1 Introduction

Clustering is an important combinatorical optimization 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 [JD88, KR90, E92, GG92]. Clustering is also needed for designing a codebook in vector quantization. The clustering problem contains two subproblems: *determining the number* of clusters and *finding the location* of clusters.

The process of solving a pattern recognition problem (see Figure 1.1) typically involves the following steps [JD88, JMF99]:

(1) Pattern representation (including feature extraction and/or selection).

(2) Definition of a pattern proximity measure appropriate to the data [A73, DS76, MS83, JD88].

(3) Clustering or grouping.

(4) Data abstraction (if needed) [DS76].

(5) Assessment of output (if needed) [D87, D93, C95b, K01].

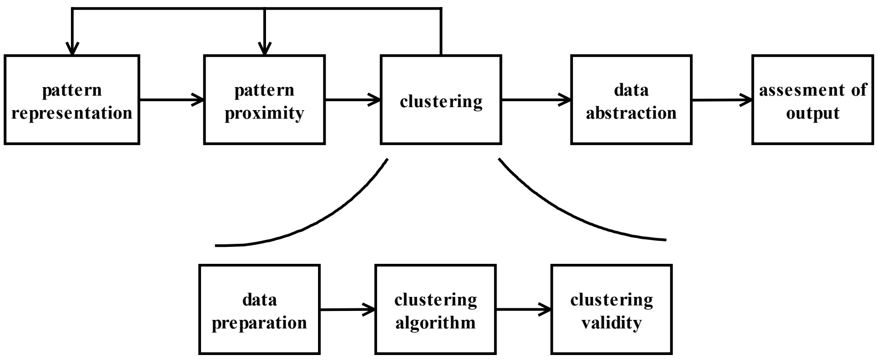


Figure 1.1. The steps of cluster analysis for pattern recognition.

The preprocessing of data objects improves the efficiency and the outcome of the clustering algorithm. With appropriate preprocessing, larger data sets can be clustered and, in addition, better clustering results can be achieved during the actual clustering. Although most of the preprocessing methods are designed to aid in the manipulation of the clusters, unsupervised learning may also focus on *description* tasks [T99]. The preprocessing of data (data preparation) can be considered as a separate task that takes place before actual clustering. It is common that a data set includes unnecessary features (noise and outliers) and asymmetrically scaled features [MS03]. If a data set is not properly preprocessed, it can be assumed that the outcome of clustering is suboptimal. Some preprocessing methods are:

(1) Indexing of the data [B75, FBF77, B80, B90, S91, Y93, NN97, B02, BM02] for nearest neighbor search.

(2) Projecting the data [FBS75, GK92, RK93, LC94, LC95a, BJS97, **P1**] for a nearest neighbor search.

(3) Using an *approximating and eliminating search algorithm* (*AESA*) [V86, MOV94, V94, V95, RP00].

(4) Scaling and weighting of the data [MS03].

(5) Selecting the features [GM86, F87, ZRL97, T99, T00, DCSL02].

(6) Using heuristic clustering to get an initial partition [S80, JD88].

(7) Filtering noise and outliers [BCQY97, RRS00, WC04].

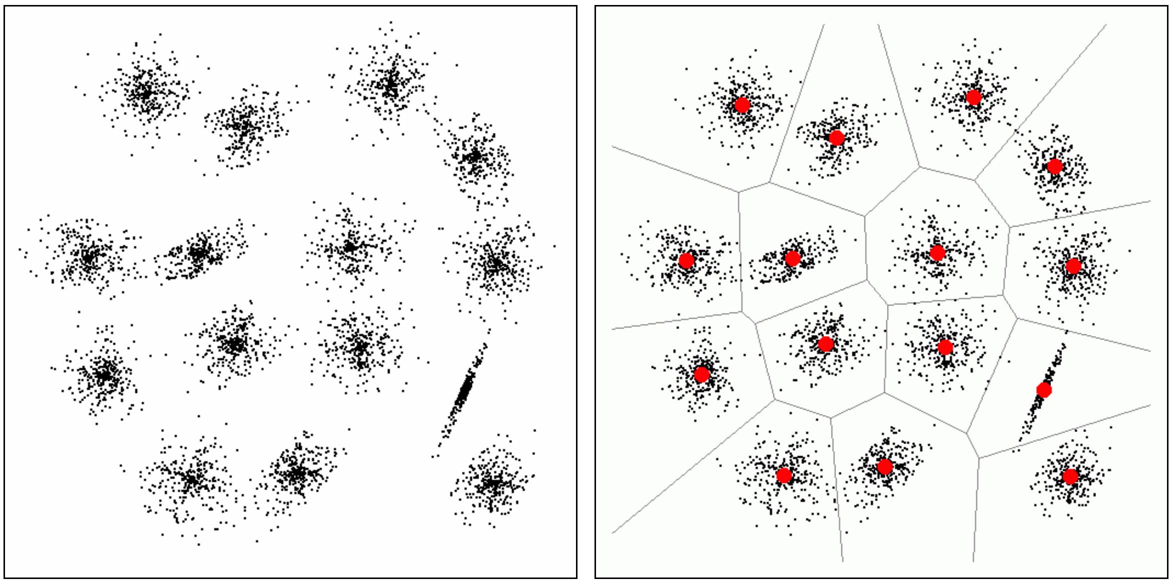


Figure 1.2. An example of clustering. On the left, a sample data set is shown. On the right, a clustering of the data set with 15 clusters and their centroids is shown.

Clustering entails partitioning a data set so that similar objects are grouped together and dissimilar objects are placed in separate groups; see Figure 1.2 for an example. The dots represent two-dimensional data points in Euclidean space. The 5000 data objects have been divided into 15 clusters separated by straight lines in the figure. The data objects separated by straight lines form a *partition* of the data. Centroids of the partition are represented by large dots.

Formally, the clustering problem is defined as follows: Given a set of N data vectors $X=\{x_1, x_2, \ldots, x_N\}$ in a K-dimensional space, 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 is assigned to. Cluster s_a is defined as the set of data vectors that belong to the same partition a:

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

Clustering is then represented as the set of clusters $S = \{s_1, s_2, ..., s_M\}$. In vector quantization, the output of clustering is a codebook $C = \{c_1, c_2, \ldots, c_M\}$, which is usually the set of cluster centroids (code vectors).

The most important choice in clustering is the cost function f for evaluating the goodness of the clustering. For a given application, the criterion can be based on the principle of *minimum description length* (*MDL*) [R78, S78]. Otherwise, the criterion is based on certain assumptions of data normalization and spherical clusters. Basically, the function should correlate with high inter cluster distance and low intra cluster diversity. In the case of binary data, *stochastic complexity* has been applied [GKV97].

When data vectors belong to Euclidean space, a commonly used function is the *mean square error* (*MSE*) between the data vector and their cluster centroids. Given a partition P and the codebook C, the MSE is calculated as:

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

where p_i is the cluster (partition) index of the data vector x_i. From here on, the vectors are assumed to belong to Euclidean space and the mean square error (MSE) is used as the objective function. The number of clusters is also assumed to be known beforehand, unless otherwise stated.

2 Clustering algorithms

Clustering is an *NP-complete* [GJW82] combinatorical optimization problem, for which optimal solutions can be found by the *branch-and-bound* technique, but in exponential time [FN75, KNF75, FG88, W91, WL93, C95a, P97, GYZ98, GRS99, IA99, FV02, FVK02, MP02, **P5**]. The total number of different clusterings equals *Stirling's number of the second kind* [GKP94]. Suboptimal algorithms must therefore be used in practice.

Existing methods can be roughly divided into *heuristic*, *optimization, graph-theoretical* and *hierarchical* methods. The heuristic methods are usually fast but rough, and hence, they are not adequate alone. Instead, they can be used for creating an initial solution for more sophisticated algorithms. Optimization methods produce a partition that aims at minimizing an *a priori* chosen objective function, which is a goodness (of fit?) criterion for a partition [B99]. Hierarchical clustering methods start with an initial partition. After that, clusters are split or merged repeatedly. In hierarchical methods, an attempt is made to perform locally optimal steps, but this does not necessarily lead to the global optimal solution relative to any criterion function.

2.1 Algorithms

There are many heuristic clustering algorithms proposed in the literature [H75, S80, JD88]. They belong to the group of *partitional methods* that produce only one partition. An explicit objective function is not necessarily used since the object is to allocate the data vectors to the cluster they seem to fit best at the moment. Usually, each data vector is considered only once in a greedy manner, and it is allocated to a cluster or it is not allocated at all. These algorithms tend to produce rather weak solutions. Three simple heuristic methods are *leader* [H75, S80], *nearest neighbor clustering* [JD88] and *joiner* [H75, S80].

Optimization methods [K04a] are usually based on *k-means* [F65], which is also referred to by several other names, such as *c-means* [M67], *generalized Lloyd algorithm* (*GLA*) [L57], *Linde-Buzo-Gray* algorithm (*LBG*) [LBG80] or *iterative self-organizing data analysis technique* (*ISODATA*) [BH65]. These are slightly different variations on each other but the main concept is nevertheless the same. *K-means* applies two optimization steps iteratively: (i) it calculates an optimal partition for the given codebook, and (ii) calculates a new codebook as the cluster centroids. These steps are based on two optimality conditions:

(1) *Nearest neighbor condition*: for a given codebook, the optimal clustering of the data set is obtained by mapping each data vector to its nearest code vector in the codebook (or cluster representative) with respect to the evaluation function.

(2) *Centroid condition*: for a given partition, the optimal code vector (cluster representative) is the centroid (average vector) of the data vectors within the partition.

Each iteration of the *k-means* algorithm decreases the distortion; however, the process leads only to a *local optimum* that depends on the initial codebook, with *M* random data vectors, that was used. Several other techniques have also been proposed for generating an initial codebook [LBG80, YG88, GG92, NT92, BM93, KKZ94, AR96, PLL99]. Thus, *k-means* can fine-tune a codebook. It can also be integrated in many other of the more complicated algorithms discussed later in this thesis.

There are many variants of the *k-means* algorithm that include fast implementations [HLC91, WG94, CKS95, M95, CL96, KFN96, F97, BJLS98, KFN00, Z01, KMNPSW02, LVV03]. The fuzzy variant is known as *fuzzy c-means* [D73, DL94, CCLH97, OZ00, WCS01]. Among many other variants, there are *simulated annealing* [KGV83, ZG89, HPLSH01] and *deterministic annealing* [RGF90, HB97] variants. These methods are based on interesting theories but it does not mean that they would work well in practice.(this sentence, still, seems like it is an abrupt change of tone to the rest of the paper. The author takes a subjective position about the interest of these theories) Among those, simulated annealing is probably the one that works best in practice.

The best clustering results, in term of minimizing the distortion function, have been obtained by the *genetic algorithm* (*GA*) [H75, G89, MC96], which uses agglomerative methods in the crossover. The method was first reported by [FKKN97] using an effective but rather slow algorithm. The algorithm was then simplified and made faster [F00]. It is noted that the use of random crossover is not sufficient to be superior to the other clustering methods.

An alternative approach has been offered by the *randomized local search* algorithm (*RLS*) [FK00, FXK03]. In terms of minimizing the distortion function, it is almost as good as genetic algorithms but the *RLS* method is much simpler to implement. It uses a simple trial-and-error approach in which new candidate solutions are generated by random swapping of the code vectors and then uses *k*-means for fine-tuning the candidate solutions. The *RLS* algorithm has been successfully applied to the classification of bacteria [FGGKKLN00].

The graph-theoretical methods can be divided into three categories:

(1) Methods where the initial partition is the singleton partition and a hierarchical sequence of partitions obtained by agglomeration [KHK99,

HK01, **P2**].

(2) Methods that produce a connected graph from which a clustering is obtained by cutting inconsistent edges [Z71, J78, T80a, T80b, U82, U83, KN86, KNT87, SC91a, SC91b].

(3) Methods that produce an unconnected graph where the connected components correspond to the clusters [U82].

Hierarchical methods [K99] are either *divisive* or *agglomerative*. Divisive methods start by putting all data vectors in a single cluster. New clusters are created by dividing existing ones. This approach involves two main design problems: which cluster to divide, and how the division is performed. The division can be made along a selected dimension of the vector space as in the *median cut* algorithm [H82], or along the principal axis [WZ91]. The latter approach has been studied and a method has been proposed for locally fine-tuning the cluster boundaries after the divisions have been made [FKN97]. The divisive methods can be fast, e.g., $O(N \cdot \log N)$, but more complicated than the agglomerative methods.

Agglomerative clustering is simple to implement and it provides better clustering results than the divisive approach. It can also be combined with *k-means* as proposed [GPF95], or used as the merge phase in the split-and-merge algorithm [KFN98], which results in a good time-distortion performance.

Another clustering method is the *self-organizing map* (*SOM*) [K88, K95], which is commonly applied to data mining and to the visualization of complex data sets. SOM can also be used for clustering [NF88, CTC94]. Unfortunately, for large clustering problems *SOM* is inferior to the majority of other, noteworthy methods (noteworthy is redundant, you could leave it out for clarity's sake) [K04a]. Furthermore, the clustering results depend strongly on the parameter setup [F99].

2.2 Unknown number of clusters

Determination of the number of clusters in data requires that one has both an algorithm that searches for the correct clustering of data and has a criterion that is capable of recognizing the number of clusters [DB79, I80, FK97, SYK97, BP98, KF02b, KLL04, SWJ04, WC04, X04]. The simplest approach is to loop the number of clusters, use an existing algorithm for a fixed number of clusters in the loop, and then select the best solution using some criterion. This *brute force* search is guaranteed to work (assuming that the criterion is valid) but it is also slow.

The *stepwise clustering* algorithm, which reduces the workload required by the brute force approach, has been proposed [KF02b]. The idea is to utilize the previous solution as a starting point when solving the next clustering problem that has a different number of clusters. A stopping criterion is applied to estimate the potential improvement of the algorithm and to stop the iteration when the estimated improvement stays below a predefined threshold value.

