Optimal clustering by merge-based branch-and-bound

(submitted for publication 5.10.2004)

Pasi Fränti and Olli Virmajoki

Department of Computer Science, University of Joensuu
P.O. Box 111, FIN-80101 Joensuu, FINLAND
franti@cs.joensuu.fi, ovirma@cs.joensuu.fi

Abstract: We propose a method for constructing 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. The result has theoretical interest and it can provide new insight to the problem itself. We introduce two suboptimal but polynomial time variants based on the proposed branch-and-bound technique.

Keywords: Clustering, algorithms, agglomeration, vector quantization.

Statistics: 18 pages, 12 figures, 4 tables, 5878 words, 29648 characters.

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, 2, 3, 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 vectors $X=\{x_1, x_2, \ldots, x_N\}$, partition the data set into M clusters such that a given distortion 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 vector 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] selects the cluster pair to be merged so that it increases the given objective function value 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* vectors. Optimal clustering can be found by considering all the possible merge sequences and finding the one that minimizes the distortion 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 of two vectors 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 vectors are assigned to their own clusters. At the level *N*-*m*, there are all possible clusterings to *m* clusters. The final clustering with *M* groups is located at the level *N*-*M*. All branches of the tree must be generated in order 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, review previous literature, and recall the *PNN* method. The branch-and-bound technique is introduced in Section 3. We first study the redundancy of the search tree in Section 3.1, and then present a method to generate minimum redundancy tree in Sections 3.2 and 3.3. A bounding criterion is formalized in Section 3.4 in order to cut irrelevant branches of the tree. Two suboptimal polynomial time variants are introduced in Sections 4. Experimental tests are made in Section 5, and conclusions drawn in Section 6. The proposed branch-and-bound method was first reported in a conference [8], and the polynomial time variants later in [9].

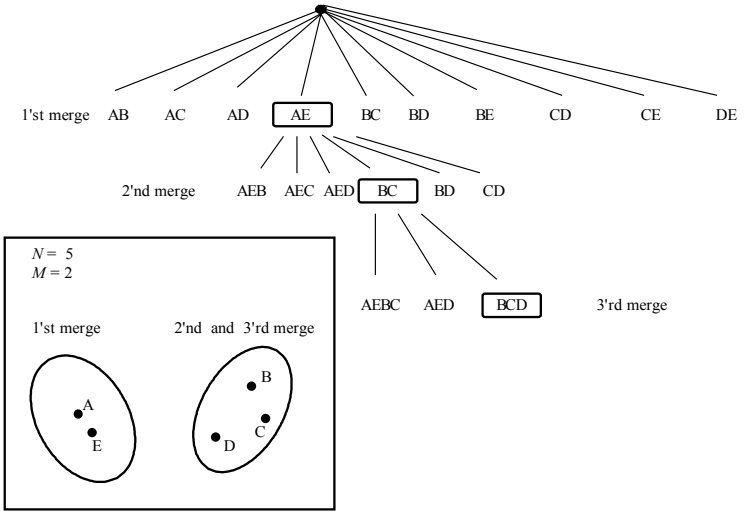

Fig. 1. Illustration of the *PNN* method as a search tree.

2. Pairwise nearest neighbor method

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

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

Clustering is then represented as the set $S=\{s_1, s_2, ..., s_M\}$. In vector quantization, the output of clustering is a codebook $C=\{c_1, c_2, \ldots, c_M\}$,

The most important choice in clustering is the cost function f for evaluating the goodness of clustering. When the data objects belong to the Euclidean vector space, a commonly used function is the mean square error between the data objects and their cluster centroids. Given a partition P and the cluster representatives 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.

2.1 Optimal clustering

Optimal solution can be solved by constructing all $M^N/M!$ possible groupings of N data vectors into M groups, and selecting the optimal one. This can be implemented by *brute force* by permuting all possible partitions of the data vectors. 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 vectors to the clusters one by

one, and use a partial distortion 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.

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* [14]. 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 [15].

Polynomial time solutions are known but only for some special cases. For example, the clustering has been formulated as a graph problem in [16] 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 [17] 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 [18].

