous cluster s_{a0} and consider all potential pairs in a loop. Clusters pairs (s_a, s_b) to be considered are such that $1 \le a < b \le m$, where m is the current number of clusters. The index a also indicates how many clusters have been completed by now. This is because they are not allowed to be included in the merge operation any more due to condition 3.

The same applies also to the clusters that have index between a and b. Since we have m-M more merges to be performed, we know that there must be equally many valid cluster pairs. Concluding from this, we can derive the upper bound for the index a at any stage of the process.

Condition 5*: The first cluster index in the merge is upper limited to* $a \le M$.

Greater values than this would lead to situation that we cannot complete the clustering with M clusters. Using the conditions 1, 3, 4 and 5, we can now present the algorithm for generating non-redundant search tree as the pseudo code shown in Fig. 4.

4.4. Bounding criterion

The search can be terminated earlier when we know that the current branch cannot lead to a better solution than the best solution found so far. The termination is based on the fact that every merge operation increases the MSE-value of the solution, see Fig. 5. Thus, when we have generated the first solution in the search tree, we can use its MSE-value as the upper bound for the optimal solution. Other branches of the tree can then be terminated if the following condition is true:

Bounding criterion 1: $MSE_t \ge MSE_{\min}$,

where MSE_t is the value of the current solution after the t^{th} merge, and $MSE_{\min}$ is the value of the best solution found so far. We call this as a *simple bounding*.

```
Branch-and-bound(X, M) → S;
    FOR i←1 to N DO
        s_i ← {x_i};
    S, MSE_best ← BB(S, 1, 2, M);

BB(S_0, a_0, b_0, M) → S_best, MSE_best;
    MSE_best ← ∞;
    IF |S_0| = M THEN RETURN S_0, MSE(S_0);
    FOR a ← a_0 to M
        IF a=a_0    THEN    b_min ← b_0
                    ELSE    b_min ← a+1
        FOR b ← b_min to |S_0|
            S ← S_0;
            S ← Merge(S, s_a, s_b);
            S, mse ← BB(S, a, b, M);
            IF mse < MSE_best THEN
                MSE_best ← mse;
                S_best ← S;
            END-IF
        END-FOR
    END-FOR
    RETURN S_best, MSE_best;
```

Figure 4. Algorithm for generating non-redundant search tree for the optimal clustering.

It has been shown in [27] that the merge costs of the PNN method are monotonically increasing if the cluster pair with minimum cost is always merged. We denote the series of merge costs by d_1, d_2, ..., $d_{N\text{-}M}$, where d_t is the merge cost at the t^{th} merge. The monotony property implicates:

$$d_1 \le d_2 \le \ldots \le d_{N-M} . \tag{9}$$

The criterion has been shown to apply to the PNN method where we always select the merge with minimum cost. From this property, we could derive a stronger termination criterion for the branch-and-bound algorithm:

Bounding criterion 2: $MSE_{\mathrm{t}} + (N - M - t) \cdot d_t \ge MSE_{\min}$.

Here (N-M-t) indicates the number of forth coming merges in the algorithm, and d_t is the previous merge cost. This bounding criterion is based on the assumption that all forth-coming merge operations increase the MSE no less than the previous merge operation. This would allow termination of sub-optimal solution earlier than the criterion 1.

The only problem of using the bounding criterion 2 is that the monotony property does not necessarily hold true in the algorithm of Fig. 4. In the branch-and-bound method, we can also perform sub-optimal merges, which can result in a non-monotonic series of merge costs. Consequently, we could terminate a path to the optimal solution because of using the stronger criterion.

The optimal solution can be reached via several different search paths. It is expected that at least one of these paths fulfills the monotony property, and therefore, termination of other redundant paths would not be a big problem. The algorithm in Section 3.3, however, generates non-redundant search tree and there is no guarantee that only path to the optimal solution would meet the monotony

property. The consequence of this is that optimality cannot be guaranteed if the stronger termination criterion was used with the non-redundant search tree.