There are also methods that solve the number and location of the clusters jointly. The c*ompetitive agglomeration* algorithm (*CA*) [FK97] decreases the number of clusters until there are no clusters smaller than a predefined threshold value. The drawback is that the threshold value must experimentally be determined. Another approach is the *dynamic local search* [KF02a] that solves the number and location of the clusters jointly. The algorithm uses a set of basic operations, such as cluster addition, removal and swapping.

2.3 Fast search methods

In the clustering problem, one typically needs to search for the *nearest neighbor* (*NN*) during the process. The nearest neighbor search problem involves searching for a set of K-dimensional vectors $C = \{c_i\}$ that are nearest to a given target vector x with respect to a distance function d. A large part of the running time of the search is due to the computation of the O(K) distance function. A *full search* solution involves calculating the distance between the data vector, x, and every code vector, c_i, in order to find the nearest to x; however, the full search solution comes at the cost of O(NK). (a ';' should separate two complete clauses, not a clause and a phrase.

In the search of nearest code vector in Euclidean space, several speed-up methods have been introduced that reduce the computation required by distance calculations [FN75, S75, BS76, FBF77, WL83, BG85, V89, S90a, S90b, CH91, MNS91, O91, OB91, GK92, HBSH92, RP92, RK93, EE94, LC94, CK95, LC95a, LC95b, LC95c, LS95, DE96, BJS97, RP97, AMNSW98, GG98, KS98, SC98, BBS99, KFN00, RP00, S00, TSL00, WL00, Y00, BBK01, CNBM01, HC01, AP02, BM02, KMNPSW02, M02, BN03, HS03a, HS03b, M03, SR03, K04b].

The *triangular inequality elimination* (*TIE*) technique presented by Chen and Hsieh [CH91] maintains the distances between all code vectors and then reduces the number of distance calculations by a condition derived from the triangle inequality.

The *partial distortion search* (*PDS*) proposed by Bei and Gray [BG85] terminates a single distance calculation immediately when the partial distance exceeds the shortest distance previously found. Let s_a be the cluster for which one seeks the nearest neighbor. One uses full search to calculate the distance values $d_{a,j}$ between s_a and all other clusters s_j. Let $d_{\min}$ be the distance of the best candidate found so far. The distance is calculated cumulatively by summing up the squared differences in each dimension. In *PDS*, one utilizes the fact that the cumulative summation is nondecreasing, since the individual terms are nonnegative. The calculation is therefore terminated and the candidate is rejected if the partial distance value exceeds the current minimum $d_{\min}$.

The *mean-distance-ordered partial search* (*MPS*) technique introduced by Ra and Kim [RK93] applies two different techniques to speed-up the search of the nearest code vector. First, it uses a fast precondition for checking whether the distance calculation to a given candidate cluster can be omitted. Second, it sorts the codebook according to the component means of the code vectors and derives limits for the search.

An interesting approach for clustering is to use *graph theoretical methods* [GR69, U82, JD88, OM95, EPY97, BCQY97, KHK99, HS00, HK01, B04]. For example, by first creating a complete undirected graph where the nodes correspond to the data vectors and the edges correspond to vector distances according to a given *similarity* or *dissimilarity* measure. The resulting graph can be trimmed to a *minimum spanning tree*, which can be interpreted as one large cluster. The clustering can then be generated by iteratively dividing the cluster by removing the longest edges from the graph. In the final graph, clusters can be determined by finding the separate components of the graph [GR69]. This algorithm can be seen as a variation of split-based methods with a similar criterion to the criterion in *single-linkage* agglomerative clustering.

Many agglomerative clustering algorithms construct a sparse graph and then perform the clustering on this graph [JD88, KHK99, HK01]. Two main characteristics of these approaches are that

(1) the methods construct undirected graphs, and

(2) the methods neglect the original data after building the weighted graph (meaning that weights of the new edges are determined by the weights of current edges).

Generating and utilizing the graph efficiently is problematic for this approach. For example, standard solutions for solving minimum spanning tree takes O(N^2) time, which would prevent any speed-up.

A *k-nearest neighbor graph* (*kNN graph*) is defined as a weighted, directed graph in which every node represents a single cluster and the edges correspond to pointers to neighbor clusters. A pointer maintains the index of the neighbor cluster and the corresponding distance. Every node has exactly k edges to the k nearest clusters according to a given distance function. The distance of clusters is defined by the merge cost function of the agglomerative clustering, see Eq. (3) below. Note that this is not the only possible definition of the graph: Other definitions have been given in [AM93, CBC00].

2.4 Multilevel thresholding

Multilevel thresholding, which is needed in the compression of the medical images [KOKKNN98], can be seen as a 1-dimensional special case of the clustering problem. The time complexity in the 1-dimensional case of clustering is expected to be lower than in the general case. For example, the time complexity of the *GLA* (for vector quantization in the multidimensional space) is O(NM) while the time complexity of its 1-dimensional counter-part, the *Lloyd-Max quantizer* (*LMQ*) [LS55, L57], is only O(N).

Over the years, many thresholding techniques have been proposed [KR79, O79, P80, WH84, KSW85, T85, B86, KI86, SSWC88, A89, PP89, LSP90, B92, CW92, TC92, PP93, AST94, CWY94, BM95, YCC95, YC97, CSS98, GLC98, Y99, ST00, LCC01, SS01, Y02]. The thresholding methods can be categorized in six groups according to the information they explore. These categories are [SS04]

(1) histogram shape-based methods where the peaks, valleys and curvatures of the smoothed histogram are analyzed;

(2) clustering-based methods where the gray level samples are clustered in two parts as background and foreground or, alternately, are modeled as two Gaussian distributions;

(3) entropy-based methods that use the cross-entropy between the original and binarized image;

(4) object attribute-based methods that search for a measure of similarity between the gray-level and binarized images, such as fuzzy similarity, shape, edges, or the number of objects;

(5) spatial methods that use the probability mass function models, which take into account the correlation between pixels on a global scale; and

(6) local methods that do not determine a single value of threshold, but adapt the threshold value depending upon the local image characteristics instead. (see note on last page).

When extended to multilevel thresholding, many of these methods have high computational complexity since they carry out an exhaustive search, which takes O(N^{M-1}) time. A faster algorithm based on Otsu's method was proposed by Liao et al. [LCC01]. However, it still requires O(N^{M-1}) time. Faster O($N \cdot \mathrm{sqrt}(M \cdot \log N) + N \cdot \log N$) and O($N^2 M$) time algorithms have been developed for multilevel thresholding by Aggarwal et al. [AST94] and by Kundu [K98] respectively.

The best, known method uses *dynamic programming* [B57] and has the time complexity O(NM) for globally optimal scalar quantizers [W91, WZ93]. This method is based on monotonicity properties of optimal scalar quantizers. The same technique can also be adopted to multilevel thresholding because the problem statement is equal. Thus, optimal multilevel thresholding can be calculated ultimately in O(NM) time.

3 Agglomerative clustering

Agglomerative clustering [LW67, CO72, BS93] generates clusters by a sequence of merge operations. 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 has been obtained. The *single linkage* (*SL*) [SS73] method determines the cluster pair to be merged based on the two closest vectors. The *complete linkage* (*CL*) [SS73] method determines the cluster pair to be merged based on the two furthest vectors. *Ward's* method [W63] selects the cluster pair to be merged so that the merge increases the given objective function value least. In the vector quantization context, this is also known as the *pairwise nearest neighbor* (*PNN*) method, attributed to Equitz [E89].

The algorithm is straightforward to implement in its basic form and, in comparison to *k*-means, it gives better results (i.e., it has a codebook with a lower MSE value). The *PNN* method also has the advantage that the hierarchical approach produces multiple codebooks of different sizes as a side-product. Thus, the *PNN* method can be applied to joint minimization of distortion and entropy of code vector indices [FGP90, GPF91, GPF95, KS98]. The algorithm can also be used as a part of hybrid method such as a *genetic algorithm* [FKKN97, F00, KFN03, FV03], or an iterative *split-and-merge* method [KFN98].

3.1 PNN method

The basic structure of the *exact PNN* method is shown in Figure 3.1. The method starts by initializing each data vector x_i as its own code vector c_i. In each step of the algorithm, the size of the codebook is reduced by merging two nearby clusters. The cost of merging two clusters s_a and s_b (*merge cost*), which is also the *distance* between these clusters, is defined as the increase in the distortion of the codebook if the clusters are merged [W63, E89]:

$$d_{a,b} = \frac{n_a n_b}{n_a + n_b} \cdot \left\| c_a - c_b \right\|^2, \tag{3}$$

where n_a and n_b denote the sizes of the corresponding clusters s_a and s_b. This minimizes the increase of MSE caused by the merge operation. The cost function is symmetric ($d_{a,b} = d_{b,a}$) and can be calculated in $O(K)$ time, assuming that n_a, n_b, c_a, and c_b are known.

The costs for merging three clusters are illustrated in Figure 3.2. The cost values are: $d_{a,b} = 32{,}40$; $d_{b,c} = 56{,}25$ and $d_{a,c} = 90{,}75$. It can be observed that, in this case, it is better to merge the central cluster (b) with the smaller cluster (a) than with the larger cluster (c) even though the latter is closer in respect of the Euclidean distance.

```
PNN(X, M) → C, P
    $s_i \leftarrow \{x_i\} \ \forall\, i \in [1,N]$;
    $m \leftarrow N$;
    REPEAT
        $(s_a, s_b) \leftarrow$ NearestClusters();
        MergeClusters($s_a, s_b$);
        $m \leftarrow m-1$;
        UpdateDataStructures();
    UNTIL $m=M$;
```

Figure 3.1. Structure of the exact *PNN* method.

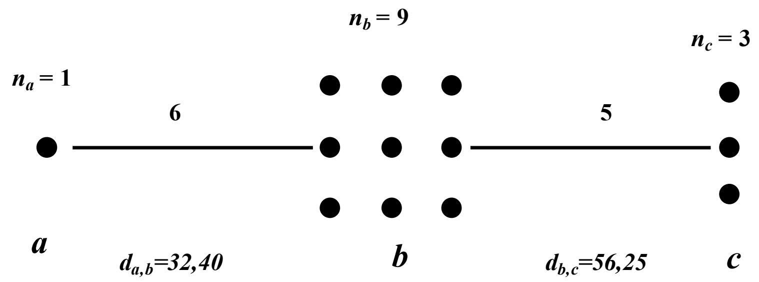


Figure 3.2. Illustration of the distances between the clusters s_a, s_b and s_c, where the Euclidean distance between the clusters s_a and s_b is $\|a-b\|^2 = 6$ and similarly between the clusters s_b and s_c the distance is $\|b-c\|^2 = 5$.

The exact variant of the *PNN* method applies a local optimization strategy: all possible cluster pairs are considered and the one (s_a, s_b) increasing the distortion least is chosen:

$$a,b = \arg \min_{\substack{i,j\in[1,N] \\ i\neq j}} d_{i,j} . \tag{4}$$

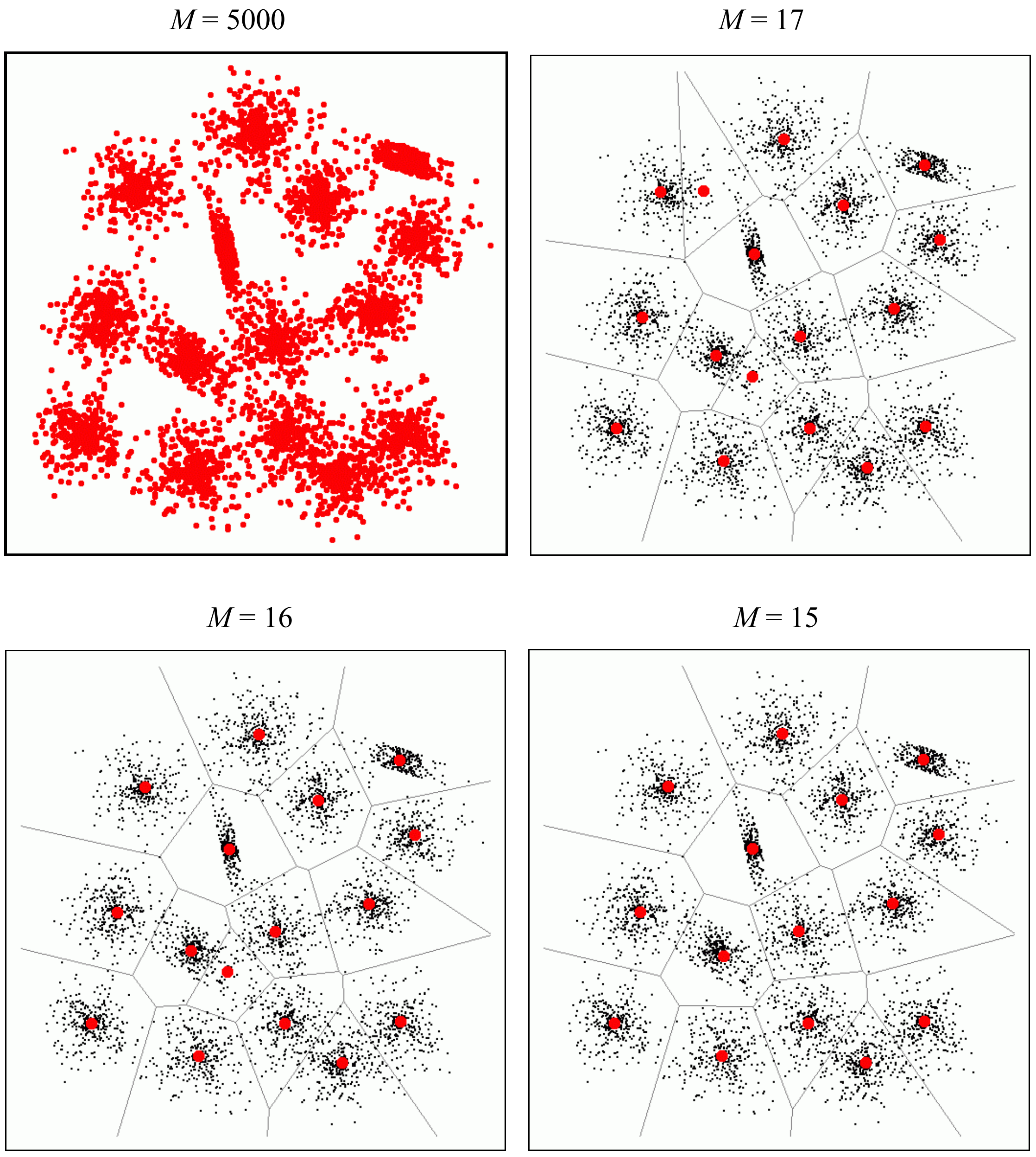
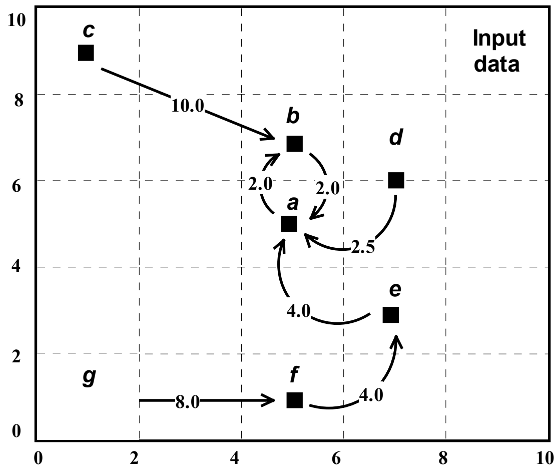