2.2 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]:

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

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

The PNN applies local optimization strategy: all possible cluster pairs are considered and the one increasing the distortion 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 the *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 the *PNN* varies from O(N^2) to O(N^3) depending on the implementation and data set [19].

```
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;
```

Fig. 2. Structure of the *PNN* method.

3. 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 vectors 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 vector 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.

3.1 Redundancy of the search tree

The search tree includes a lot of redundancy as the same clustering can be constructed with many different orders of the merge operations. The clusterings are generated from the search tree as follows. At the first step, there are $N \cdot (N-1)/2$ alternatives for the

merge operation, and therefore, equally many clusterings at the level (m=N-1). Similarly, at the second level (m=N-2) there are (N-1)·(N-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)·(m-1)/2 children at the level with m clusters. We can therefore derive the total number of merge sequences, which lead to clustering of M clusters as follows:

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

At the same time we know from [20] that the total number of different clusterings equals to *Stirling's number of second kind* [21]:

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

We can calculate for N data items the average number of sequences per different clusterings of size M as:

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

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

3.2 Permuting non-redundant clusters

We consider next a single cluster represented as a list of the data vectors, 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 vectors 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: $i < j \forall x_i \in s_a, x_j \in s_b$.*

As a consequence of this *merge rule*, the order of the data vectors will be automatically retained. Yet, every possible clusters can be obtained by merging the data vectors starting to merge the vectors from the smallest 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 vectors are not sorted. Furthermore, if the current clustering were (AE) (B) (C) (D), we could not merge the cluster (AE) with any other cluster any more and (AE) would inevitably remain as such in the final clustering.

3.3 Permuting 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 sequence.

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 permute 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 guarantee 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 any more 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 any more as it has already been created by the sequence (A)+(B), (AB)+(C), (ABC)+(D).

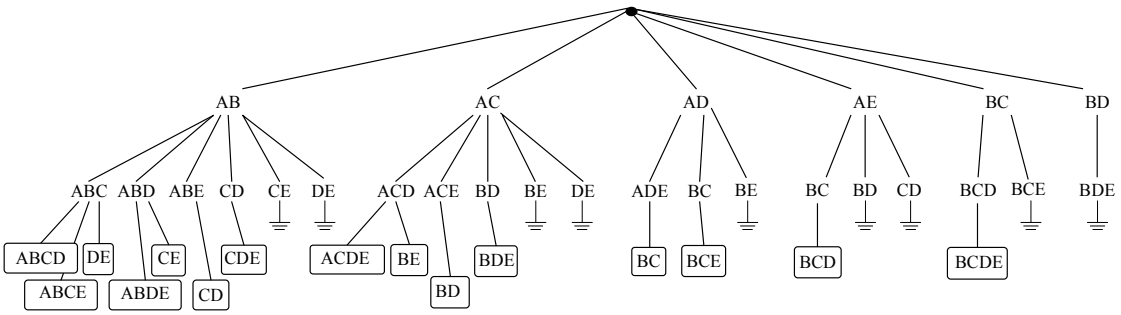

Fig. 3. Example of non-redundant search tree. Branches that do not have any valid clustering have been cut out.

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 vectors in the cluster (AB) any more, 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 permute new cluster pairs for merge, we always start to permute from the pervious 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.

```
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;
```

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

3.4 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: $MSE_t \geq MSE_{\min}$,

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*.

It has been shown in [22] 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 $d_1, d_2, \ldots, d_{N-M}$, where d_t is the merge cost at the t^{th} merge. The monotony property implicates:

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

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: $MSE_t + (N - M - t) \cdot d_t \geq MSE_{\min}$.

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. As a consequence, 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.

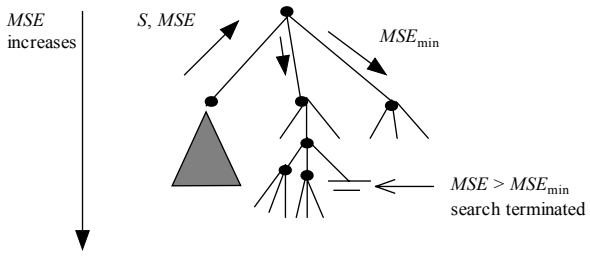

Fig. 5. Illustration of the use of the bounding criterion.