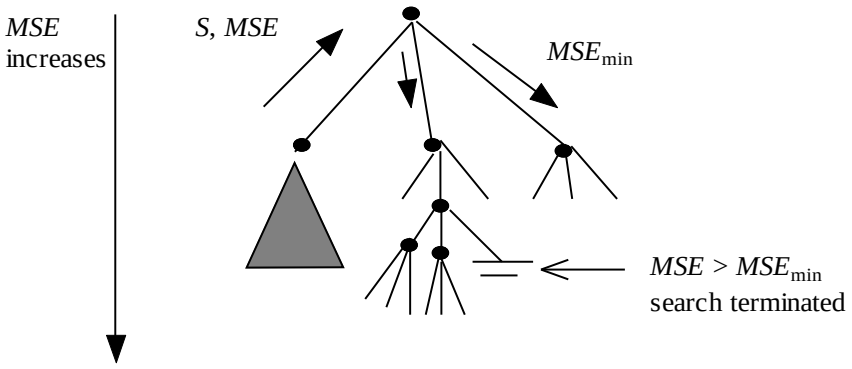


Figure 5. Illustration of the use of the bounding criterion.

5. Polynomial time variants

The time complexity of the branch-and-bound technique is exponential regardless the bounding criterion used. The practical usability of the method is therefore limited to small special cases only. We propose next two sub-optimal variants that compromise the optimality but work in polynomial time.

5.1 Piecewise optimization

The first method, called *Piecewise optimization*, divides the original problem into a series of smaller sub problems that are solved independently. The input at each stage of the algorithm is N clusters (whole data set in the beginning), and the output is the optimal clustering to N-Z clusters, where Z is a parameter of the algorithm. The result is then input to the same procedure, and the process is repeated until the desired number of M clusters is reached. The method is illustrated in Fig. 6, and its pseudo code given in Fig. 7.

If the size of the search tree of a single sub-problem is Z, we need to repeat the algorithm $\lceil (N\text{-}M)/Z \rceil$ times. The quality of the result depends on the parameter Z; greater values will give better clustering result at the cost of longer run time. The extreme case is when we set $Z = N\text{-}M$, which would result to the same algorithm as the branch-and-bound technique in Section 3. By setting $Z = 1$, on the other hand, the method would be the same as the PNN method.

At the starting point of the algorithm when we have N clusters as the input, we have $O(N^2)$ possible merge operations to be considered. Two subsequent merge operations result in $O(N^4)$ different alternatives. In general, the time complexity of z subsequent merge operations is $O(N^{2Z})$, and the overall *Piecewise algorithm* $(N/Z)\cdot O(N^{2Z}) = O(N^{2Z+1}/Z)$. The algorithm works in polynomial time if Z is small enough to be considered as a constant. On the other hand, the time complexity increases exponentially as a function of Z. The algorithm is therefore useful only with very small values such of Z. For example, the time complexities for $Z = 2$ and $Z = 3$ are $O(N^5)$ and $O(N^7)$, respectively.

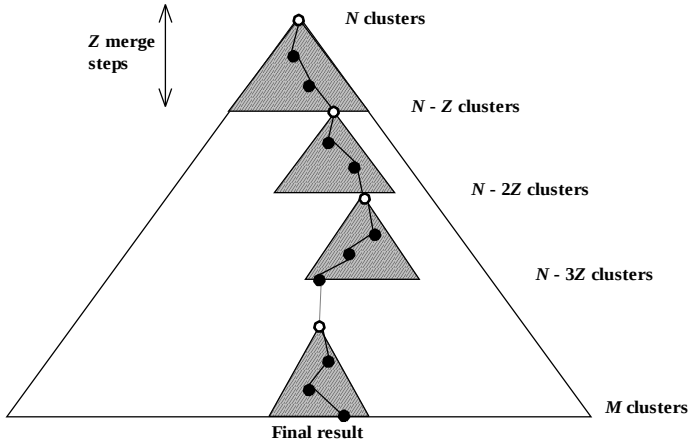


Figure 6. Illustration of the *Piecewise optimization*. The start points at each step are the white dots.

```
PiecewiseOptimization(X, M, Z) → S;
    FOR i←1 to N DO
        s_i ← {x_i};
    REPEAT
        size ← max(M, |S|-Z);
        S ← BB(S, 1, 2, size);
    UNTIL |S| = M;
```

Figure 7. Piecewise optimization algorithm.

5.2 Look-ahead optimization

The *Piecewise optimization* traverses the search tree through the local optima. At each stage of the algorithm, the last merge is the most critical because the algorithm considers only one level further. Slightly better (but Z times slower) variant, denoted as *Look-ahead optimization*, can be designed by the following modifications.