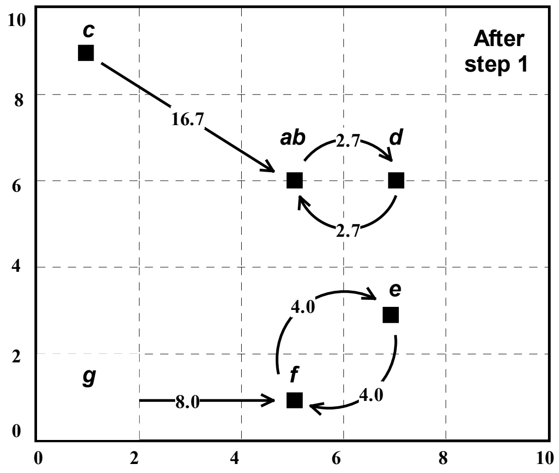
Figure 3.3. Illustration of the agglomeration of the *PNN* method for the data set S_2. The final clustering has 15 separate clusters.

The code vector (c_a) of the combined cluster (s_a) is calculated as the weighted average of the code vectors (c_a and c_b) of the merged clusters s_a and s_b:

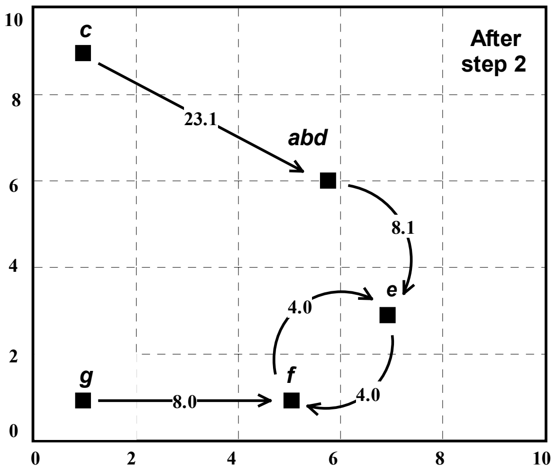
$$c_a \leftarrow \frac{n_a c_a + n_b c_b}{n_a + n_b}. \tag{5}$$



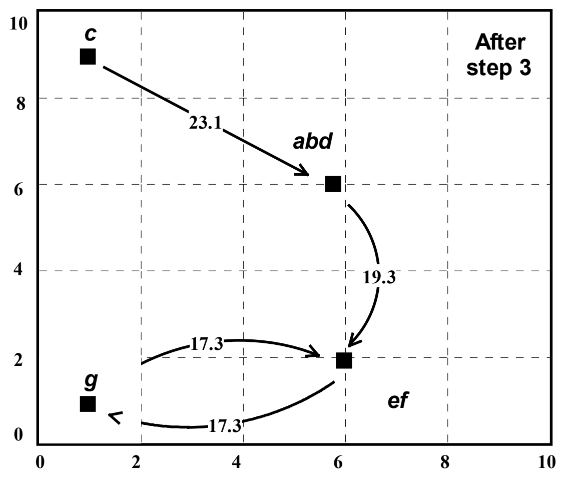
cluster	*a*	*b*	*c*	*d*	*e*	*f*	*g*	min	*NN*
a	--	2.0	16.0	2.5	4.0	8.0	16.0	2.0	*b*
b	2.0	--	10.0	2.5	10.0	18.0	26.0	2.0	*a*
c	16.0	10.0	--	22.5	36.0	40.0	32.0	10.0	*b*
d	2.5	2.5	22.5	--	4.5	14.5	30.5	2.5	*a*
e	4.0	10.0	36.0	4.5	--	4.0	20.0	4.0	*a*
f	8.0	18.0	40.0	14.5	4.0	--	8.0	4.0	*e*
g	16.0	26.0	32.0	30.5	20.0	8.0	--	8.0	*f*



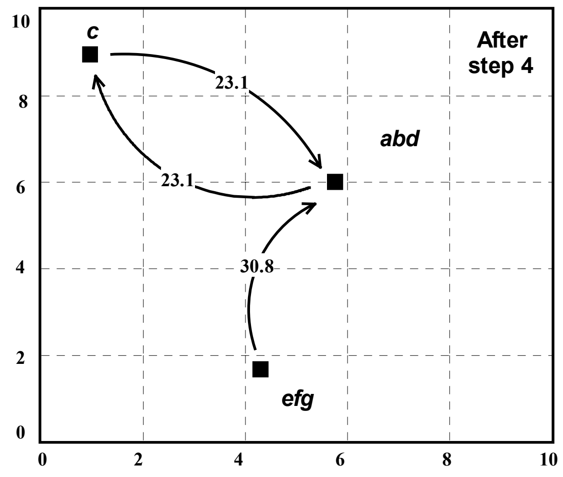
cluster	*ab*	*c*	*d*	*e*	*f*	*G*	min	*NN*
ab	--	16.7	2.7	8.7	16.7	27.3	2.7	*d*
c	16.7	--	22.5	36.0	40.0	32.0	16.7	*ab*
d	2.7	22.5	--	4.5	14.5	30.5	2.7	*ab*
e	8.7	36.0	4.5	--	4.0	20.0	4.0	*f*
f	16.7	40.0	14.5	4.0	--	8.0	4.0	*e*
g	27.3	32.0	30.5	20.0	8.0	--	8.0	*f*



cluster	*abd*	*c*	*e*	*f*	*g*	min	*NN*
abd	--	23.1	8.1	19.1	35.1	8.1	*e*
c	23.1	--	36.0	40.0	32.0	23.1	*abd*
e	8.1	36.0	--	4.0	20.0	4.0	*f*
f	19.1	40.0	4.0	--	8.0	4.0	*e*
g	35.1	32.0	20.0	8.0	--	8.0	*f*



cluster	*abd*	*c*	*ef*	*g*	min	*NN*
abd	--	23.1	19.3	35.1	19.3	*ef*
c	23.1	--	49.3	32.0	23.1	*abd*
ef	19.3	49.3	--	17.3	17.3	*g*
g	35.1	32.0	17.3	--	17.3	*ef*



cluster	*abd*	*c*	*efg*	min	*NN*
abd	--	23.1	30.8	23.1	*c*
c	23.1	--	48.7	23.1	*abd*
efg	30.8	48.7	--	30.8	*abd*

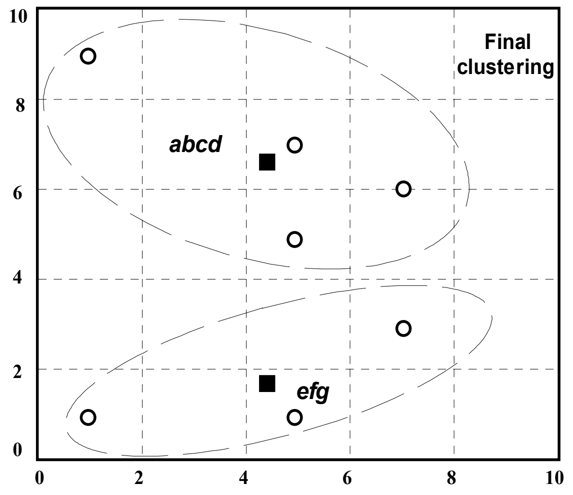


Figure 3.4. *PNN* example. At the end of the agglomeration there are two clusters left.

The clusters are then merged and the process is repeated until the codebook reaches the desired size, M. It should also be noted that, although a single merge operation is always performed optimally, the whole process does not guarantee an optimal codebook. An illustration of the agglomeration of the *PNN* method for a sample data set is shown in Figure 3.3. On the top left, the initial phase after each data vector has been assigned to its own cluster (M=5000) is shown. On the top right, the clustering before the last two merge steps (M=17) is shown. On the bottom left, the clustering before the last merge step (M=16) is shown. Finally, on the bottom right, the final clustering (M=15) is shown. The code vectors are presented by large dots.

A detailed example of the *exact PNN* method is illustrated in Figure 3.4. In the matrix, all pairwise distances of the clusters are shown. On the right hand side of the matrix, the nearest neighbor cluster and the associated distance for each cluster are shown. Note that these data structures are not maintained by the basic *PNN* algorithm. At the beginning there are seven clusters: *a*, *b*, *c*, *d*, *e*, *f*, and *g*. During each step of the algorithm, two clusters are always merged. Thus after the first step, there are six clusters left: *ab*, *c*, *d*, *e*, *f*, and *g*. At the end of the agglomeration, there are only two clusters left: *abcd* and *efg*.

A drawback of the *PNN* method is its relatively high running time in its basic form [SO97]. There are almost N steps to be performed by the algorithm because, typically, $M<<N$. Straightforward implementation [E89] recalculates all pairwise distances of the clusters at each step of the algorithm for finding the pair of clusters to be merged in the algorithm. No additional data structures are required, but the algorithm takes $O(N^3K)$ time [SO97] because there are $O(N)$ steps and in each step there are $O(N^2)$ cost function values to be calculated. Therefore, the algorithm is very slow for large data sets.

3.2 Using a distance matrix

To reduce the number of the calculations of the merge cost function of Eq. (3), pairwise distances can be stored in an $N×N$ matrix; see Figure 3.5. A strictly upper triangular matrix, which is shown in gray, is only used because the distances between the clusters are symmetrical. The minimum value is searched for from the distance matrix and the corresponding cluster pair is merged. New merge cost values are then calculated between the new cluster and remaining clusters, only. Thus, the number of the calculations per iteration falls from $O(N^2)$ to $O(N)$, but the search for the minimum still takes $O(N^2)$. The time complexity of using the distance matrix is thus $O(N^2K+N^3)$, where the first term originates from the calculations of the merge cost values and the second term originates from the search for the minimum [SO97]. The disadvantage of this approach is its quadratic memory consumption.

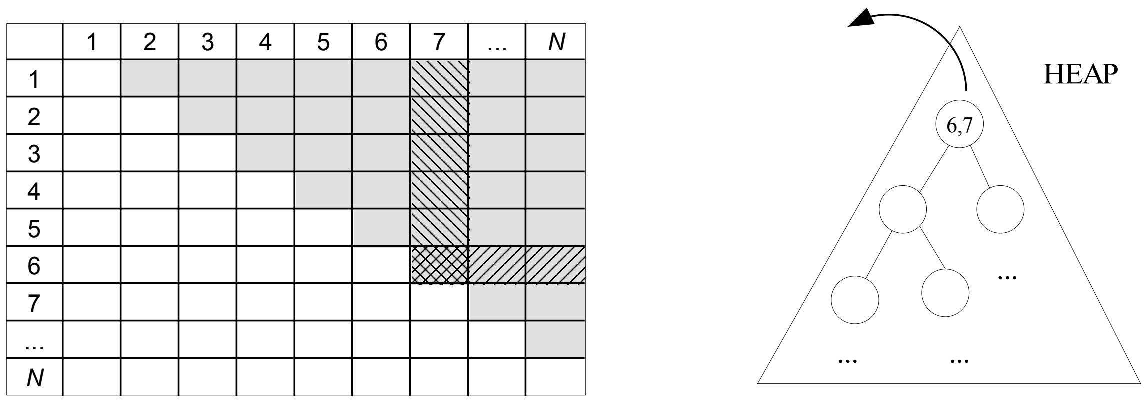


Figure 3.5. Distance matrix. A strictly upper triangular matrix shown in gray is only used because the distances between the clusters are symmetrical.

Kurita's method [K91] stores all pairwise distances into a matrix but utilizes a heap structure for searching the minimum distance; see Figure 3.5. The merged clusters can be found by taking the smallest element from the top of the heap in O(log$N$) time. Although the number of the elements in the heap is even O(N^2) in the beginning, the search can be done nevertheless in O(log$N$) time because O(log(N^2)) = O(2log$N$). Only O(N) distance updates are needed after each merge step; each of these updates takes O(K+log$N$) time. There, the first term (K) originates from the time for the calculation of the merge cost and the second term is attributed to the heap operation. *Kurita's* method thus runs in O(N^2K+N^2·log N) time; however, it still requires O(N^2) memory, which is impractical for large data sets.

3.3 Fast exact PNN

A much faster variant of the *PNN* method can be implemented by maintaining, for each cluster, a pointer to its nearest neighbor [FKSC00]. The nearest neighbor nn_a for a cluster s_a is defined as the cluster minimizing the merge cost:

$$nn_a = \arg\min_{\substack{j\in[1,N]\\ j\neq a}} d_{a,j}\,. \qquad (6)$$

The nearest neighbor property is not symmetrical, (i.e., $nn_a=b$ does not imply $nn_b=a$). The nearest neighbor of the cluster a is the cluster b, according to Eq. (3), but the nearest neighbor of the cluster b is the cluster c (see Figure 3.6 where the nearest neighbor pointers are illustrated). In this way, only a few nearest neighbor searches are needed in each iteration. The method is denoted as the *fast exact PNN*. Its implementation details are given next.

