UNIVERSITY OF JOENSUU
DEPARTMENT OF COMPUTER SCIENCE
Report Series A

Optimal clustering by branch-and-bound technique

Pasi Fränti[1] Olli Virmajoki[1] and Timo Kaukoranta[2]

Report A-2002-2

ACM	I.4.2, I.5.3
UDK	519.712
ISSN	0789-7316
ISBN	952-458-111-6

[1] *Department of Computer Science*
University of Joensuu
Box 111, FIN-80101 Joensuu
FINLAND

[2] *Turku Centre for Computer Science (TUCS)*
Department of Computer Science,
University of Turku
FIN-20520 Turku, FINLAND

Optimal clustering by branch-and-bound technique

Pasi Fränti[1], Olli Virmajoki[1] and Timo Kaukoranta[2]

[1] *Department of Computer Science*
University of Joensuu
P.O. Box 111, FIN-80101 Joensuu
FINLAND

[2] *Turku Centre for Computer Science (TUCS)*
Dept. of Computer Science, University of Turku
Lemminkäisenkatu 14A, FIN-20520 Turku
FINLAND

Abstract: In literature, no much attention has been paid to algorithms for finding the optimal clustering except in some special, in which the problem reduces to polynomial time problem. In this paper, we give solution for the general case. The method generates all possible clusterings by a series of merge steps. The clusterings are organized as a minimum redundancy search tree and the optimal clustering is found by a branch-and-bound technique. The result has theoretical interest and could also provide new insight to the problem itself. From the branch-and-bound technique, we also derive two suboptimal but polynomial time variants.

Keywords: Clustering, algorithms, optimization, PNN, vector quantization.

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 aims at partition a given set of N data vectors into M groups so that similar vectors are grouped together and dissimilar vectors to different groups. The clustering task is usually formalized as a combinatorial optimization problem, in which the goal is to find the partition that minimizes a given cost function.

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

Agglomerative clustering is one 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 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, 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 motivated by the PNN. 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 optimization function. The idea can be implemented as a *branch-and-bound technique* that uses a search tree for finding the optimal clustering.

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, PNN 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 of *M* is located at the level *N*-*M*. All branches of the tree must be generated in order to find the optimal clustering.

The rest of the paper is organized as follows. In Section 2, we give formal description of the clustering problem 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 but polynomial time variants are introduced in Sections 4. Experimental tests are made in Section 5, and conclusions drawn in Section 6.

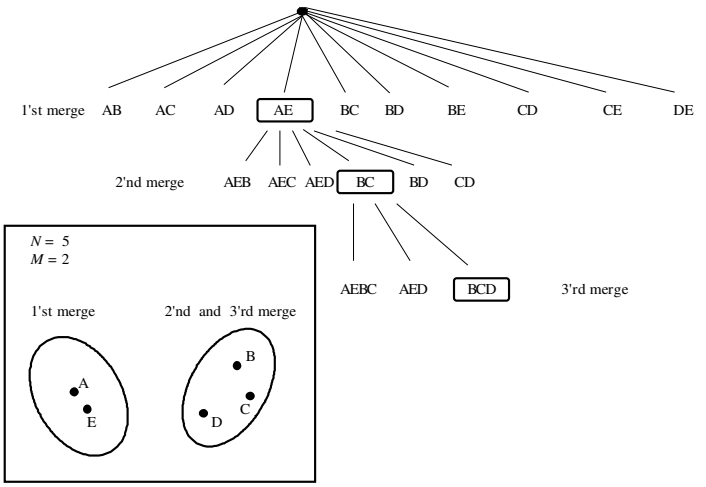

Fig. 1. Illustration of the PNN as a search tree.

2. Pairwise nearest neighbor method

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 similar vectors are grouped together and dissimilar vectors to different groups. Partition $P=\{p_1, p_2, \ldots, p_N\}$ defines the clustering by giving for each data object the cluster index of the group where it is assigned to. *A cluster* s_a is defined as the set of data vectors that belong to the same partition a:

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

The clustering is then represented as the set $S=\{s_1, s_2, ..., s_M\}$. The clusters are sometimes also represented as their centroids: $C=\{c_1, c_2, \ldots, c_M\}$.