As in the *Piecewise optimization*, we generate complete search tree to the level Z and search for the optimal clustering with N-Z clusters. Instead of moving to this local optimum at the level Z, we proceed only one level in the tree along the path towards the direction of the local optimum. After this, we regenerate a completely new search tree starting from the level N-1, and then repeat the procedure N-M times. The process is illustrated in Fig. 8.

This variant is less critical for the local minima in practice. As a drawback, the time complexity of the algorithm is Z times that of the previous variant; that is $O(N^{2Z+1})$. With large Z-values, we would also do unnecessary work as a part of the search tree would be re-generated several times. In practice, the algorithm can be realized only with very small Z-values.

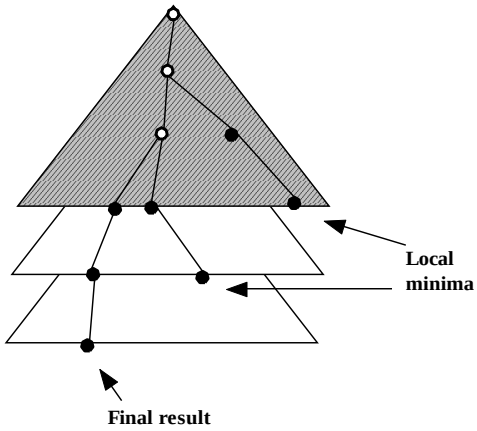


Figure 8. Illustration of the *Look-Ahead optimization*.
The starting points at each step are shown as white dots.

6. Experiments

We consider the data sets that are summarized in Table 2. The first set (B) includes a randomly selected pixel blocks from a 256 × 256 size image Bridge. The second set (S) is artificially generated two-dimensional data set with varying complexity in terms of spatial data distributions with $M = 15$ predefined clusters. The third set (SS2) is a standard clustering test problem of [25], pp. 103−104. The data set contains 89 postal zones in Bavaria (Germany) and their attributes are the number of self-employed people, civil servants, clerks and manual workers in these areas. The attributes are normalized to the scale [0, 1] according to their minimum and maximum values.

Table 2: Summary of the data sets used.

Data set	Type of data set:	Data size (N)	Clusters (M)	Dimensions
B	Random 4 × 4 blocks from gray-scale image *Bridge*.	3−20	2 / 5 / 9	16
S	Synthetically generated.	30−120	15	2
SS2	Attributes of postal zones in Bavaria, Germany.	89	7	4

The results with the B sets are summarized in Figs. 9−11 with different number of clusters (M = 2,5,9) using Pentium 450 MHz computer. Comparative results are given for the following methods:

- Partition-based full search (implementation by Juha Kivijärvi).
- Partial partition-based branch-and-bound [10]
- Merge-based full search
- Merge-based branch-and-bound

6.1 Results

Proposed methods work faster when the number of clusters is small (M = 2), or large (M = 9). These correspond to situations, in which the number of possible solutions is smallest. The use of bounding criterion improves the full search remarkably when the number of clusters is large (M = 5 and M = 9). In these cases, the proposed merge-based branch-and-bound is also faster than the partition-based counterpart. In the case of M = 2, on the other hand, the partition-based approach is superior. This is reasoned by the fact that the more merge steps are required the less there are clusters in the solution. We can also conclude that the practical usability of any of the optimal algorithms tested here are limited to very small size problem instances only.

Similar results are also reported in Fig. 12 for the polynomial time variants in the case of set *S*. The polynomial time variants are faster and can process larger data sets than the optimal branch-and-bound technique although only small sub tree sizes (Z = 2,3) were applied.

The qualities of the sub-optimal variants are compared in Table 3 with other heuristic methods. Random clustering is obtained by randomly selecting M data points and using them as cluster representatives, and then partition the data set optimally by minimizing Euclidean distance from the data points to the cluster representatives. K-means is an iterative clustering algorithm, which is also known as GLA in vector quantization context due to [28]. PNN is implemented as in [24], and GAIS refers to a genetic algorithm which is the best clustering algorithm that we experimented in [29].