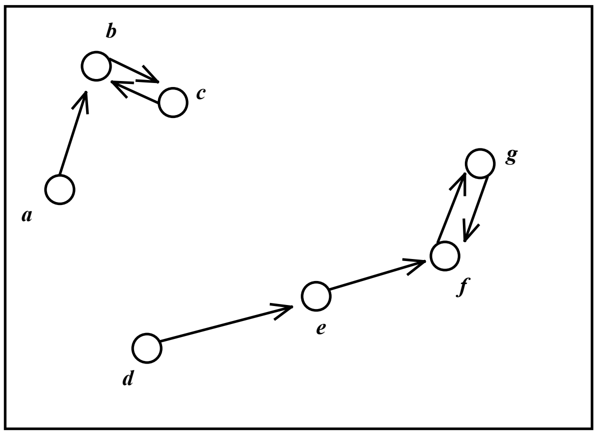


Figure 3.6. Nearest neighbor pointers, which reduce the amount of the nearest neighbor search in each iteration.

For each cluster s_j, one maintains the cluster size n_j, the corresponding cluster center c_j, and the pointer to its nearest neighbor nn_j. The nearest neighbor pointer is assigned with the cost value d_j. The cost values of pointers indicate the amount of increase in the distortion if the cluster s_j is merged to s_{nn_j}. For each data vector the algorithm maintains the index of the cluster p_i, which the data vector belongs to.

In the initialization, each data vector x_i is assigned to its own cluster of the size one and the cluster center c_i is initialized to the data vector itself:

$$p_i \leftarrow i; n_i \leftarrow 1; c_i \leftarrow x_i \; for \; all \; i \in [1, N]. \tag{7}$$

In order to generate the nearest neighbor table for the cluster centers, one must find the nearest neighbor nn_i for every cluster. This is done by considering all other clusters as tentative neighbors and by selecting the one that minimizes Eq. (3). There are $O(N^2)$ pairs to be considered and thus, the initialization phase takes $O(N^2K)$ time in total.

The optimal cluster pair (s_a and s_b) to be merged is the cluster that has the minimum d_j-distance according to Eq. (3) and its nearest neighbor nn_j:

$$a \leftarrow \arg\min_{j \in [1,N]} d_j \; ; \; b \leftarrow nn_a \, . \tag{8}$$

This pair can be found in $O(N)$ time using linear search in the nearest neighbor table. The merge of the clusters is then performed as follows. First, one updates the partition indices so that the combined cluster replaces s_a, and the cluster s_b becomes obsolete:

$$p_i \leftarrow a \text{ for all } i = 1, 2, ..., N \text{ such that } p_i = b. \tag{9}$$

In order to minimize rounding errors it is advantageous to calculate the new code vector as the centroid of the cluster. This can now be easily done by the aid of the partition indices p_i:

$$c_a \leftarrow \frac{1}{n_a} \cdot \sum_{p_i = a} x_i . \tag{10}$$

The above steps can be performed at most in O(NK) time.

The nearest neighbor nn_a for the merged cluster (now s_a) must be resolved by calculating the merge cost values of Eq. (3) between the new cluster and all other remaining clusters. This can be done in an O(NK) time.

As mentioned before, the nearest neighbor function is not symmetrical. Therefore, one must also resolve the nearest neighbor pointer for all clusters whose nearest neighbor was, before the merge, either *a,* or *b* ($nn_i=a$, or $nn_i=b$). This takes O(NK) time for a single cluster and (according to practical tests) there are approximately 3-5 clusters on average to be updated at each step of the algorithm, see [FKSC00]. The overall time complexity of the update operation is thus O(τNK), where τ denotes the number of clusters whose nearest neighbor pointer must be resolved. To sum up, the time complexity of the *fast exact PNN* method is O(τN^2K).

The range of τ is [0-N] and is closely related to the *kissing number problem* (also sometimes called the *Newton number*, *contact number*, *coordination number*, or *ligancy*), which asks the maximum number of spheres of radius one that can simultaneously touch the unit sphere in K-dimensional Euclidean space [MTTV97, CS98, W04]. The cluster merge cost values of Eq. (3), however, are not Euclidean and therefore the kissing number only applies in cases when all cluster sizes are equal. This is the case at least in the initial stage of the *PNN* method, if all clusters have same initial frequencies. Usually the initial frequency of the cluster is one, but this is not necessarily always the case (see the data sets in Section 5).

In the worst case, the same cluster can be the nearest neighbor for all the other clusters, and thus τ=O(N). This situation could appear when there is one small cluster and all the rest are large. This cannot happen in the exact *PNN* method; however, it has been reported to be possible in other situations [F00]. Thus, this is not common in practice and the connection to the kissing number (even as an open problem in the general case) indicates that τ is a function of the vector dimension K.

In a favorable case, two merged clusters are chosen randomly. Since each cluster has only one nearest neighbor pointer, a randomly chosen cluster is the nearest neighbor for one cluster for the average case. The average number of the updated pointers is therefore two because there are two clusters involved in the merge operation. However, the merged clusters are not chosen randomly, rather they tend to be located in areas of high concentrations of clusters. Thus, in practice, τ is greater than two. Ultimately, τ can be zero when two clusters far away from other clusters are merged because then it is possible that the merged new cluster is not the nearest neighbor of any other cluster.

cluster	*a*	*b*	*c*	*d*	*e*	*f*	*g*	min	*NN*
a	--	2.0	16.0	2.5	4.0	8.0	16.0	2.0	*b*
b	2.0	--	10.0	2.5	10.0	18.0	26.0	2.0	*a*
c	16.0	10.0	--	22.5	36.0	40.0	32.0	10.0	*b*
d	2.5	2.5	22.5	--	4.5	14.5	30.5	2.5	*a*
e	4.0	10.0	36.0	4.5	--	4.0	20.0	4.0	*a*
f	8.0	18.0	40.0	14.5	4.0	--	8.0	4.0	*e*
g	16.0	26.0	32.0	30.5	20.0	8.0	--	8.0	*f*

Figure 3.7. Initial distance matrix of the *PNN* example shown in Figure 3.4. The cells shown in gray contain the cluster distance to the nearest neighbor for each cluster.

The initial distance matrix of the *PNN* example shown in Figure 3.4 is illustrated in Figure 3.7. The cluster pair *a* and *b* will be merged in the first step because their merge cost is smallest. After the merge, the distance matrix is updated before the search for the next cluster pair to be merged. In the *original PNN* variant (full search), the whole matrix is recalculated during the next iteration since nothing is stored. In the *distance matrix* variant, only the values of the rows and the columns for the merged cluster *ab* must be calculated and stored in the distance matrix. In the *fast exact PNN* method, only the values of the rows and the columns for the clusters *ab*, *c*, *d* and *e* are recalculated because the nearest neighbor pointers must be updated for those clusters. In this method, only the nearest neighbor pointers to the nearest neighbors and the corresponding distance values are stored in the nearest neighbor table.

3.4 Lazy PNN

The number of distance calculations can be reduced further by delaying the update of the nearest neighbor pointers. The method is based on the *monotony property*; see [KFN99]. Suppose that the current minimal merge cost is $d(s_a, s_b)$ and the clusters s_a and s_b are merged. It is possible that the centroid of the merged cluster (c_{a+b}) becomes closer to the centroid of the third cluster s_c than c_c was in respect of the original cluster centroids (c_a and c_b); see Figure 3.8. However, the monotony property states that the merge cost $d(s_{a+b}, s_c)$ cannot be smaller than $\min\{d(s_a, s_c), d(s_b, s_c)\}$. So the minimum cluster distortion d never decreases due to the merge of the optimal cluster pair. This is formalized in the following lemma [KFN99]:

Lemma 3.1. Consider three clusters s_a, s_b, s_c with centroids c_a, c_b, c_c and the number of objects in these clusters n_a, n_b, n_c. Assume that $d(s_a, s_b) \le d(s_a, s_c) \le d(s_b, s_c)$ and n_a, n_b, $n_c \ge 1$. Then it holds that $d(s_a, s_c) \le d(s_{a+b}, s_c)$.

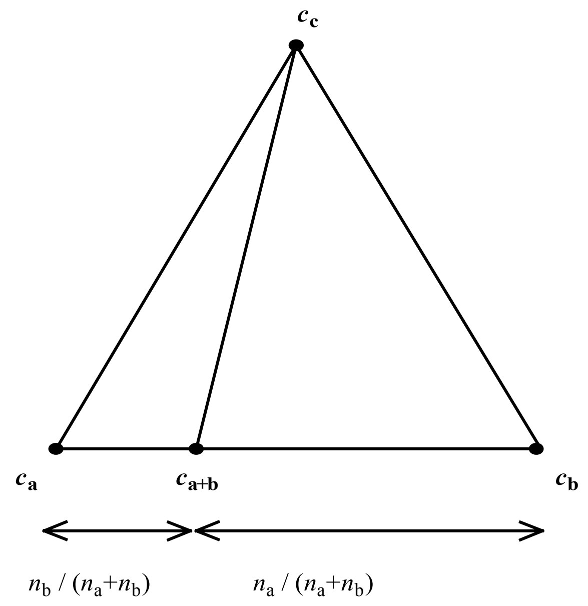


Figure 3.8. Illustration of the three clusters in 2-dimensional space. Here c_a and c_b are the centroids of the two clusters to be merged, c_{a+b} is the centroid of the merged cluster, and c_c is the centroid of any other cluster.

For example, assume that the nearest neighbor for a cluster s_i was s_a before the merge of the two clusters s_a and s_b, and s_x after the merge. From the monotony property one knows that $d_{i,a} \le d_{i,a+b}$. One therefore does not need the exact cluster distance because the previous cluster distance value serves as a lower bound. In practice, one can assume that the nearest neighbor after the merge is s_{a+b} so the previous cost function value $d_{i,a}$ (or $d_{i,b}$) can be used as a lower bound. The cluster dissimilarity value is labeled as outdated. It will be updated only if it becomes a candidate for being the smallest distance of all. In this way, one can reduce the computation by about 35% while preserving the exactness of the *PNN* method [KFN99].

3.5 Inexact variants

Several inexact variants of the *PNN* method have been considered in the literature. These variants decrease the running time at the cost of increased distortion. Equitz proposed an $O(NK \cdot \log N)$ time variant, which he referred as the *fast PNN* [E89]. It uses a KD-tree [B75, FBF77] for localizing the search for the code vectors. The algorithm merges several vector pairs at the same time. The KD-tree structure and the corresponding partition of the 2-dimensional space are illustrated in Figure 3.9. All data vectors are assigned to the buckets of the KD-tree so that the similar data vectors are in the same bucket. The method; however, has not gained as much popularity as the *exact PNN* method, probably because of its more complex implementation and suboptimal results.

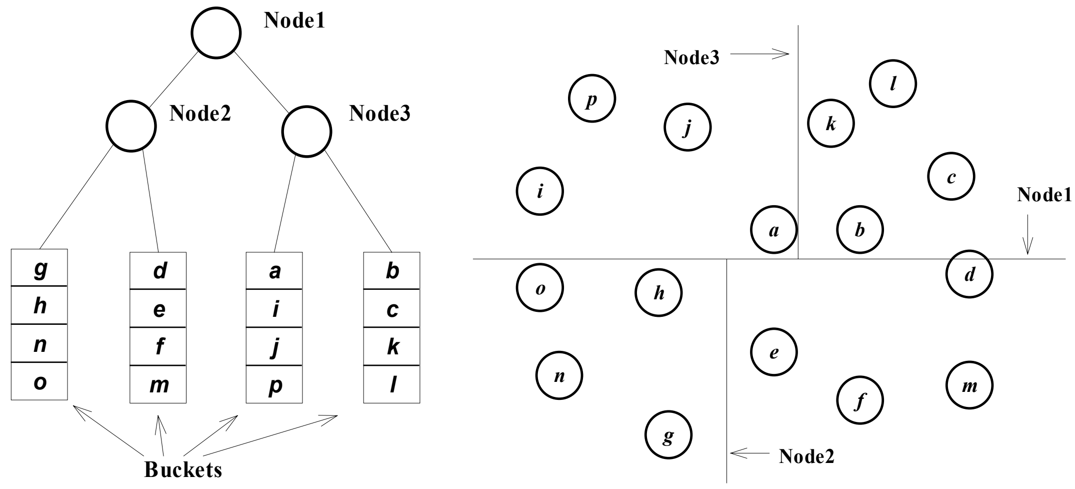


Figure 3.9. KD-tree and the corresponding partition.

Another possibility is to generate a preliminary codebook of size M_0 ($N > M_0 > M$) using the *GLA* with a random initialization of the data vectors. Any other fast algorithm could be used as well. The *exact PNN* method is then applied until the codebook reaches its final size M [GPF95]. The *exact PNN* method has now only (M_0 - M) steps, and the time complexity of this method is thus $O(NM_0) + O(M_0^3)$, where the first term is due to the *GLA* and the second term is due to the *PNN* method.

Thus, one has several different possibilities to combine the *GLA* and the *PNN* method:

(1) *Random initialization + GLA*
(2) *PNN*
(3) *PNN + GLA*
(4) *GLA + PNN*
(5) *GLA + PNN + GLA*.

In the first case, one uses the *GLA* by itself with a random initialization, which can be done very fast by choosing random data vectors as cluster centroids. In the second case, one uses the *PNN* method by itself. Since it has quadratic running time, this method is quite slow with a large data set. In the third case, *exact PNN* method is used first. After that, one fine-tunes the codebook with the *GLA*. In that way, one gets better codebooks than with the *PNN* method or the *GLA* alone.

Table 3.1. Time complexities of the different scenarios of combining the *PNN* and G*LA*.