4. 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.

4.1 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-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.

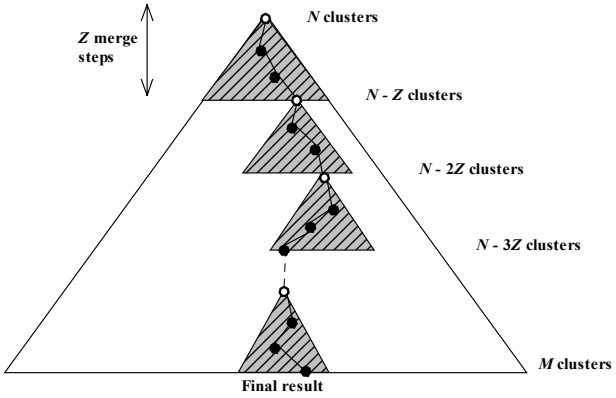

Fig. 6. Illustration of the *Piecewise optimization*.
The starting points at each step are shown as white dots.

```
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;
```

Fig. 7. Piecewise optimization algorithm.

4.2 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.

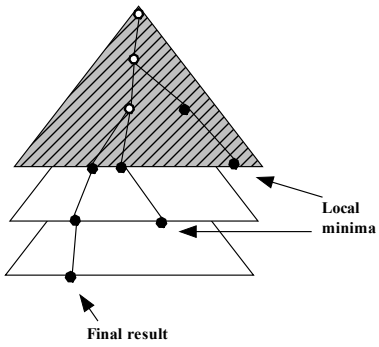

Fig. 8. Illustration of the *Look-Ahead optimization*.
The starting points at each step are shown as white dots.

5. 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 [20], 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:	Number of vectors (*N*)	Number of clusters (*M*)	Dimensionality
Set B	Random 4×4 blocks from gray-scale image *Bridge*.	3-20	2 / 5 / 9	16
Set 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 Fig. 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

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 counter-part. 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 is 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 clustering methods. The *random clustering* is obtained by randomly selecting *M* data vectors and using them as cluster representatives, and then partition the data set optimally by minimizing Euclidean distance from the data vectors to the cluster representatives. *K-means* is an iterative clustering algorithm, which is also known as the *GLA* in vector quantization context due to [23]. The *PNN* is implemented as in [19], and the *GAIS* refers to the best clustering method that we have experimented [24].

In comparison to the *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 the *PNN* and branch-and-bound variants. The *GAIS*, on the other hand, gives somewhat better results than the *PNN* and *BB*. It is noted that the optimality of these results is not known but it is expected that the *GAIS* results are very close to optimum.

The results for the set SS2 are summarized in Table 4. They indicate that the *Piecewise* and *Look-ahead* algorithms can improve over the *PNN* and the *K-means* but at the cost

of significant increase in run time. The results, however, are still worse than that of the *GAIS*.

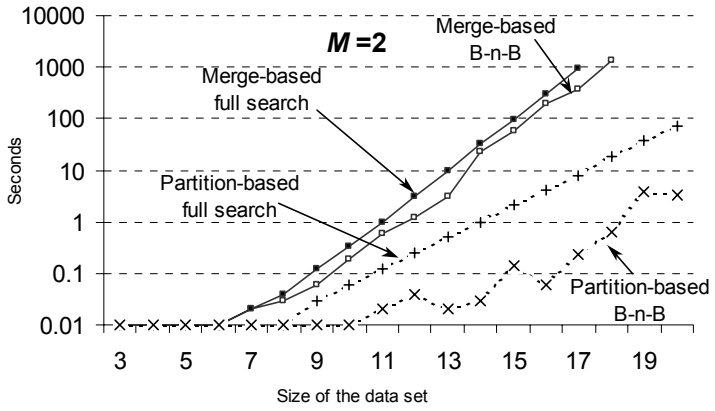

Fig. 9. The effect of problem size to the running time (*M*=2) for the set *B*.