In comparison to *PNN*, the polynomial time variants (*Piecewise* and *Look-ahead*) manage to produce better clustering in two cases (N = 70, 80) but the difference is marginal. For the rest of the data there are no differences between *PNN* and branch-and-bound variants. *GAIS*, on the other hand, gives somewhat better results than *PNN* and *BB*. It is noted that the optimality of these results is not known but it is expected that *GAIS* results are very close to optimum.

The results for SS2 are summarized in Table 4. They indicate that the *Piecewise* and *Look-ahead* algorithms can improve over *PNN* and *K-means* but at the cost of significant increase in run time. The results are still worse than that of *GAIS*. The additional time spent for the improvement is simply not worth as better results can be obtained sophisticated heuristics like genetic algorithm [29] much faster.

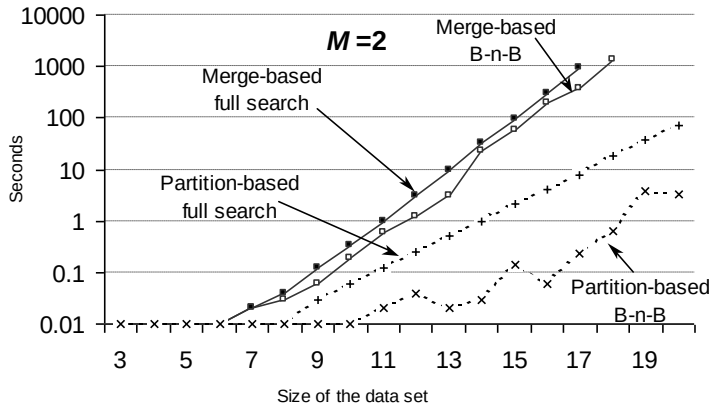


Figure 9. The effect of problem size to the running time (M = 2) for the set *B*.

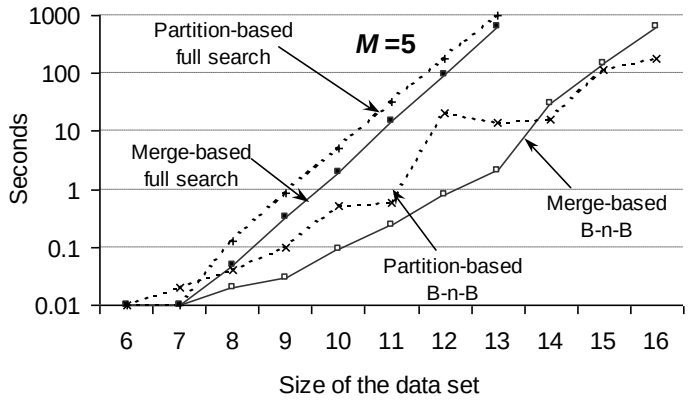


Figure 10. The effect of problem size to the running time (M = 5) for the set *B*.

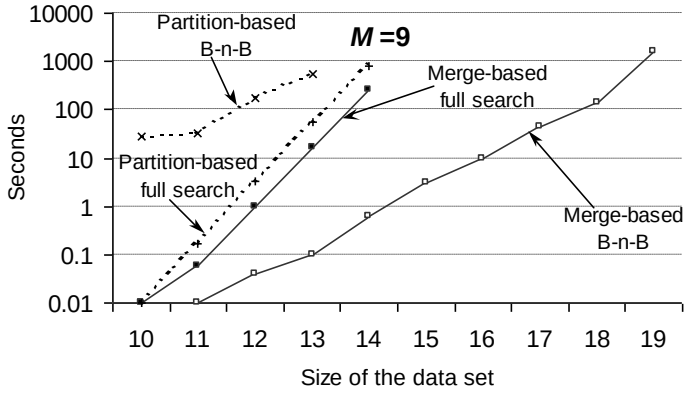


Figure 11. The effect of problem size to the running time (M = 9) for the set *B*.

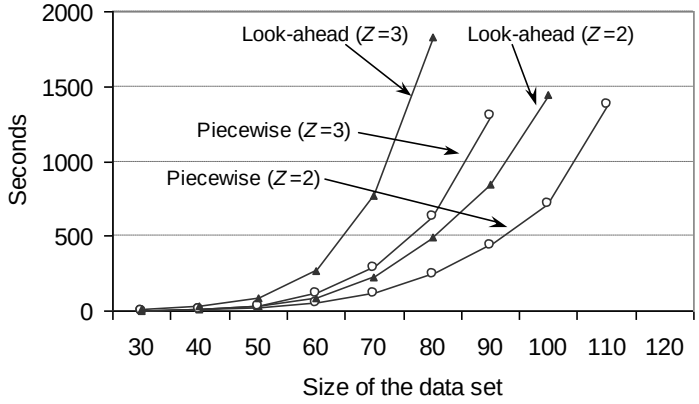


Figure 12. The effect of problem size to the running time of the polynomial time variants (M = 15) for the set *S*.