Combination:	**Time complexity**	
	In general	Assuming $M_0 = O(M)$
Random + GLA	$O(gNMK)$	$O(gNMK)$
PNN	$O(\tau N^2 K)$	$O(\tau N^2 K)$
PNN + GLA	$O(\tau N^2 K) + O(gNMK)$	$O(\tau N^2 K)$
GLA + PNN	$O(gNM_0K) + O(\tau M_0^2 K)$	$O(gNMK)$
GLA + PNN + GLA	$O(gNM_0K) + O(\tau M_0^2 K) + O(gNMK)$	$O(gNMK)$

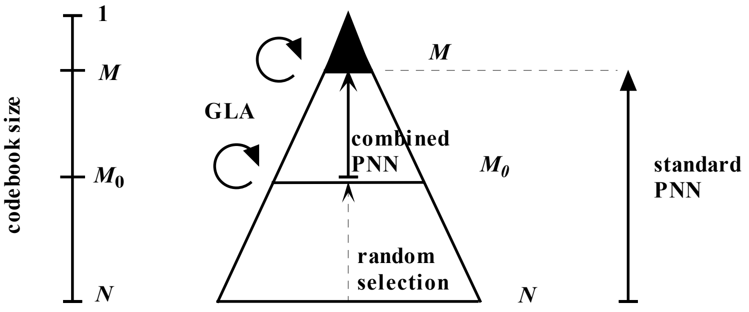


Figure 3.10. Different scenarios of combining the *PNN* and *GLA*.

In the fourth case, one first uses the *GLA* to generate a preliminary codebook of size M_0, and then applies the *exact PNN* method until the codebook reaches the final size, *M*. In this way, one can speed-up the *PNN* method remarkably yielding to $O(NM_0K) + O(\tau M_0^2 K)$ time complexity. Unfortunately, then the exactness of the *PNN* method cannot be preserved. In the fifth case, one first uses the *GLA* to generate a preliminary codebook of size M_0 and then applies the *exact PNN* method until the codebook reaches the final size, *M*. The codebook is finally fine-tuned with the *GLA*. Thus, one can speed-up the *PNN* method like in the third case. Even though the exactness of the *PNN* method is then lost, one can still get with this method a better codebook compared to the fourth case. The time complexities of the different combination scenarios are summarized in Table 3.1, where *g* denotes the number of iterations of the *GLA*. For illustration of these variants, see Figure 3.10.

3.6 Genetic algorithm

Genetic algorithms (*GA*) are based on the model of natural selection that occurs in nature. The main idea is to maintain a set of solutions (called a *population of individuals*) that are iteratively manipulated using *genetic operations* (*crossover* and *mutations*) and *selection*. Each initial solution is created by selecting M random data vectors as the code vectors and by calculating the optimal partition. Crossing the best solutions of the current population then creates the solutions for the next population. The number of iterations and the population size are the main parameters of the algorithm.

Several *genetic algorithms* have been previously considered for the clustering problem [DK95, PMJ95, MC96, S96, FKKN97, KM99, F00, TY00, KFN03]. The best, known (or best, known) *genetic algorithms* use a crossover method, which is based on the *PNN* method [FKKN97, F00, KFN03]. The crossover proposed in [F00] will be described below.

The *PNN crossover* starts by merging two parent codebooks by taking their union. The new partition of a data vectors is then constructed based on the existing partitions P^1 and P^2 [F00]. The partition of the data vector x_i is either p_i^1 or p_i^2. The one with the smaller distance to x_i is chosen. In this way, the new partition can be generated using $2{\cdot}N$ distance calculations only. The new codebook is then updated in respect to the new partition. This procedure gives a solution in which the size of the codebook is twice the size of the final codebook. A final task is to reduce the codebook size from $2{\cdot}M$ to M using the exact *PNN* method with the following two differences. First, one does not perform the *PNN* method for a full data set. Rather, one starts from an initial solution of $2{\cdot}M$ clusters at most. The crossover can therefore be performed in $O(\tau M^2)$ time instead of the original $O(\tau N^2)$ time. Second, the partition data is also updated during the crossover and, therefore, the partition is not needed to be recalculated after the *PNN* method. Figure 3.11 illustrates the *PNN* crossover.

The revised version of [F00] has three vital improvements over [FKKN97]: (i) The representation of solution is revised so that one does not merge only the codebooks, but maintain both partition and codebook for each solution. In this way, the partition of a new solution can efficiently be computed from those of the parent solutions. Access to the partition also gives a more precise initialization for the *PNN* method, which results in higher quality candidate solutions. (ii) Empty partitions are removed before the application of the *PNN* method, which is vital for avoiding the worst-case behavior of the *PNN* method. (iii) Since the new candidate solutions are already close to a local minimum, the *GLA* iterations are extremely fast using the activity detection technique [KFN99].

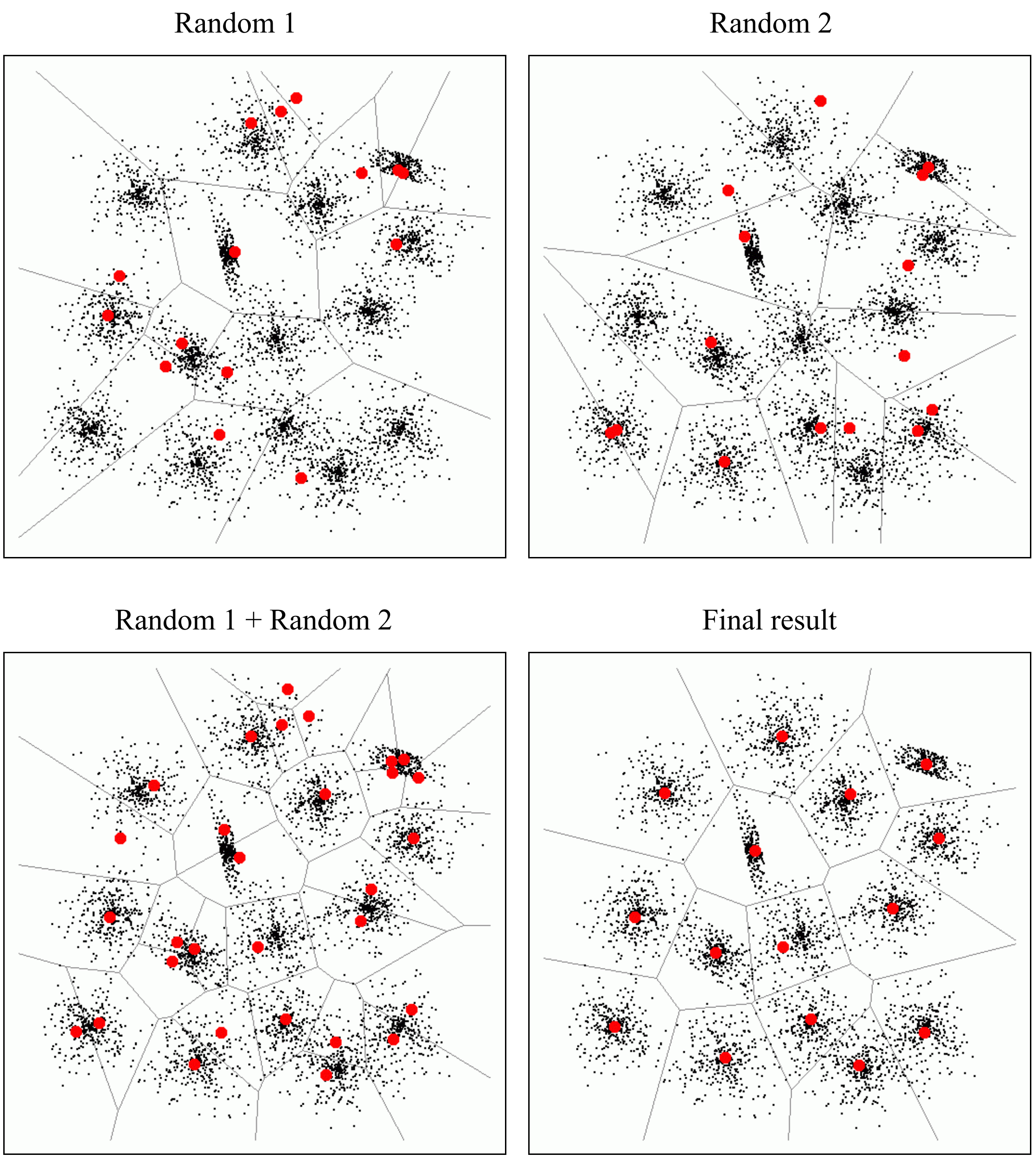


Figure 3.11. Illustration of the use of the *PNN* method as a deterministic crossover method in the *genetic algorithm* for the data set S_2. Panels on top left and right show two initial codebooks, which have been generated randomly among the data vectors (M=15). The panel on bottom left shows the codebook after combining two initial codebooks (M=30). On the bottom right the final codebook after the 15 merge steps of the *PNN* method (M=15) is shown.

According to the experiments in [F00], the *genetic algorithm* with the *PNN crossover* method outperforms all the comparative methods, including the previous variants of the genetic algorithm. The only other method that has been reported to give better results is the *self-adaptive genetic algorithm* (*SAGA)* [KFN03], which still uses the *PNN crossover* as the key component. The use of the *PNN* method as a deterministic crossover method also achieves a fast convergence with a rather small population size. The algorithm is therefore remarkably faster than any of the previously reported genetic algorithms.

3.7 Summary

The results of the *PNN* method are quite good; in general, they are better than the results of the opposite hierarchical methods performing divisions of greater clusters. The *PNN* method does not suffer from bad data sets as *k-means* does. It can be expected that the results of the *PNN* method are quite good, in general. The hierarchical property of the *PNN* method also makes it possible to get the clustering results of many different levels with the same effort. It is the best, known clustering method when used with the *genetic algorithm* as a crossover method. The *PNN* method is simple and it can be easily applied to many other problems.

On the other hand, the *PNN* method is slow. The running time of the best *PNN* algorithm is lower bounded by $\Omega(N^2)$ and there are not any clear, simple and practical ways to speed-up the algorithm. Since the merge phase is based on the local optimization strategy and is a greedy heuristic, there is no guarantee of the global optimum. If the number of the clusters is small, the result can remain quite far away from the best possible result. Thus, there is a need for improvement, but it is not obvious how this could be done.

4 Summary of the publications

In the first paper [**P1**], we study methods for speeding-up the *pairwise nearest neighbor* method. We consider two different speed-up methods. The first is the *partial distortion search* (*PDS*) originally proposed by Bei and Gray [BG85] for *k*-means. It terminates a single distance calculation immediately when the partial distance exceeds the shortest distance that has been found so far. Since the idea is independent of the chosen metrics, it can, therefore, also be directly applied to the *PNN* method.

The second method is the *mean-distance-ordered partial search* (*MPS*) technique introduced by Ra and Kim [RK93]. This technique uses the component means of the vectors to derive a precondition for the distance calculations and, in this way, a large number of the distance calculations can be omitted completely. Although this idea utilizes properties of the Euclidean space, we will show that the precondition can also be generalized for the distance calculations in the *PNN* method.

When combining all the speed-up methods discussed in this paper, one can reduce the run time to 8-15% of the basic version of the *PNN* method in the case of four favorable training sets. We can reduce the run time to about 50% of the basic version in the case of the less favorable sets (residual vectors). When the dimensions of the vectors are very large (256 or greater), the run time can be reduced to 2% of the basic version. We also demonstrate that the improvements are applicable within hybrid methods such as the *GLA-PNN-GLA*, and the *GA with PNN crossover*.

In the second paper [**P2**], we propose an algorithm for fast agglomerative clustering using a *k* nearest neighbor graph. In our approach, every node in the graph represents a cluster. The edges of the graph represent inter cluster connections between nearby clusters. The graph has a linear space complexity and it is used as a search structure for reducing the number of distance calculations. The proposed approach has two specific problems: how to generate the graph efficiently, and how to utilize it. We propose solutions for the first problem by considering a *KD-tree* [B75], [FBF77], *divide-and-conquer* [PS85], and *projection-based search* [RK93]. It appears that the *projection-based heuristic* works reasonably well in most cases. The *divide-and-conquer* approach is faster in the case of some high dimensional image data sets, and the *KD-tree* is faster in the case of 3-dimensional color clustering.

We also study the second problem and found that a relatively small neighborhood size (k=3...6) is sufficient to produce clustering with similar quality to that of a full search. At the same time, significantly fewer distance calculations and operations are needed and, therefore, remarkable speed-up is achieved. The running time is comparable to that of the *k-means* with a lower distortion than that with the *k-means*. The improvement due to the neighborhood graph is significant.

In the third paper [**P3**], we introduce a new *multilevel thresholding* algorithm for image processing. The algorithm is derived from the *pairwise nearest neighbor* (*PNN*) method. The time complexity of the *PNN* method is lower bounded by $\Omega(N^2)$ in vector quantization. It is therefore not self-evident that the *PNN* method could be useful in real-time applications. For example, if the implementation is made poorly, the time complexity is $O(N^2)$ as in [CC03]. Our contribution is to show that *PNN thresholding* can be implemented in $O(N \cdot \log N)$ time.

Unlike in vector quantization, the one-dimensionality of the histogram of the image can be utilized so that the neighbor classes can be determined by using a simple linked list structure. This allows constant time updates of the data structures. At the same time, we use a heap structure for the search of the minimum cost class pair. The proposed method works in real time for any number of thresholds. Experiments also show that the proposed method, when combined with the *Lloyd-Max quantizer*, provides MSE values that are much closer to that of *optimal thresholding* than using *LMQ* alone. The difference is small when a low number of thresholds are needed (M = 2 or 3), but the difference is significant when the number of thresholds is higher (from M = 10 to M = 20).

In the fourth paper [**P4**], we propose a more general approach for agglomerative clustering called *iterative shrinking* (*IS*) that generates the clustering by a sequence of cluster removal operations. In the *IS* method, clusters are removed one at a time by reassigning the vectors in the removed cluster to the remaining nearby clusters. The *PNN* method can be considered a special case of the *IS* method since it removes one cluster and forces the vectors to move to the same neighbor-cluster. In the *IS* method, the vectors can be reassigned more freely. Apart from the difference in the removal operation, we follow the local optimization strategy of the *PNN* method and always remove the cluster that increases the cost function value least.

