Branch-and-bound technique for solving optimal clustering

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

The problem of finding optimal clustering has not been well covered in literature. Solutions can be found only for special cases, which can be solved in polynomial time. 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.

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]. 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* [4]. 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* [5] 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 [6]. 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 this idea.

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 method to the PNN method is demonstrated in Fig. 1. At the first step, the method generates all possible merges of two vectors. 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.

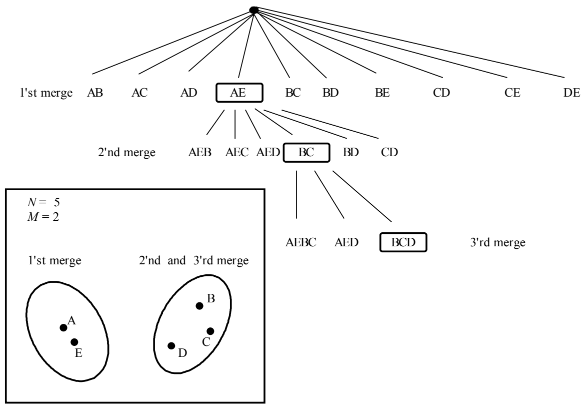

Figure 1. Illustration of the PNN as a search tree.

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. 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 in the same partition a:

$$s_a = \{x_i \mid p_i = a\} \tag{1}$$

The most important choice in the clustering is the cost function f for evaluating 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 centroids $C=\{c_i\}$, 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 combinatorial optimization problem.

The *pairwise nearest neighbor* (*PNN*) method [5, 6] generates the clustering hierarchically using a sequence of merge operations. At each step of the algorithm, the number of clusters is reduced by merging two 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 [6]:

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

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.

3. Branch-and-bound technique

We described next a branch-and-bound technique that generates 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. It 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 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 represent a single clustering with m 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 by different merge sequences. At the first step, there are $N\cdot(N-1)/2$ alternatives for the merge operation. Similarly, at the second level there are $(N-1)\cdot(N-2)/2$ alternatives for the merge operation independent of the merge operation made at the previous step.

In general, every node has $(m)\cdot(m-1)/2$ children at the level with m clusters. We can therefore derive the total number of merge sequences as:

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

At the same time we know from [1] that the total number of different clusterings equals to the *Stirling's number of second kind* [7]:

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

By closer examination, one can see that the number of sequences is significantly higher than the number of different clusterings. Thus, 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 (AE) (BCD), and their merge as (AEBCD). Using this representation, the same cluster has several different representations. For example, the cluster (BCD) has the following representations: (BCD), (BDC), (CBD), (CDB), (DBC) and (DCB).

The data vectors x_i can be ordered by their index i. We define that the only valid representation of a cluster $s_j = \{x_1, x_2, \ldots, x_{nj}\}$ is the one, in which the data vectors are sorted according to their index. The only valid representation for the previous example is then (BCD).

The above 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:

$$(s_a, s_b) : i < j \forall x_i \in s_a, x_j \in s_b \tag{6}$$

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.

3.3 Permuting minimum redundancy search tree

The rule expressed in (6) removes the redundancy in the case of representing 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 the intra cluster representation but also the permutation of the search paths. Redundant search paths can be removed 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 is always monotonically non-decreasing during the process. In other words, if we have merged clusters s_{a0} and s_{b0} at the previous level, we consider only cluster pairs s_a and s_b such that $a \geq a_0$. From (6), we can also derive another restriction $b > a$. Thus, only clusters that meet the following permutation criterion are accepted:

$$\left(s_a, s_b\right): a \geq a_0 \wedge b > a \tag{7}$$

Any cluster $s_j=(s_{j1}\ s_{j2}\ s_{j3}\ ...\ s_{jn})$ can be 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 breaking the permutation criterion (7). Thus, the use of the criterion (7) produces non-redundant search tree.

The non-redundant search tree is illustrated in Fig. 2. 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) and (BD), the cluster (ACBD) do not appear anymore as it has already been created by the sequence (A)+(B), (AB)+(C), (ABC)+(D).