Table 3: Performance comparison for the simulated data set (N = 30..100, M = 15) for the set S. The values are mean square errors ($\times 10^9$).

Method:	***N* = 30**	***N* = 40**	***N* = 50**	***N* = 60**	***N* = 70**	***N* = 80**	***N* = 90**	***N* = 100**
Random clustering	2.996	2.914	7.093	4.181	5.399	4.940	4.545	4.062
K-means [28]	0.948	1.081	1.108	1.541	1.647	1.614	1.628	1.837
PNN [24]	0.396	0.573	0.992	1.181	1.246	1.225	1.274	1.373
BB: Piecewise (Z=2)	0.396	0.573	0.992	1.181	1.240	1.206	1.274	1.373
BB: Look-ahead (Z=2)	0.396	0.573	0.992	1.181	1.240	1.206	1.274	1.373
GAIS [29]	0.396	0.573	0.992	1.141	1.189	1.169	1.238	1.335

Table 4: Performance comparison for the data set SS2 (N = 89, M = 7) from [24]. The values are mean square errors ($\times 10^9$).

Method:	**Error:**	**Time:**
Random clustering	1.760	<1 s
K-means [28]	1.140	<1 s
PNN [24]	0.336	<1 s
BB: Piecewise (Z = 2)	0.323	342 s
BB: Piecewise (Z = 3)	0.336	1061 s
BB: Piecewise (Z = 4)	0.323	4851 s
BB: Look-ahead (Z = 2)	0.323	652 s
BB: Look-ahead (Z = 3)	0.316	3129 s
GAIS [29]	0.313	<1 s

6.2 Discussion

Here we discuss the pros and cons of the studied merge-based branch-and-bound approach and compare it against alternative splitting approach. This approach puts all the points in a single cluster and then splits one cluster at a time until M clusters are reached. These two approaches are compared in Fig. 13.

The main benefit of the merge-based approach is that the search tree is narrower. There are only $O(N^2)$ possible merges at each step compared to $O(2^N)$ of possible splits at every step. The split approach would also require minimum redundancy search tree to avoid the same clustering result to be found via different order of splits. The merge-based search tree is also deeper as there are $O(N)$ merges in total compared to $O(M)$ of split. Theoretically this provides more effective branch-and-bound as each successful bound would cut bigger proportion of the tree while the opposite splitting approach reaches candidate solutions faster in M steps.

However, a bigger problem is that the last merge is the costliest of all merges. For example, MSE-values for M = (30, 20, 10, 7, 2) are (0.036, 0.070, 0.191, 0.314, 4.086). The early merges are therefore insignificant compared to the ones at the end, and sub-optimal merges at the earlier stages rarely trigger the bounding criterion. The situation of the splitting approach would be opposite.

When compared to the results in literature, we see that Brusco's method [14] found optimal solutions up to N = 60 with M = 8 clusters in about 6½ hours. This is significantly larger than any of our results which remained smaller than N = 20. The bounding criterion of the merge-based approach is too poor because of the increasing nature of the merge errors. The opposite splitting-based approach is expected to work better due to this reason.