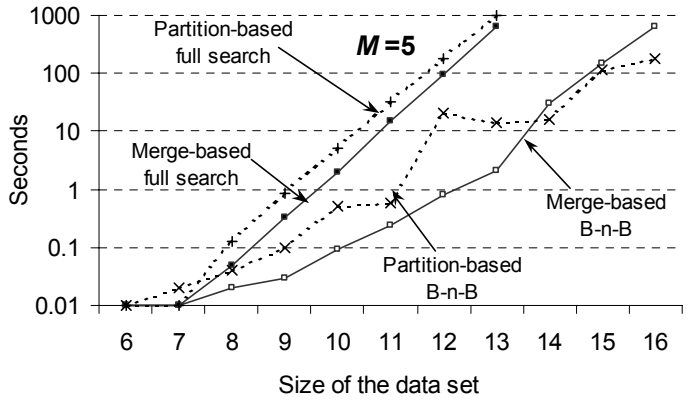

Fig. 10. The effect of problem size to the running time (*M*=5) for the set *B*.

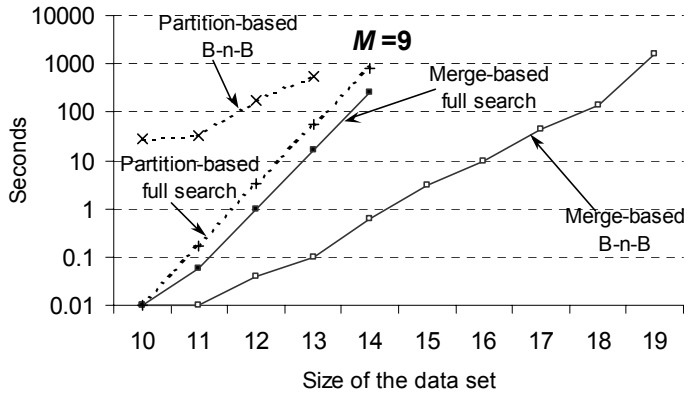

Fig. 11. The effect of problem size to the running time (*M*=9) for the set *B*.

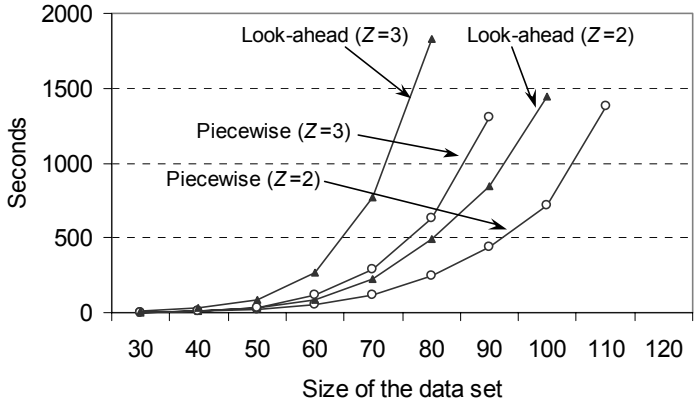

Fig. 12. The effect of problem size to the running time of the polynomial time variants (M=15) for the set S.

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 [23]	0.948	1.081	1.108	1.541	1.647	1.614	1.628	1.837
PNN [19]	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 [24]	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 [20]. The values are mean square errors ($\times 10^9$).

Method:	**Error:**	**Time:**
Random clustering	1.760	< 1 s
K-means [23]	1.140	< 1 s
PNN [19]	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 [24]	0.313	< 1 s

6. Conclusions

We have introduced a merge-based branch-and-bound technique to solve optimal clustering. The proposed algorithm works better than the partition-based counter parts 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] Everitt, B.S. (1992) *Cluster Analysis* (3rd edition). Edward Arnold / Halsted Press, London.

[2] Kaufman, L. and Rousseeuw, P.J. (1990) *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley Sons, New York.