The most important choice in the clustering is the cost function f for evaluating the goodness of the 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. In this paper, the task is to find such partition P that minimizes *MSE*.

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]:

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

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

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

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 [8].

```
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. Branch-and-bound technique

We described next a branch-and-bound technique that generates clustering also by a sequence of merge operations. It is easy to see that any clustering can be produced in this way; 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 clustering and 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 S in the tree represent a single clustering with m clusters, and its children represent the clusterings that have been produced by merging any two of m existing clusters. The optimal clustering can then be found by systematic search from the tree.

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 $N\cdot(N\text{-}1)/2$ 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 $(N\text{-}1)\cdot(N\text{-}2)/2$ 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 clustering of M clusters as follows:

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

At the same time we know from [9] that the total number of different clusterings equals to the *Stirling's number of second kind* [10]:

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

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

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

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 (AE) (BCD), and their merge as (AE) + (BCD) = (AEBCD). Using this representation, we can see that the same cluster has several different representations. The clusters (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 define that the only valid representation of a cluster $s_j = \{x_1, x_2, ..., x_{nj}\}$ is the one, in which the data vectors are given in the sorted order according to their index. The only valid representation for the previous example is then (BCD). When using this validity rule, we can still represent all possible clusterings but without redundant representations of a single cluster.

The validity rule can be applied in the *Branch-and-bound* algorithm as follows. In the merge operation, we consider only cluster pairs, in which the data vectors of the first cluster s_a have smaller indices than the data vectors in the second data cluster s_b:

$$\{s_a, s_b\}: i < j \forall x_i \in s_a, x_j \in s_b \tag{9}$$

As a consequence, the order of the data vectors will be automatically retained in the merge operation. For example, 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. Despite of this, the cluster (ABE) is still possible to obtain but only using the sequence that merges first (A) + (B), and then (AB) + (E). Furthermore, if the current clustering is (AE) (B) (C) (D), we cannot merge anymore clusters with (AE) and it will inevitably remain as such in the final clustering. However, it is possible that other clusters will merge together.

3.3 Permuting minimum redundancy search tree