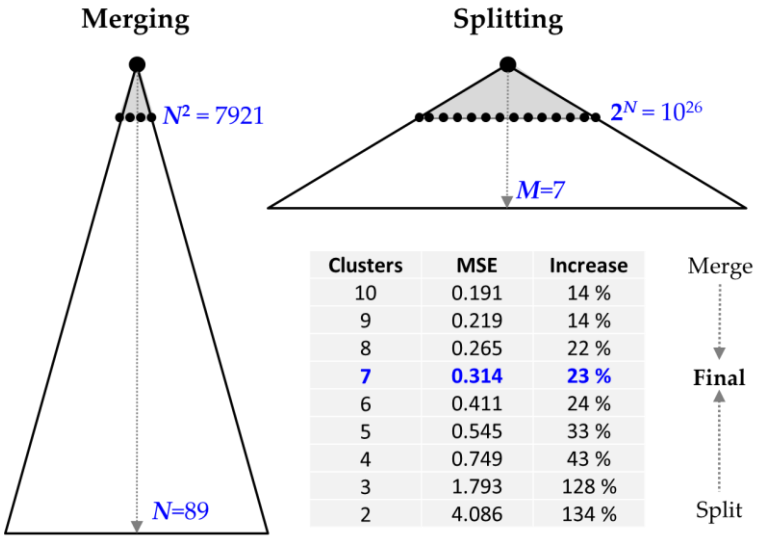
One factor for better efficiency is the nearest neighbor separation in [14], which places the points with smallest pairwise distance at the opposite ending in the ordering. The idea is to have diverse dataset early where *disruptive* points appear early in the ordering. This makes even smaller mistakes to trigger the bounding criterion. Kaminka [15] enhanced this further by applying furthest point heuristic by Gonzalez [30]. This heuristic (also known as *minmax*) has been shown to provide much better initialization for k-means than random centroids in [31].

Another important factor is to choose the initial solution well. In the context of travelling salesperson problem, an undocumented rule of thumb states that the initial solution should be as good as 1% for the branch-and-bound to be significantly more efficient to exhaustive search according to [32]. In our branch-and-bound we have used the PNN algorithm but the results for SS2 (0.336 with M = 7) are actually 2.6% worse than obtained by GAIS (0.314).

Brusco [14] also reached optimal solutions for problems size up to N = 240 with M = 8 clusters in case of well-separated clusters. However, such data is significantly easier, and we can expect to reach optimal results by a simple but effective heuristic algorithm called random swap [33] even for higher values of N. The only purpose of branch-and-bound would then be merely to verify the optimality of the result.

Finally, we note that our results have been ran already in 2004 using 450 MHz computer, around the same time that those of Brusco and others. At first sight one might expect these results being already quite old. However, our current machine runs in 3.7 GHz which is merely 8:1 increase in speed from 2004 to 2022 (see also Fig. 14). Since the problem has exponential in nature, time complexity of 2^N would mean that we could solve problems of size N+3 in 2022 compared to problem of sizes N in 2004. This is hardly significant.

Our machine itself has 20 cores available for single test research experiments. While branch-and-bound indeed would parallelize nicely, this would mean another 20:1 increase, so we could hope to solve problems of size *N*+7.



Clusters	MSE	Increase
10	0.191	14 %
9	0.219	14 %
8	0.265	22 %
7	**0.314**	**23 %**
6	0.411	24 %
5	0.545	33 %
4	0.749	43 %
3	1.793	128 %
2	4.086	134 %

Figure 13. Comparison of using merge and split in the branch-and-bound. The data is from SS2 dataset when using greedy PNN algorithm.