[3] Dubes, R. and Jain, A. (1987) *Algorithms that Cluster Data*. Prentice-Hall, Englewood Cliffs, NJ.

[4] Gersho, A. and Gray, R.M. (1992) *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, Dordrecht.

[5] Garey, M.R., Johnson, D.S and Witsenhausen, H.S. (1982) The complexity of the generalized Lloyd-Max problem. *IEEE Transactions on Information Theory*, 28, 255-256.

[6] Ward, J.H. (1963) Hierarchical grouping to optimize an objective function. *J. Amer. Statist.Assoc.*, 58, 236-244.

[7] Equitz, W.H. (1989) A new vector quantization clustering algorithm. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37, 1568-1575.

[8] Fränti, P., Virmajoki, O. and Kaukoranta, T. (2002) Branch-and-bound technique for solving optimal clustering. *Int. Conf. on Pattern Recognition (ICPR'02*), Québec, Canada, 11–15 August, vol. 2, pp. 232-235.

[9] Fränti, P. and Virmajoki, O. (2002) Polynomial-time clustering algorithms derived from branch-and-bound technique. *Advanced Concepts for Intelligent Vision Systems (ACIVS'2002)*, Gent, Belgium, 9-11 September, pp. 118-123.

[10] Koontz, W.L.G., Narendra, P.M. and Fukunaga, K. (1975) A branch and bound clustering algorithm. *IEEE Transactions on Computers*, 24, 908-915.

[11] Cheng, C.-H. (1995) A branch and bound clustering algorithm. *IEEE Transactions on Systems, Man, and Cybernetics*, 25, 895-898.

[12] Palubeckis, G. (1997) A branch-and-bound approach using polyhedral results for a clustering problem. *INFORMS Journal of Computing*, 9, 30-42.

[13] Iyer, L.S. and Aronson, J.E. (1999) A parallel branch-and-bound method for cluster analysis. *Annals of Operations Research*, 90, 65-86.

[14] Feder, T. and Greene, D. (1988) Optimal algorithms for approximate clustering. *ACM Symposium on Theory of Computing*, Chicago, Illinois, 2-4 May, pp. 434-444. ACM Press.

[15] Mettu, R.R. and Plaxton, C.G. (2002) Optimal time bounds for approximate clustering. *18th Conf. on Uncertainty in Artificial Intelligence*, 1-4 August, pp. 344-351.

[16] Wu, Z. and Leahy, R. (1993) An optimal graph theoretic approach to data clustering: theory and its application to image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15, 1101-1113.

[17] Gao, L., Rosenberg, A.L. and Sitaraman, R.K. (1999) Optimal clustering of tree-sweep computations for high-latency parallel environments. *IEEE Transactions on Parallel and Distributed Systems* 10, 813-824.

[18] Wu, X. (1991) Optimal quantization by matrix searching. *Journal of Algorithms,* 12, 663-673.

[19] Fränti, P., Kaukoranta, T., Shen, D.-F. and Chang, K.-S. (2000) Fast and memory efficient implementation of the exact PNN. *IEEE Transactions on Image Processing*, 9, 773-777.

[20] Späth, H. (1980) *Cluster Analysis Algorithms for Data Reduction and Classification of Objects*. Ellis Horwood Limited, West Sussex, UK.

[21] Graham, R.L., Knuth, D.E. and Patashnik, O. (1994) *Concrete Mathematics – a Foundation for Computer Science* (2nd edition). pp. 257-267, Addison-Wesley.

[22] Kaukoranta, T., Fränti, P. and Nevalainen, O. (1999) Vector quantization by lazy pairwise nearest neighbor method. *Optical Engineering*, 38, 1862-1868.

[23] Linde, Y., Buzo, A. and Gray, R.M. (1980) An algorithm for vector quantizer design. *IEEE Transactions on Communications*, 28, 84-95.

[24] Fränti, P. and Virmajoki, O. (2003) Genetic algorithm using iterative shrinking for solving clustering problems. *Proc. Wessex Data Mining Conf. 2003*, Rio de Janeiro, Brazil, 1-3 December, pp. 193-204.