The criterion (7) removes redundant clusterings but there are also partial branches that cannot be completed. For example, the merge sequence (AB) $\rightarrow$ (CE) is such as there are 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 as follows.

When permutating new cluster pairs for merge, we always start to permute from the pervious cluster s_{a0}, and consider all potential pairs (s_a, s_b) 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. This is because they are not allowed to be included in the merge operation anymore due to (7). The same rule applies also to the clusters whose index is between a and b. As we have m-M more merges to be performed, there must remain equally many valid cluster pairs. Concluding from this, we can derive the upper bound for the index a:

$$a \leq M \tag{8}$$

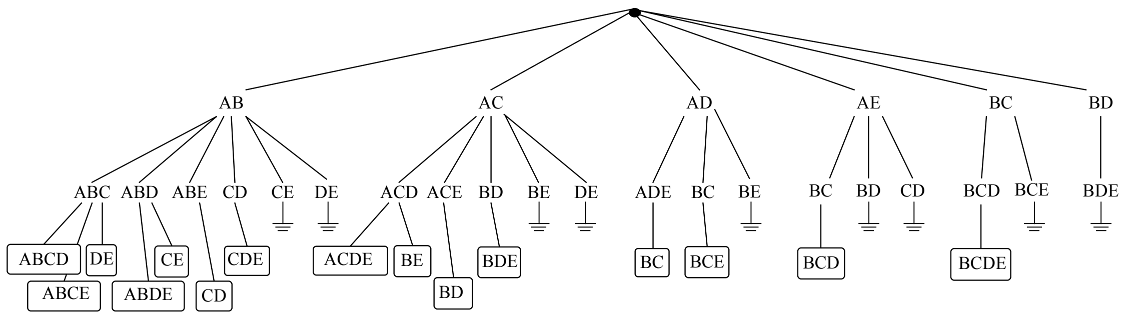

Fig. 2. Example of non-redundant search tree. Branches that do not have any valid clustering have been cut out.

```
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. 3. Algorithm for generating non-redundant search tree.

Greater values than that would lead to situation in which we cannot complete the clustering with *M* clusters. Pseudo code for the algorithm is shown in Fig. 3.

3.4 Early termination of bad solutions

The search can be terminated earlier if 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 increases the MSE-value. Thus, when we have generated the first solution in the search tree, we can use its MSE-value as the upper limit. Other branches of the tree can then be terminated if:

$$MSE_t \geq MSE_{\min} \tag{9}$$

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.

It was shown in [8] 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 $d_1, d_2, ..., d_{N-M}$, where d_t is the merge cost at the t'th merge step. The monotonicity property says that:

$$d_1 \leq d_2 \leq \ldots \leq d_{N-M} \tag{10}$$

The criterion applies 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:

$$MSE_t + (N - M - t) \cdot d_t \geq MSE_{\min} \tag{11}$$

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.

The only problem of using the stronger termination criterion of (11) is that the monotonicity property does not necessarily hold true in the implementation of Fig. 3. 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 (11). The consequence of this is that we cannot use the stronger termination criterion with the non-redundant search tree.

4. 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.

The main idea, however, can also be used to design sub-optimal but polynomial time algorithm as follows. The original problem is divided into a series of smaller subproblems that are solved optimally. The larger the size of the subproblems, the better (and slower) the algorithm is expected to be. This idea is a point of future research.

References

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

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

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

[4] M.R. Garey, D.S. Johnson, H.S. Witsenhausen, "The complexity of the generalized Lloyd-Max problem". *IEEE Trans. on Inf. Theory*, **28** (2), 255-256, March 1982.

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

[6] W.H. Equitz, "A new vector quantization clustering algorithm", *IEEE Transactions on Acoustics, Speech, and Signal Processing* **37** (10), 1568-1575, October 1989.

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

[8] T. Kaukoranta, P. Fränti and O. Nevalainen, "Vector quantization by lazy pairwise nearest neighbor method", *Optical Engineering*, **38** (11), 1862-1868, November 1999.