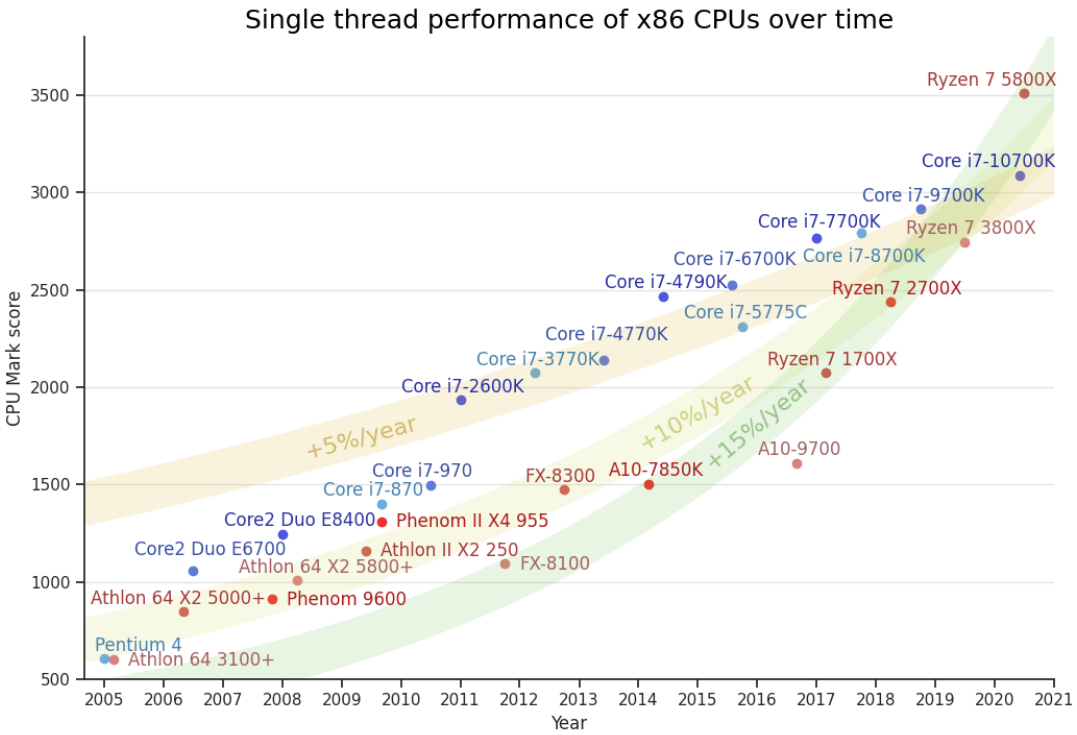


Figure 14. Evolution of single-thread processor performance from 2005 to 2021. Source: https://mlech26l.github.io/pages/2020/12/17/cpus.html

7. Conclusions

We have introduced a merge-based branch-and-bound technique to solve optimal clustering. The proposed algorithm works better than the partition-based counterparts when the number of clusters is high. In the case of a small number of clusters, however, the partition-based approach works faster. Nevertheless, all variants have exponential time complexity, and therefore, the results are mainly of theoretical interest only.

Two polynomial time algorithms were also introduced inspired by the proposed branch-and-bound technique but with limited success. Further improvement could be achieved by using stronger bounding criteria in the polynomial time variants.

References

1. B. S. Everitt, *Cluster Analysis*, 3rd ed., Edward Arnold, London / Halsted Press, New York, 1993. ISBN 978-0340584798 / 978-0470220436
2. L. Kaufman, P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*, John Wiley Sons, New York, 1990. https://doi.org/10.1002/9780470316801
3. R. Dubes, A. Jain, *Algorithms for Clustering Data*, Prentice-Hall, Englewood Cliffs, NJ, 1988. ISBN 978-0-13-022278-7
4. A. Gersho, R. M. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic Publishers, Dordrecht, 1992. https://doi.org/10.1007/978-1-4615-3626-0
5. M. R. Garey, D. S. Johnson, H. S. Witsenhausen, The complexity of the generalized Lloyd-Max problem, *IEEE Trans. Inf. Theory*, **28** (1982), 255−256. https://doi.org/10.1109/TIT.1982.1056488
6. J. H. Ward, Hierarchical grouping to optimize an objective function, *J. Amer. Statist. Assoc.*, **58** (1963), 236−244. https://doi.org/10.1080/01621459.1963.10500845
7. 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
8. P. Fränti, O. Virmajoki, T. Kaukoranta, Branch-and-bound technique for solving optimal clustering, *Int. Conf. on Pattern Recognition (ICPR'02)*, Québec, Canada, **2** (2002), 232−235. https://doi.org/10.1109/ICPR.2002.1048281
9. P. Fränti, O. Virmajoki, Polynomial-time clustering algorithms derived from branch-and-bound technique, *Advanced Concepts for Intelligent Vision Systems (ACIVS'2002)*, Gent, Belgium, (2002), 118−123.
10. W. L. G. Koontz, P. M. Narendra, K. Fukunaga, A branch and bound clustering algorithm, *IEEE Trans. Comput.*, **24** (1975), 908−915. https://doi.org/10.1109/T-C.1975.224336
11. C.-H. Cheng, A branch and bound clustering algorithm, *IEEE Trans. SMC*, **25** (1995), 895−898. https://doi.org/10.1109/21.376504
12. G. Palubeckis, A branch-and-bound approach using polyhedral results for a clustering problem, *INFORMS Journal of Computing*, **9** (1997), 30−42. https://doi.org/10.1287/ijoc.9.1.30
13. L. S. Iyer, J. E. Aronson, A parallel branch-and-bound method for cluster analysis, *Ann. Oper. Res.*, **90** (1999), 65−86. https://doi.org/10.1023/A:1018925018009
14. M. J. Brusco, A repetitive branch-and-bound procedure for minimum within-cluster sums of squares partitioning, *Psychometrika*, **71** (2006), 347−363. https://doi.org/10.1007/s11336-004-1218-1