Experimental results show that the proposed method achieves better results than the comparative methods at the cost of slower speed. The time complexity of the method varies from $O(N^2)$ to $O(N^2 \cdot \log^2 N)$ depending on the variant. The proposed method was also applied as a crossover method in the *genetic algorithm* (*GAIS*). Experiments indicate that the proposed combination outperforms other clustering algorithms in terms of minimizing distortion. The *iterative shrinking* method also extends to the case where the number of the clusters must be determined simply by changing the optimization function. This does not add to the complexity since the solutions for variable number of clusters can be found during a single run of the proposed algorithm.

In the fifth paper [P5], we present an optimal clustering algorithm that was motivated by the *PNN* method. Optimal clustering can be found by considering all possible merge sequences and finding the one that minimizes the optimization function. The idea can be implemented as a *branch-and-bound* (*BB*) *technique* that uses a search tree for finding the optimal clustering. We present also two suboptimal, but polynomial, time variants from the branch-and-bound technique.

The time complexity of the optimal 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 two polynomial time algorithms from the branch-and-bound technique offer a good compromise between the optimal algorithm and the *PNN* method. They can operate with larger data sets than the optimal branch-and-bound technique.

The contributions of the papers can be briefly summarized as follows. All the ideas have been developed through the teamwork of all authors. The author of this thesis is responsible for all the new implementations in this work, has run most of the experiments in this thesis and has made significant contributions to the writing of the papers. The order of the authors is determined by the estimated overall contribution to the writing of the papers.

5 Summary of results

Next, the results of the proposed methods used in this thesis are summarized. For the comparisons, the following methods are selected:

(1) The *fast exact PNN* method [FKSC00].

(2) The *PNN with the speed-up* methods [**P1**].

(3) The *graph-based PNN* method [**P2**].

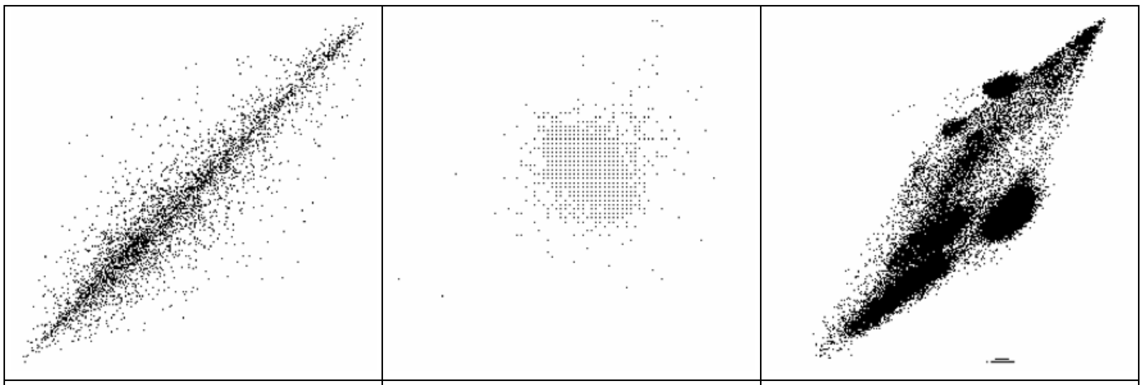
(4) The *iterative shrinking* method [**P4**].

(5) The *k-means* algorithm [KFN00].

(6) The *genetic algorithm* with *iterative shrinking* as crossover (*GAIS*) [**P4**].

Spatial vectors:	Spatial residual vectors:	Color vectors:
Bridge (256×256) K=16, N=4096	*Miss America* (360×288) K=16, N=6480	*House* (256×256) K=3, N=34112*

Figure 5.1. Image data sets used in this thesis. *Duplicate training vectors are combined and frequency information is stored. Note that when duplicate vectors are merged, all distance and merge cost calculations must always be multiplied by the frequency of the data vectors representing multiple instances of the original data set.

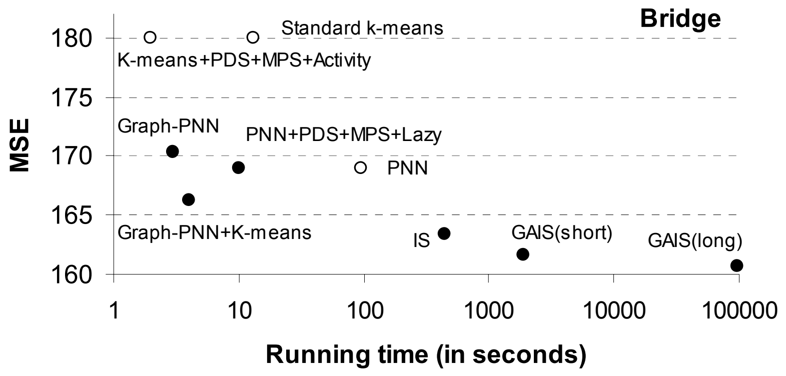


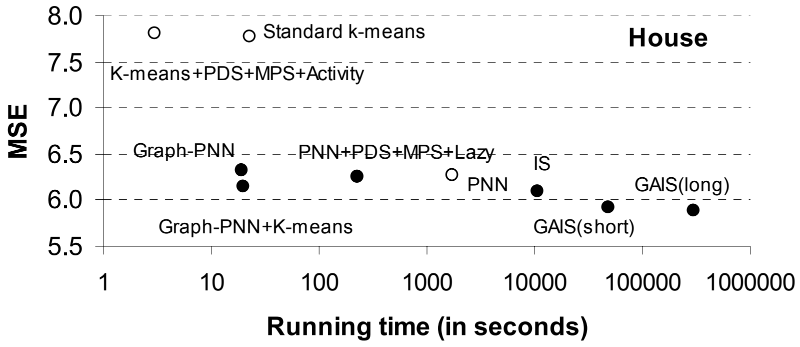
Data set *Bridge*	Data set *Miss America*	Data set *House*

Figure 5.2. Two-dimensional plots of the image data sets used in the experiments.

The experiments were performed for three data sets generated from different images; see Figures 5.1 and 5.2. The gray-scale image *Bridge* was divided into non-overlapping 4×4 pixel blocks. The data set *House* consists of color values of the *RGB* image. The third data set, *Miss America*, has been obtained by subtracting two subsequent image frames of the original video image sequence, and then constructing 4×4 spatial pixel blocks from the residuals. Only the first two frames have been used. In the two-dimensional plots, only the data of the first two dimensions of the image data sets are shown and the scale of each axis is from 0 to the maximum value of that dimension. The algorithms are coded in *DJGPP C* Version 2.01 and are run on a 450 MHz Pentium III personal computer that uses Microsoft Windows 98 Operating system.

Memory consumption of the methods is linear, except for the *GA* that stores *S* codebooks and mappings, and thus the *GA* consumes $O(NK + S(MK+N))$ space. Figure 5.3 illustrates the MSE values for the different methods as a function of running time. The summary of the MSE values and the running times as a function of codebook size is summarized in Table 5.1. Comparisons that are more detailed (or comparisons that are more detailed or more comparisons that are detailed?) are included in the individual papers.





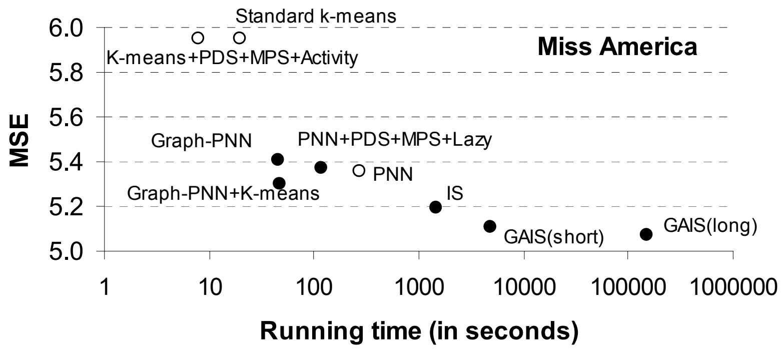


Figure 5.3. MSE values for the different methods as a function of running time.

Table 5.1. MSE values and running times for the different methods as a function of codebook size.

Bridge	MSE						Running time (in seconds)					
M	32	64	128	256	512	1024	32	64	128	256	512	1024
PNN	326.24	264.91	214.88	168.92	121.68	72.61	96	96	96	96	95	92
P1	326.24	264.91	214.88	168.92	121.68	72.61	11	10	10	10	10	9
P2	325.04	267.48	217.22	171.11	123.90	74.92	3	3	4	3	3	3
P4	318.07	258.36	209.42	163.38	117.41	70.52	469	465	459	444	413	362

House	MSE						Running time (in seconds)					
M	32	64	128	256	512	1024	32	64	128	256	512	1024
PNN	24.13	15.18	9.83	6.27	3.91	2.38	1734	1737	1736	1742	1736	1743
P1	24.04	15.27	9.85	6.26	3.92	2.39	230	231	230	226	227	229
P2	27.85	16.20	10.09	6.37	4.00	2.43	20	20	20	20	20	20
P4	23.65	14.83	9.56	6.09	3.79	2.29	10508	10449	10555	10569	10409	10113

Miss America	MSE						Running time (in seconds)					
M	32	64	128	256	512	1024	32	64	128	256	512	1024
PNN	9.04	7.81	6.60	5.36	4.11	2.85	268	266	268	270	269	264
P1	9.04	7.81	6.60	5.37	4.11	2.84	117	117	117	117	117	116
P2	9.04	7.83	6.64	5.44	4.19	2.90	48	47	47	47	47	47
P4	8.92	7.69	6.43	5.19	3.95	2.72	1580	1577	1553	1466	1390	1220

6 Conclusions

The main contributions of this thesis can be summarized as follows:

We have developed several speed-up improvements for the *PNN* method based on projection-based search, partial distortion search, and the use of a k nearest neighbor graph. The last idea could also be applied to other clustering algorithms that require a large number of distance calculations.

An efficient $O(N \cdot \log N)$ time implementation of the *PNN* method has been given for the 1-dimensional special case.

A generalization of the merge phase was proposed by introducing similar decremental clustering algorithms based on a slightly improved cluster removal operation. The results are systematically better than that of the *PNN* method although the differences are often small. However, in certain cases, the difference between the proposed *iterative shrinking* approach and the *PNN* method was decisive: the proposed method found the correct number of clusters (15), whereas the *PNN* method found only 14 clusters.

The merge approach of the *PNN* method was also used for generating optimal clustering, although the result is mainly theoretical. The idea, however, can have practical implications as was shown by introducing two polynomial time variants, motivated by the proposed optimal branch-and-bound algorithm.

7 References

[A73] M.R. Anderberg, *Cluster Analysis for Applications*, Academic Press, Inc., New York, NY.

[A89] A.S. Abutaleb, Automatic thresholding of gray-level pictures using two-dimensional entropy, *Computer Vision, Graphics, and Image Processing*, 47(1): 22-32, July 1989.

[AM93] S. Arya and D.M. Mount, Algorithm for fast vector quantization, *Proceedings of Data Compression Conference*, Snowbird, Utah, 381-390, 1993.

[AMNSW98] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman and A.Y. Wu, An optimal algorithm for approximate nearest neighbor searching in fixed dimensions, *Journal of the ACM*, 45(6): 891-923, November 1998.

[AP02] P.K. Agarwal and C.M. Procopiuc, Exact and approximation algorithms for clustering, *Algorithmica*, 33(2): 201-226, 2002.

[AR96] M.B. Al-Daoud and S.A. Roberts, New methods for the initialisation of clusters, *Pattern Recognition Letters*, 17(5): 451-455, May 1996.

[AST94] A. Aggarwal, B. Schiever and T. Tokuyama, Finding a minimum-weight *k*-link path in graphs with concave Monge-property and applications, *Discrete & Computational Geometry*, 12: 263-280, 1994.

[B57] R. Bellman, *Dynamic programming*, Princeton University Press, New Jersey, 1957.

[B75] J.L. Bentley, Multidimensional binary search trees used for associative searching, *Communications of the ACM*, 18(9): 509-517, September 1975.

[B80] J.L. Bentley, Multidimensional divide-and-conquer, *Communications of the ACM*, 23(4): 214-229, April 1980.

[B86] J. Bernsen, Dynamic thresholding of grey-level images, *Proceedings of International Conference on Pattern Recognition (ICPR'86)*, Paris, France, 1251-1255, 1986.

[B90] J.L. Bentley, K-d trees for semidynamic point sets, *Proceedings of the 6th Annual ACM Symposium on Computational Geometry*, Berkley, CA, USA, 187-197, June 1990.

[B92] A.D. Brink, Thresholding of digital images using two-dimensional entropies, *Pattern Recognition*, 25(8): 803-808, August 1992.

[B99] J. Boberg, *Cluster Analysis: A Mathematical Approach with Applications to Protein Structures*, Ph.D. Thesis, TUCS Dissertations 20, University of Turku, Turku, Finland, October 1999.

[B02] P. Berkhin, Survey of clustering data mining techniques, Research paper, Accrue Software, 2002. http://www.accrue.com/products/rp_cluster_review.ps, 11.10.2002.

[B04] S. Bandyopadhyay, An automatic shape independent clustering technique, *Pattern Recognition*, 37(1): 33-45, January 2004.

[BBK01] C. Böhm, S. Berchtold and D.A. Kleim, Searching in high-dimensional spaces - index structures for improving the performance of multimedia databases, *ACM Computing Surveys*, 33(3): 322-373, September 2001.

[BBS99] S.J. Baek, M.J. Bae and K.-M. Sung, A fast vector quantization encoding algorithm using multiple projection axes, *Signal Processing*, 75(1): 89-92, January 1999.

[BCQY97] M.R. Brito, E.L. Chávez, A.J. Quiroz and J.E. Yukich, Connectivity of the mutual *k*-nearest-neighbor graph in clustering and outlier detection, *Statistics & Probability Letters*, 35(1): 33-42, August 1997.

[BG85] C.-D. Bei and R.M. Gray, An improvement of the minimum distortion encoding algorithm for vector quantization, *IEEE Transactions on Communications,* 33(10): 1132-1133, October 1985.

[BH65] G.H. Ball and D.J. Hall, ISODATA, a novel method of data analysis and classification, *Technical report AD-699616*, Stanford Research Institute, Stanford, CA, USA, 1965.