The rule expressed in Equation (9) removes the redundancy in the case of representing a single cluster but it is still possible to construct the same cluster via different *paths* in the search tree. For example, the cluster (BCD) can be constructed using two different paths (merge sequences):

Sequence 1: (B) (C) (D) $\rightarrow$ (BC) (D) $\rightarrow$ (BCD)
Sequence 2: (B) (C) (D) $\rightarrow$ (B) (CD) $\rightarrow$ (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 in order to remove the redundancy but we must also limit the permutation of the search paths in the tree. The redundant of the search paths can be removed using as follows.

We force the algorithm to permutate the cluster pair s_a and s_b in a predefined order so that the index of the first cluster (A) is always monotonically non-decreasing during the process. In other words, if we have merged clusters s_{a_0} and s_{b_0} at the previous level, we are restricted to consider only cluster pairs s_a and s_b such that $a \geq a_0$. From the

Equation (9), we can also derive another restriction $b > a$. These restrictions can be formalized so that only the clusters that meet the following permutation criterion are accepted:

$$(s_a, s_b) : a \geq a_0 \wedge b > a \tag{10}$$

where a_0 is the index of the first cluster in the previous merge. Any individual cluster $s_j=(s_{j1}\ s_{j2}\ s_{j3}\ \ldots\ s_{jn})$ 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 $\{s_1, s_2, \ldots, s_m\}$ can then be generated by constructing the clusters aone 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 breaking contradicting the permutation criterion (10). Thus, the use of the criterion (10) produces non-redundant search tree.

The non-redundant search tree is illustrated in Fig. 3. At the first level, the permutation creates the merges: (AB) (AC) (AD) (AE) (BC) (BD) (BE). We note that the internal order of the clusters is maintained automatically; e.g. the permutation creates cluster (AC) but not (CA). We can also see that after the merge (BC), the merge (AC) do not appear anymore because it has already been created in the branch where (AC) was constructed before (BD). Furthermore, if the previous merges are (AC) ja (BD), the cluster (ACBD) do not appear anymore as it has already been created by the sequence (A)+(B), (AB)+(C), (ABC)+(D).

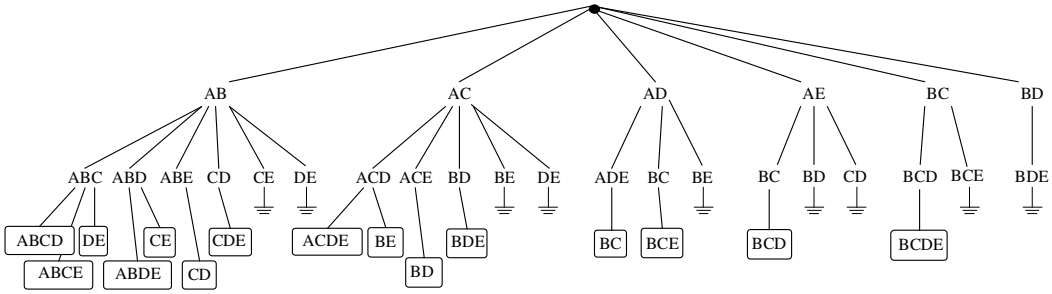

Fig. 3. Example of non-redundant search tree. Branches that do not have any valid clustering have been cut out.

The criterion (10) removes redundant clusterings but there are still exist partial branches that cannot be completed. For example, the merge sequence (AB) $\rightarrow$ (CE) is such as there is no more valid merges left because the permutation criterion does not allow us to add anymore vectors in the cluster (AB), and because the merge (CE)+(D) would break the intracluster order. Fortunately, such branches can be eliminated using rather simple bounds for the permutation loop. The bounds will be derived below.

When permutating 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 $1 \leq a < b \leq m$, where m is the current number of clusters. The index a also indicates how many clusters we already have completed by now. This is because they are not allowed to be included in the merge operation anymore due to (10). The same applies also to the clusters whose index is 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 as:

$$a \leq M \tag{11}$$

Greater values than this would lead to situation that we cannot complete the clustering with M clusters. Pseudo code of the algorithm for generating non-redundant search tree is 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 Early termination of bad solutions

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 limit for the optimal solution. Other branches of the tree can then be terminated if the following condition is true:

$$MSE_t \geq MSE_{\min}, \tag{12}$$

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

It was shown in [11] 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 step. The monotonicity property says that:

$$d_1 \le d_2 \le \ldots \le d_{N-M} \tag{13}$$

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:

$$MSE_t + (N - M - t) \cdot d_t \ge MSE_{\min} \tag{14}$$

Here (N-M-t) indicates the number of forth-coming merges in the algorithm, and d_t is the previous merge cost. The termination 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 us termination of sub-optimal solution much earlier than using (12).

The only problem of using the stronger termination criterion of (14) is that the monotonicity property does not necessarily hold true in the branch-and-bound implementation 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 (14).

The optimal solution can be reached via several different search paths. It is expected that at least one of these paths fulfills the monotonicity 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 monotonicity property. The consequence of this is that we cannot use the stronger termination criterion with the non-redundant search tree. Nevertheless, it is expected that the redundancy of the tree is much greater than the potential improvement of the termination criterion (14).

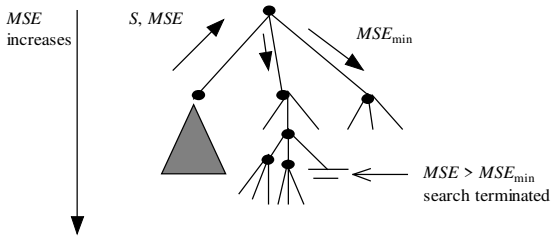

Fig. 5. The use termination criterion in the branch-and-bound technique.

4. Polynomial time variants

The complexity of the branch-and-bound technique is exponential despite the previous improvements. The practical usability of the method is therefore strongly limited to small special cases only. We design next two sub-optimal variants that do not guarantee the optimality of the result but they work in polynomial time.

4.1 Piecewise optimization

The original optimization problem is divided into a series of smaller subproblems that are solved optimally. 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. 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 denoted as *Piecewise optimization*, and it is illustrated in Fig. 6.

If the size of a single sub-problem is Z, we need to repeat the $\lceil (N\text{-}M)/Z \rceil$ optimization stages in the algorithm. The quality of the result depends on the parameter Z; greater values will give better clustering result at the cost of exponentially higher 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 original PNN method. The pseudo code of the *Piecewise optimization* is shown in Fig. 7.

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 results 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 $O(N \cdot N^{2Z})$. 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 as Z=2, or Z=3.

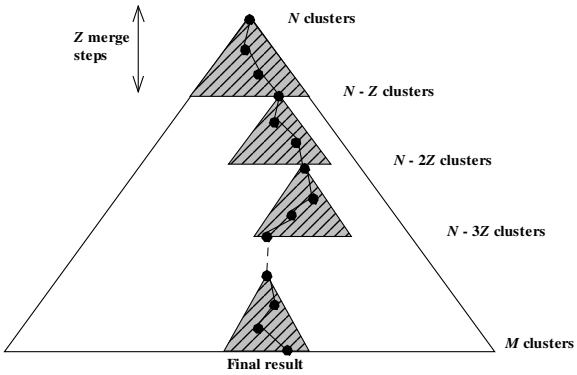

Fig. 6. Illustration of the search using *Piecewise optimization*.