15. G. Kaminka, Repetitive branch-and-bound using constraint programming for constrained minimum sum-of-squares clustering, *European Conf. on Artificial Intelligence, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*, **285** (2016), 462−470. IOS Press. https://doi.org/10.3233/978-1-61499-672-9-462
16. J. Bennell, G. Scheithauer, Y. Stoyan, T. Romanova, A. Pankratov, Optimal clustering of a pair of irregular objects, *J. Glob. Optim.*, **61** (2015), 497−524. https://doi.org/10.1007/s10898-014-0192-0
17. Y. Han, L. Zhu, Z. Cheng, J. Li, X. Liu, Discrete optimal graph clustering, *IEEE Trans. Cybern.*, **50** (2018), 1697−1710. https://doi.org/10.1109/TCYB.2018.2881539
18. S. Boluki, S. Z. Dadaneh, X. Qian, E. R. Dougherty, Optimal clustering with missing values, *BMC bioinformatics*, **20** (2019), 1−10. https://doi.org/10.1186/s12859-019-2832-3
19. T. Feder, D. Greene, Optimal algorithms for approximate clustering, *ACM Symposium on Theory of Computing*, (1988), 434−444. https://doi.org/10.1145/62212.62255
20. R. R. Mettu, C. G. Plaxton, Optimal time bounds for approximate clustering, *Machine Learning*, **56** (2004), 35−60. https://doi.org/10.1023/B:MACH.0000033114.18632.e0
21. Z. Wu, R. Leahy, An optimal graph theoretic approach to data clustering: theory and its application to image segmentation, *IEEE T. Pattern Anal.*, **15** (1993), 1101−1113. https://doi.org/10.1109/34.244673
22. L. Gao, A. L. Rosenberg, R. K. Sitaraman, Optimal clustering of tree-sweep computations for high-latency parallel environments, *IEEE T. Parall. Distr.*, **10** (1999), 813−824. https://doi.org/10.1109/71.790599
23. X. Wu, Optimal quantization by matrix searching, *J. Algorithms*, **12** (1991), 663−673. https://doi.org/10.1016/0196-6774(91)90039-2
24. P. Fränti, T. Kaukoranta, D.-F. Shen, K.-S. Chang, Fast and memory efficient implementation of the exact PNN, *IEEE Trans. Image Process.*, **9** (2000), 773−777. https://doi.org/10.1109/83.841516
25. H. Späth, *Cluster Analysis Algorithms for Data Reduction and Classification of Objects*, Ellis Horwood Limited, West Sussex, 1980. ISBN 978-0853121411
26. R. L. Graham, D. E. Knuth, O. Patashnik, *Concrete Mathematics – a Foundation for Computer Science*, 2nd ed., Addison-Wesley, 1994. ISBN 978-0201558029
27. T. Kaukoranta, P. Fränti, O. Nevalainen, Vector quantization by lazy pairwise nearest neighbor method, *Optical Engineering*, **38** (1999), 1862−1868. https://doi.org/10.1117/1.602251
28. Y. Linde, A. Buzo, R. M. Gray, An algorithm for vector quantizer design, *IEEE Trans. on Comm.*, **28** (1980), 84−95. https://doi.org/10.1109/TCOM.1980.1094577
29. 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
30. T. F. Gonzalez, Clustering to minimize the maximum intercluster distance, *Theor. Comput. Sci.*, **38** (1985), 293−306. https://doi.org/10.1016/0304-3975(85)90224-5
31. P. Fränti, S. Sami, How much can k-means be improved by using better initialization and repeats? *Pattern Recogn.*, **93** (2019), 95−112. https://doi.org/10.1016/j.patcog.2019.04.014
32. P. Fränti, N. Teemu, M. Yuan, Converting MST to TSP path by branch elimination, *Applied Sciences*, **11** (2020), 177. https://doi.org/10.3390/app11010177
33. P. Fränti, Efficiency of random swap clustering, *Journal of Big Data*, **5** (2018) 1−29. https://doi.org/10.1186/s40537-018-0122-y

AIMS Press