[BJLS98] S.J. Baek, B.K. Jeon, D. Lee and K.-M. Sung, Fast clustering algorithm for vector quantization, *Electronics Letters*, 34(2): 151-152, January 1998.

[BJS97] S.J. Baek, B.K. Jeon and K.-M. Sung, A fast encoding algorithm for vector quantization, *IEEE Signal Processing Letters*, 4(12): 325-327, December 1997.

[BM93] G.P. Babu and M.N. Murty, A near-optimal initial seed value selection in *k*-means algorithm using a genetic algorithm, *Pattern Recognition Letters*, 14(10): 763-769, October 1993.

[BM95] G.P. Babu and M.N. Murty, Optimal thresholding using multi-state stochastic connectionist approach, *Pattern Recognition Letters*, 16(1): 11-18, January 1995.

[BM02] S. Bandyopadhyay and U. Maulik, Efficient prototype reordering in nearest neighbor classification, *Pattern Recognition*, 35(12): 2791-2799, December 2002.

[BN03] M. Belkin and P. Niyogi, Laplacian eigenmaps for dimensionality reduction and data representation, *Neural Computation*, 15(6): 1373-1396, June 2003.

[BP98] J.C. Bezdek and N.R. Pal, Some new indexes of cluster validity, *IEEE Transactions on Systems, Man and Cybernetics-Part B: Cybernetics*, 28(3): 301-315, June 1998.

[BS76] J.L. Bentley and M.I. Shamos, Divide-and-conquer in multidimensional space, *Proceedings of the 8th Annual ACM Symposium on the Theory of Computing*, 220-230, Hershey, PA, USA, May 1976.

[BS93] J. Boberg and T. Salakoski, General formulation and evaluation of agglomerative clustering methods with metric and non-metric distances, *Pattern Recognition*, 26(9), 1395-1406, September 1993.

[C95a] C.-H. Cheng, A branch and bound clustering algorithm, *IEEE Transactions on Systems, Man, and Cybernetics*, 25(5): 895-898, May 1995.

[C95b] Y. Cheng, Mean shift, mode seeking, and clustering, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8): 790-799, August 1995.

[CBC00] A.D. Constantinou, D.R. Bull and C.N. Canagarajah, A new class of VQ codebook design algorithms using adjacency maps, *SPIE Electronics Imaging 2000*, San Jose, 3974: 625-634, 2000.

[CC03] K.-L. Chung and W.-Y. Chen, Fast adaptive PNN-based thresholding algorithms, *Pattern Recognition*, 36(12): 2793-2804, December 2003.

[CCLH97] C.-W. Chao, C.-C. Chiu, P.-C. Lu and C.-H. Hsieh, Codebook design for vector quantization of images based on fuzzy *c*-means clustering algorithm, *Optical Engineering*, 36(2): 580-587, February 1997.

[CH91] S.-H. Chen and W.M. Hsieh, Fast algorithm for VQ codebook design, *Communications, Speech and Vision*, *IEE Proceedings I*, 138(5): 357-362, October 1991.

[CK95] P.B. Callahan and S.R. Kosaraju, A decomposition of multidimensional point sets with application to *k*-nearest-neighbors and *n*-body potential fields, *Journal of the Association for Computing Machinery*, 42(1): 67-90, January 1995.

[CKS95] C.-Q. Chen, S.-N. Koh and P. Sivaprakasapillai, VQ codebook design algorithm based on partial GLA, *Electronics Letters*, 31(21): 1803-1805, October 1995.

[CL96] S.-M. Cheng and K.-T. Lo, Fast clustering process for vector quantization codebook design, *Electronics Letters*, 32(4): 311-312, February 1996.

[CNBM01] E. Chávez, G. Navarro, R. Baeza-Yates and J.L. Marroquín, Searching in metric spaces, *ACM Computing Surveys*, 33(3): 273-321, September 2001.

[CO72] K.M. Cunningham and J.C. Ogilvie, Evaluation of hierarchical grouping techniques: a preliminary study, *The Computer Journal*, 15(3): 209-213, August 1972.

[CS98] J.H. Conway and N.J.A. Sloane, *Sphere Packings, Lattices and Groups* (3rd edition), Springler Verlag, New York, 1998.

[CSS98] M. Cheriet, J.N. Said and C.Y. Suen, A recursive thresholding technique for image segmentation, *IEEE Transactions on Image Processing*, 7(6): 918-921, June 1998.

[CTC94] T.-D. Chiueh, T.-T. Tang and L.-G. Chen, Vector quantization using tree-structured self-organizing feature maps, *IEEE Journal on Selected Areas in Communications*, 12(9): 1594-1599, December 1994.

[CW92] R. Conzales and R. Woods, *Digital image processing*, Addison-Welsey Inc., 443-457, 1992.

[CWY94] W.-T. Chen, C.-H. Wen and C.-W. Yang, A fast two-dimensional entropic thresholding algorithm, *Pattern Recognition*, 27(7): 885-893, July 1994.

[D73] J.C. Dunn, A fuzzy relative of the ISODATA process and its use in detecting compact, well-separated clusters, *Journal of Cybernetics*, 3(3): 32-57, March 1973.

[D87] R.C. Dubes, How many clusters are best?--an experiment, *Pattern Recognition*, 20(6): 645-663, 1987.

[D93] R.C. Dubes, Cluster analysis and related issues, in: *Handbook of Pattern Recognition & Computer Vision*, C.H. Chen, L.F. Pau and P.S.P. Wang (eds.), World Scientific Publishing Co., Inc., River Edge, NJ, 3-32, 1993.

[DB79] D.L. Davies and D.W. Bouldin, A cluster separation measure, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1(2): 224-227, 1979.

[DCSL02] M. Dash, K. Choi, P. Scheuermann and H. Liu, Feature selection for clustering – a filter solution, *Proceedings of IEEE International Conference on Data Mining (ICDM'02)*, 115-122, 2002.

[DE96] M.T. Dickerson and D. Eppstein, Algorithms for proximity problems in higher dimensions, *Computational Geometry*, 5(5): 277-291, January 1996.

[DK95] V. Delport and M. Koschorreck, Genetic algorithm for codebook design in vector quantisation, *Electronics Letters*, 31(2): 84-85, January 1995.

[DL94] V. Delport and D. Liesch, Fuzzy-c-mean algorithm for codebook design in vector quantisation, *Electronics Letters*, 30(13): 1025-1026, June 1994.

[DS76] E. Diday and J.C. Simon, Clustering analysis, in: *Digital Pattern Recognition*, K.S. Fu (ed), Springer-Verlag, Secaucus, NJ, 47-94, 1976.

[E89] W.H. Equitz, A new vector quantization clustering algorithm, *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(10): 1568-1575, October 1989.

[E92] B.S. Everitt, *Cluster Analysis* (3rd edition), Edward Arnold / Halsted Press, London, 1992.

[EE94] D. Eppstein and J. Erickson, Iterated nearest neighbors and finding minimal polytopes, *Discrete & Computational Geometry*, 11: 321-350, 1994.

[EPY97] D. Eppstein, M.S. Paterson and F.F. Yao, On nearest-neighbor graphs, *Discrete & Computational Geometry*, 17(3): 263-282, April 1997.

[F65] E.W. Forgy, Cluster analysis of multivariate data: efficiency vs. interpretability of classification, *Biometrics*, 21(3): 768-769, 1965.

[F87] D.H. Fisher, Knowledge acquisition via incremental conceptual clustering, *Machine Learning*, 2(2): 139-172, September 1987.

[F97] B. Fritzke, The LBG-U method for vector quantization – an improvement over LBG inspired from from neural networks, *Neural Processing Letters*, 5(1), January 1997.

[F99] P. Fränti, On the usefulness of self-organizing maps for the clustering problem in vector quantization, *Proceedings of the 11th Scandinavian Conference on Image Analysis (SCIA'99)*, Kangerlussuaq, Greenland, 1: 415-422, 1999.

[F00] P. Fränti, Genetic algorithm with deterministic crossover for vector quantization, *Pattern Recognition Letters*, 21(1): 61-68, January 2000.

[FBF77] J.H. Friedman, J.L. Bentley and R.A. Finkel, An algorithm for finding best matches in logarithmic expected time, *ACM Transactions on Mathematical Software*, 3(3): 209-226, September 1977.

[FBS75] J.H. Friedman, F. Baskett, L.J. Shustek, An algorithm for finding nearest neighbors, *IEEE Transactions on Computers*, C-24(10): 1000-1006, October 1975.

[FG88] T. Feder and D.H. Greene, Optimal algorithms for approximate clustering, *Proceedings of the 20th annual ACM symposium on Theory of computing*, Chicago, Illinois, 434-444, May 1988.

[FGGKKLN00] P. Fränti, H.G. Gyllenberg, M. Gyllenberg, J. Kivijärvi, T. Koski, T. Lund and O. Nevalainen, Minimizing stochastic complexity using local search and GLA with applications to classification of bacteria, *Biosystems*, 57(1): 37-48, June 2000.

[FGP90] W.A. Finamore, D.P. de Garrido and W.A. Pearlman, Clustering algorithm for entropy-constrained vector quantizer design, *SPIE Proceedings of Visual Communications and Image Processing*, 1360: 837-846, September 1990.

[FK97] H. Frigui and R. Krishnapuram, Clustering by competitive agglomeration, *Pattern Recognition*, 30(7): 1109-1119, July 1997.

[FK00] P. Fränti and J. Kivijärvi, Randomized local search algorithm for the clustering problem, *Pattern Analysis and Applications*, 3(4): 358-369, December 2000.

[FKKN97] P. Fränti, J. Kivijärvi, T. Kaukoranta and O. Nevalainen, Genetic algorithms for large scale clustering problem, *The Computer Journal*, 40(9): 547-554, 1997.

[FKN97] P. Fränti, T. Kaukoranta and O. Nevalainen, On the splitting method for VQ codebook generation, *Optical Engineering*, 36(11): 3043-3051, November 1997.

[FKSC00] 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.

[FN75] K. Fukunaga and P.M. Narendra, A branch and bound algorithm for computing *k*-nearest neighbors, *IEEE Transactions on Computers*, 24(7): 750-753, July 1975.

[FV02] P. Fränti and O. Virmajoki, Polynomial-time clustering algorithms derived from branch-and-bound technique, *Advanced Consepts for Intelligent Vision Systems (ACIVS'2002)*, Ghent, Belgium, 118-123, September 2002.

[FV03] P. Fränti and O. Virmajoki, Genetic algorithm using iterative shrinking for solving clustering problems, *Proceedings of Wessex Data Mining Conference 2003*, Rio de Janeiro, Brazil, 193-204, December 2003.

[FVH03] P. Fränti, O. Virmajoki and V. Hautamäki, Fast PNN-based clustering using *k*-nearest neighbor graph, *IEEE International Conference on Data Mining (ICDM'03)*, Melbourne, Florida, USA, 525-528, November 2003.

[FVK02] P. Fränti, O. Virmajoki and T. Kaukoranta, Branch-and-bound tecnique for solving optimal clustering, *Proceedings of the 16th International Conference on Pattern Recognition (ICPR'02)*, Quebec, Canada, 2: 232-235, August 2002.

[FXK03] P. Fränti, M. Xu and I. Kärkkäinen, Classification of binary vectors by using deltaSC distance to minimize stochastic complexity, *Pattern Recognition Letters*, 24(1-3): 65-73, January 2003.

[G89] D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, 1989.

[GG92] A. Gersho and R.M. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic Publishers, Dordrecht, 1992.

[GG98] V. Grande and O. Günther, Multidimensional access methods, *ACM Computer Surveys*, 20(2): 170-231, June 1998.

[GJW82] M.R. Garey, D.S. Johnson and H.S. Witsenhausen, The complexity of the generalized Lloyd-Max problem, *IEEE Transactions on Information Theory*, 28(2): 255-256, March 1982.

[GK92] L. Guan and M. Kamel, Equal-average hyperplane partitioning method for vector quantization of image data, *Pattern Recognition Letters*, 13(10): 693-699, October 1992.

[GKP94] R.L. Graham, D.E. Knuth and O. Patashnik, *Concrete Mathematics – a Foundation for Computer Science* (2nd edition), Addison-Wesley, 257-267, 1994.

[GKV97] M. Gyllenberg, T. Koski and M. Verlaan, Classification of binary vectors by stochastic complexity, *Journal of Multivariate Analysis*, 63(1): 47-72, October 1997.

[GLC98] J. Gong, L. Li and W. Chen, Fast recursive algorithms for two-dimensional thresholding, *Pattern Recognition*, 31(3): 295-300, March 1998.

[GM86] V.D. Gesù and M.C. Maccarone, Features selection and 'possibility theory', *Pattern Recognition*, 19(1): 63-72, 1986.

[GPF91] D.P. de Garrido, W.A. Pearlman and W.A. Finamore, Vector quantization of image pyramids with the ECPNN algorithm, *SPIE Proceedings of Visual Communications and Image Processing*, Boston, MA, USA, 1605: 221-232, November 1991.

[GPF95] D.P. de Garrido, W.A. Pearlman and W.A. Finamore, A clustering algorithm for entropy-constrained vector quantizer design with applications in coding image pyramids, *IEEE Transactions on Circuits and Systems for Video Technology*, 5(2): 83-95, April 1995.

[GR69] J.C. Gover and G.J.S. Ross, Minimum spanning trees and single linkage cluster analysis, *Applied Statistics*, 18(1): 54-64, 1969.

[GRS99] L. Gao, A.L. Rosenberg and R.K. Sitaraman, Optimal clustering of tree-sweep computations for high-latency parallel environments, *IEEE Transactions on Parallel and Distributed Systems*, 10(8): 813-824, August 1999.

[GYZ98] D. Greene, F. Yao and T. Zhang, A linear algorithm for optimal context clustering with application to bi-level image coding, *Proceedings of IEEE International Conference on Image Processing*, 1: 508-511, October 1998.

[H75] J.A. Hartigan, *Clustering Algorithms*, John Wiley & Sons, New York, USA, 1975.

[H82] P.S. Heckbert, Color image quantization for frame buffer display, *ACM Computer Graphics*, 16(3): 297-307, July 1982.