PiecewiseOptimization(X, M, Z) $\rightarrow$ S;
  FOR $i \leftarrow 1$ to N DO
    $s_i \leftarrow \{x_i\}$;
  REPEAT
    $size \leftarrow \max(M, |S|\text{-}Z)$;
    $S \leftarrow \mathrm{BB}(S, 1, size, \infty)$;
  UNTIL $|S| = M$;

Fig. 7. Piecewise optimization algorithm.

4.2 Look-ahead optimization

The *Piecewise optimization* traverse in the search tree through local optima. At each stage of the algorithm, the last merge is the most critical because the algorithm generates 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 to 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. With Z-values larger than 2, we would also do unnecessary work as part of the search tree would be re-generated several times. In practice, the algorithm can be realized only with very small Z-values.

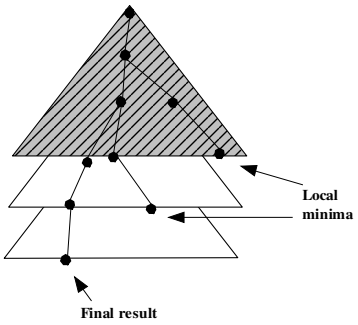

Fig. 8. Illustration of the search using *Look-Ahead optimization*.

5. Experiments

We consider first simulated data sets in order to find out how large clustering problems the optimal branch-and-bound technique can solve. The results are summarized in Fig. 9-11 with different number of clusters (M=2,5,9) using Pentium 450 MHz computer. The algorithm is fastest when the number of clusters is set small (M=2), or large (M=9). These correspond to the situations, in which the number of possible solutions is smallest. The use of early termination improves the algorithm but the improvement is only marginal. We conclude that the practical usability of the optimal branch-and-bound algorithm is limited to very small size problem instances only.

Similar results are given also to the polynomial time variants in Fig. 12. The algorithms are performed for a simulated data set with 15 clusters. The size of the data set is varied by randomly subsampling source data set of 1000 vectors. The polynomial time variants are faster and can process larger data sets than the optimal branch-and-bound technique. Only small sub tree sizes (Z=2,3) were applied.

The qualities of the sub-optimal variants are compared in Table 2 with other heuristic clustering methods. The *random clustering* is obtained by randomly selecting M data

vectors and use them as cluster representatives, and then partition the rest of the data set 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 [11]. The PNN is implemented as in [8], and the RLS refers to the *Randomized Local Search* method [13], which is one of the best heuristic clustering algorithms in terms of quality.

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 RLS, on the other hand, gives somewhat better results than the PNN and BB. It is noted that the optimality of the results is not known but it is expected that the RLS results are close to optimum.

Finally, we compare the performance of different clustering methods for a standard clustering test problem SS2 of [9], 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. The results in Table 3 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 RLS. Moreover, the slowness of the algorithms may restrict their use in the case of larger data sets.

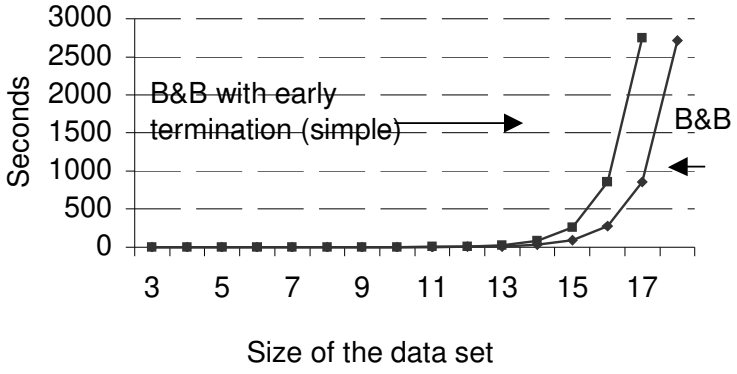

Fig. 9. The effect of the problem size to the time (M=2).

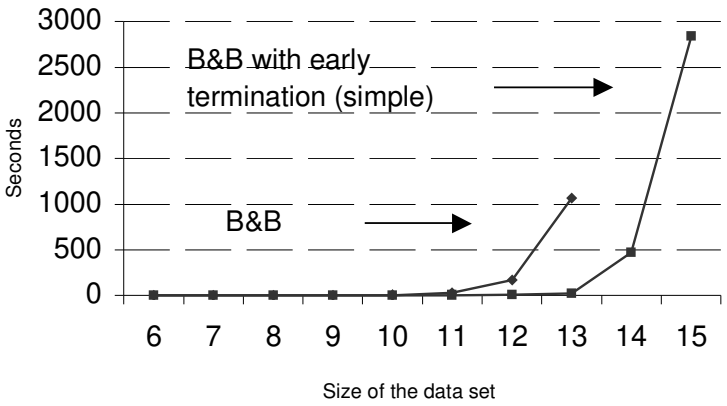

Fig. 10. The effect of the problem size to the time (M=5).

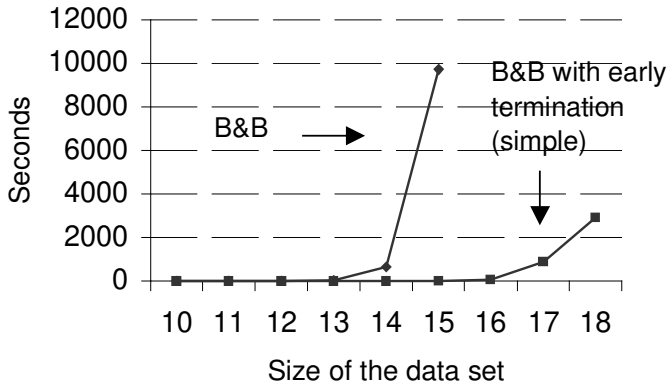

Fig. 11. The effect of the problem size to the time (M=9).

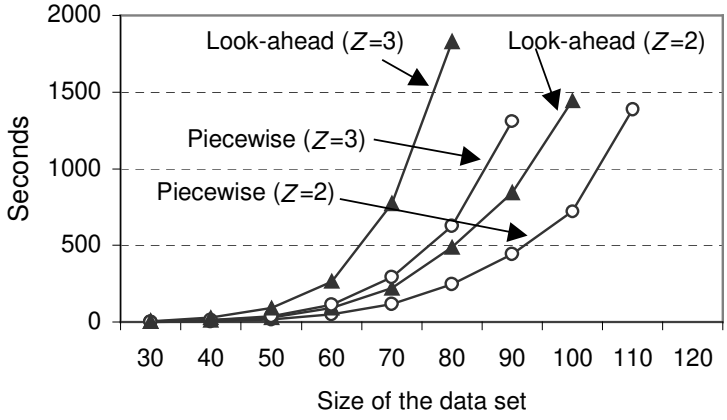

Fig. 12. The effect of the problem size to the run time in the case of the polynomial time variants (M=15).

Table 2: Performance comparison for the simulated data set (N=30..100).
The error 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 [12]	0.948	1.081	1.108	1.541	1.647	1.614	1.628	1.837
PNN [8]	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
RLS [13]	0.396	0.573	0.992	1.141	1.203	1.169	1.238	1.335

Table 3: Performance comparison for the data set SS2 (N=89) from [9].
The error values are mean square errors ($\times 10^9$).

Method:	**Error:**	**Time:**
Random clustering	1.760	≈ 0 s
K-means [12]	1.140	≈ 0 s
PNN [8]	0.336	≈ 0 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
RLS [13]	0.313	≈ 0 s

6. Conclusions

We have introduced a branch-and-bound technique to solve optimal clustering. The time complexity of the algorithm is still exponential despite the non-redundant search tree, and the designed bounding criterion. The practical usability of the algorithm is therefore limited to small special cases only.

We also derived two polynomial time algorithms from the branch-and-bound technique. The algorithms offer good compromise between the optimal algorithm and the PNN. They can operate with larger data sets than the optimal branch-and-bound technique and, on the other hand, give better results than the PNN in terms of quality. However, the methods were still outperformed by the best competitor – the RLS.

References

1. B.S. Everitt, *Cluster Analysis* (3rd edition). Edward Arnold / Halsted Press, London, 1992.
2. L. Kaufman and P.J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley Sons, New York, 1990.
3. R. Dubes and A. Jain, *Algorithms that Cluster Data*. Prentice-Hall, Englewood Cliffs, NJ, 1987.
4. A. Gersho and R.M. Gray, *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, Dordrecht 1992.
5. M.R. Garey, D.S. Johnson, H.S. Witsenhausen, "The complexity of the generalized Lloyd-Max problem". *IEEE Transactions on Information Theory*, 28 (2), 255-256, March 1982.
6. J.H. Ward, "Hierarchical grouping to optimize an objective function", *J. Amer. Statist.Assoc.*, 58, 236-244, 1963.
7. W.H. Equitz, "A new vector quantization clustering algorithm", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37 (10), 1568-1575, October 1989.
8. P. Fränti, T. Kaukoranta, D.-F. Shen and K.-S. Chang, "Fast and memory efficient implementation of the exact PNN", *IEEE Transactions on Image Processing*, 9 (5), 773-777, May 2000.
9. H. Späth, *Cluster Analysis Algorithms for Data Reduction and Classification of Objects*, Ellis Horwood Limited, West Sussex, UK, 1980.
10. R.L. Graham, D.E. Knuth, O. Patashnik, *Concrete Mathematics – a Foundation for Computer Science* (2nd edition), pp. 257-267, Addison-Wesley, 1994.
11. T. Kaukoranta, P. Fränti and O. Nevalainen, "Vector quantization by lazy pairwise nearest neighbor method", *Optical Engineering*, 38 (11), 1862-1868, November 1999.
12. Y. Linde, A. Buzo and R.M. Gray, "An Algorithm for Vector Quantizer Design", *IEEE Transactions on Communications*, 28 (1), 84-95, January 1980.
13. P. Fränti and J. Kivijärvi, "Randomized local search algorithm for the clustering problem", *Pattern Analysis and Applications* 3(4): 358-369, 2000.