[HB97] T. Hofmann, and J.M. Buchmann, Pairwise data clustering by deterministic annealing, *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 19(1): 1-14, January 1997.

[HBSH92] C.-M. Huang, Q. Bi, G.S. Stiles and R.W. Harris, Fast full search equivalent encoding algorithms for image compression using vector quantization, *IEEE Transactions on Image Processing*, 1(3): 413-416, July 1992.

[HC01] K.-F. Hwang and C.-C. Chang, Improved nearest codeword search scheme using a tighter kick-out condition, *Optical Engineering*, 40(9): 1749-1751, September 2001.

[HK01] D. Harel and Y. Koren, Clustering spatial data using random walks (extended abstract), *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'01)*, San Francisco, California, USA, 281-286, August 2001.

[HLC91] C.-H. Hsieh, P.-C. Lu and J.-C. Chang, Fast codebook generation algorithm for vector quantization of images, *Pattern Recognition Letters*, 12(10): 605-609, October 1991.

[HPLSH01] H.-C. Huang, J.-S. Pan, Z.-M. Lu, S.-H. Sun and H.-M. Hang, Vector quantization based on genetic simulated annealing, *Signal Processing*, 81: 1513-1523, 2001.

[HS00] E. Hartuv and R. Shamir, A clustering algorithm based on graph connectivity, *Information Processing Letters*, 76(4-6): 175-181, December 2000.

[HS03a] G.R. Hjaltason and H. Samet, Index-driven similarity search in metric spaces, *ACM Transactions on Database Systems*, 28(4): 517-580, December 2003.

[HS03b] G.R. Hjaltason and H. Samet, Properties of embedding methods for similarity searching in metric spaces, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(5): 530-549, May 2003.

[I80] P.K. Ito, Robustness of ANOVA and MANOVA test procedures, in: P.R. Krishnaiah (ed), *Handbook of Statistics 1: Analysis of Variance*, North-Holland Publishing Company, 199-236, 1980.

[IA99] L.S. Iyer and J.E. Aronson, A parallel branch-and-bound method for clustering analysis, *Annals of Operations Research*, 90: 65-86, 1999.

[J78] R.A. Jarvis, Shared near neighbor maximal spanning trees for cluster analysis, *Proceedings of the 4th International Conference on Pattern Recognition* (*ICPR'78*), Kyoto, Japan, 308-131, 1978.

[JD88] A.K. Jain and R.C. Dubes, *Algorithms for Clustering Data*, Prentice-Hall, Englewoods Cliffs, NJ, 1988.

[JMF99] A.K. Jain, M.N. Murty and P.J. Flynn, Data clustering: A review, *ACM Computing Surveys*, 31(3): 264-323, September 1999.

[K88] T. Kohonen, *Self-Organization and Associative Memory*, Springer-Verlag, New York, 1988.

[K91] T. Kurita, An efficient agglomerative clustering algorithm using a heap, *Pattern Recognition*, 24(3): 205-209, March 1991.

[K95] T. Kohonen, *Self-Organizing Maps*, Springer-Verlag, Berlin, Germany, 1995.

[K98] S. Kundu, A solution to histogram-equalization and other related problems by shortest path methods, *Pattern Recognition*, 31(3): 231-234, March 1998.

[K99] T. Kaukoranta, *Iterative and Hierarchical Methods for Codebook Generation in Vector Quantization*, Ph.D. Thesis, TUCS Dissertations 22, University of Turku, Turku, Finland, December 1999.

[K01] E. Kandogan, Visualizing multi-dimensional clusters, trends, and outliers using star coordinates, *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'01)*, San Francisco, CA, USA, 107-116, August 2001.

[K04a] J. Kivijärvi, *Optimization Methods for Clustering*, Ph.D. Thesis, TUCS Dissertations 48, University of Turku, Turku, Finland, February 2004.

[K04b] P. Kopylov, *Processing and Compression of Raster Map Images*, Ph.D. Thesis, Computer Science Dissertations 8, University of Joensuu, Joensuu, Finland, October 2004.

[KF02a] I. Kärkkäinen and P. Fränti, Dynamic local search for clustering with unknown number of clusters, *Proceedings of the 16th International Conference on Pattern Recognition (ICPR'02)*, Quebec, Canada, 2: 240-243, August 2002.

[KF02b] I. Kärkkäinen and P. Fränti, Stepwise algorithm for finding unknown number of clusters, *Advanced Concepts for Intelligent Vision Systems (ACIVS'2002)*, Gent, Belgium, 136-143, September 2002.

[KFN96] T. Kaukoranta, P. Fränti and O. Nevalainen, Reallocation of GLA codevectors for evading local minima, *Electronics Letters*, 32(17): 1563-1564, August 1996.

[KFN98] T. Kaukoranta, P. Fränti and O. Nevalainen, Iterative split-and-merge algorithm for VQ codebook generation, *Optical Engineering*, 37(10): 2726-2732, October 1998.

[KFN99] T. Kaukoranta, P. Fränti and O. Nevalainen, Vector quantization by lazy pairwise nearest neighbor method, *Optical Engineering*, 38(11), 1862-1868, November 1999.

[KFN00] T. Kaukoranta, P. Fränti and O. Nevalainen, A fast exact GLA based on code vector activity detection, *IEEE Transactions on Image Processing*, 9(8): 1337-1342, August 2000.

[KFN03] J. Kivijärvi, P. Fränti and O. Nevalainen, Self-adaptive genetic algorithm for clustering, *Journal of Heuristics*, 9(2): 113-129, March 2003.

[KGV83] S. Kirkpatrick, C.D. Gelatt Jr. and M.P. Vecchi, Optimization by simulated annealing, *Science*, 220 (4598): 671-680, May 1983.

[KHK99] G. Karypis, E.-H. Han and V. Kumar, Chameleon: a hierarchical clustering algorithm using dynamic modeling, *IEEE Computer*, 32(8): 66-75, August 1999.

[KI86] J. Kittler and J. Illingworth, Minimum error thresholding, *Pattern Recognition*, 19(1): 41-47, 1986.

[KKZ94] I. Katsavounidis, C.-C.J. Kuo and Z. Zhang, A new initialization technique for generalized Lloyd iteration, *IEEE Signal Processing Letters*, 1(10): 144-146, October 1994.

[KLL04] D.-W. Kim, K.H. Lee and D. Lee, On cluster vality index for estimation of the optimal number of fuzzy clusters, *Pattern Recognition*, 37(10): 2009-2025, October 2004.

[KM99] K. Krisna and M.N. Murty, Genetic *k*-means algorithm, *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics*, 29(3): 433-439, June 1999.

[KMNPSW02] T. Kanungo, D.M. Mount, N.S. Netanyahu, C.D. Piatko, R. Silverman and A.Y. Wu, An efficient *k*-means clustering algorithm: analysis and implementation, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7): 881-892, July 2002.

[KN86] J. Katajainen and O. Nevalainen, Computing relative neighbourhood graphs in the plane, *Pattern Recognition*, 19(3): 221-228, 1986.

[KNF75] W.L.G. Koontz, P.M. Narendra and K. Fukunaga, A branch and bound clustering algorithm, *IEEE Transactions on Computers*, 24(9): 908-915, September 1975.

[KNT87] J. Katajainen, O. Nevalainen and J. Teuhola, A linear expected-time algorithm for computing planar relative neighbourhood graphs, *Information Processing Letters*, 25(2): 77-86, May 1987.

[KOKKNN98] J. Kivijärvi, T. Ojala, T. Kaukoranta, A. Kuba, L. Nyúl and O. Nevalainen, A comparison of lossless compresion methods for medical images, *Computerized Medical Imaging and Graphics*, 22(4): 323-339, October 1998.

[KR79] R.L. Kirby and A. Rosenfeld, A note on the use of (gray level, local average gray level) space as an aid in thresholding selection, *IEEE Transactions on Systems, Man and Cybernetics*, 9(12): 860-864, 1979.

[KR90] L. Kaufman and P.J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*, John Wiley Sons, New York, 1990.

[KS98] F. Kossentini and M.J.T. Smith, A fast PNN design algorithm for entropy-constrained residual vector quantization, *IEEE Transactions on Image Processing*, 7(7): 1045-1050, July 1998.

[KSW85] J.N. Kapur, P.K. Sahoo and A.K.C. Wong, A new method for gray-level picture thresholding using the entropy of the histogram, *Computer Vision, Graphics, and Image Processing*, 29(3): 273-285, March 1985.

[L57] S.P. Lloyd, Least squares quantization in PCM, Memorandum, Bell Laboratories, Murray Hill, USA, 1957; published in *IEEE Transactions on Information Theory*, 28(2): 129-137, March 1982.

[LBG80] Y. Linde, A. Buzo and R.M. Gray, An algorithm for vector quantizer design, *IEEE Transactions on Communications*, 28(1): 84-95, January 1980.

[LC94] C.-H. Lee and L.-H. Chen, Fast closest codeword search algorithm for vector quantization, *IEE Proceedings – Vision, Image and Signal Processing*, 141(39): 143-148, June 1994.

[LC95a] C.-H. Lee and L.-H. Chen, A fast search algorithm for vector quantization using means and variances of codewords, *SPIE Proceedings of Visual Communications and Image Processing '95*, Taipei, Taiwan, 2501: 619-628, May 1995.

[LC95b] C.-H. Lee and L.-H. Chen, A fast search algorithm for vector quantization using mean pyramids of codewords, *IEEE Transactions on Communications*, 43(2/3/4): 1697-1702, February/March/April 1995.

[LC95c] C.-H. Lee and L.-H. Chen, High-speed closest codeword search algorithms for vector quantization, *Signal Processing*, 43(3): 323-331, May 1995.

[LCC01] P.-S. Liao, T.-S. Chen and P.-C. Chung, A fast algorithm for multilevel thresholding, *Journal of Information Science and Engineering*, 17: 713-727, 2001.

[LS55] J. Lukaszewicz and H. Steinhaus, On measuring by comparison, *Zastosowania Matematyene*, 2: 225-232, 1955.

[LS95] W. Li and E. Salari, A fast vector quantization encoding method for image compression, *IEEE Transactions on Circuits and Systems for Video Technology*, 5(2): 119-123, April 1995.

[LSP90] S.U. Lee, S.Y. Shung and R.H. Park, A comparative performance study of several global thresholding techniques for segmentation, *Computer Vision, Graphics, and Image Processing*, 52(2): 171-190, November 1990.

[LVV03] A. Likas, N. Vlassis and J.J. Verbeek, The global *k*-means clustering algorithm, *Pattern Recognition*, 36(2): 451-461, February 2003.

[LW67] G.N. Lance and W.T. Williams, A general theory of classificatory sorting strategies 1. Hierarchical systems, *The Computer Journal*, 9(4): 373-380, February 1967.

[M67] J.B. McQueen, Some methods of classification and analysis of multivariate observations, *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, Berkeley, USA, 1: 281-297, 1967.

[M95] H.A. Monaver, Image vector quantisation using a modified LBG algorithm with approximated centroids, *Electronics Letters*, 31(3): 174-175, February 1995.

[M02] J. Mielikäinen, A novel full-search vector quantization algorithm based on law of cosines, *IEEE Signal Processing Letters*, 9(6): 175-176, June 2002.

[M03] J. Mielikäinen, *Lossless Compression of Hyperspectral Images*, Ph.D. Thesis, Acta Universitatis Lappeenrantaensis 167, Lappeenranta University of Technology, Lappeenranta, Finland, December 2003.

[MC96] C.A. Murthy and N. Chowdhury, In search of optimal clusters using genetic algorithms, *Pattern Recognition Letters*, 17(8): 825-832, July1996.

[MNS91] N. Moayeri, D.L. Neuhoff and W.E. Stark, Fine-coarce vector quantization, *IEEE Transaction on Signal Processing*, 39(7): 1503-1515, July 1991.

[MOV94] M.L. Mico, J. Oncina and E. Vidal, A new version of the nearest-neighbor approximating and eliminating search algorithm (AESA) with linear preprocessing time and memory requirements, *Pattern Recognition Letters*, 15(1): 9-17, January 1994.

[MP02] R.R. Mettu and C.G. Plaxton, Optimal time bounds for approximate clustering, *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence*, Alberta, Canada, 344-351, August 2002.

[MS83] R.E. Michalski and R.E. Stepp, Automated construction of classifications: conceptual clustering versus numerical taxonomy, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(4): 396-410, September 1983.

[MS03] D.S. Modha and W.S. Spangler, Feature weighting in *k*-means clustering, *Machine Learning*, 52(3): 217-237, September 2003.

[MTTV97] G.L. Miller, S.-H. Teng, W. Thurston and S.A. Vavaris, Separators for sphere-packings and nearest neighbor graphs, *Journal of the ACM*, 44(1): 1-29, January 1997.

[NF88] N.M. Nasrabadi and Y. Feng, Vector quantization of image based upon the Kohonen self-organizing feature maps, *Proceedings of IEEE International Conference on Neural Networks* (*ICNN'88*), San Diego, CA, USA, 1: 101-108, July 1988.

[NN97] S.A. Nene and S.K. Nayar, A simple algorithm for nearest neighbor search in high dimensions, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(9): 989-1003, September 1997.

[NT92] A. Nyeck and A. Tosser-Roussey, Maximum entropy initialisation technique for image coding vector quantiser design, *Electronics Letters*, 28(3): 273-274, January 1992.

[O79] N. Otsu, A thresholding selection method from gray-level histograms, *IEEE Transactions on Systems, Man and Cybernetics*, 9(1): 62-66, January 1979.

[O91] M.T. Orchard, A fast nearest-neighbor search algorithm, *Proceedings of International Conference on Acoustics, Speech, and Signal Processing* (*ICASSP'91*), 4: 2297-2300, April 1991.

[OB91] M.T. Orchard and C.A. Bouman, Color quantization of images, *IEEE Transactions on Signal Processing*, 39(12): 2677-2690, December 1991.

[OM95] G.C. Osbourn and R.F. Martinez, Empirically defined regions of influence for cluster analyses, *Pattern Recognition*, 28(11): 1793-1806, November 1995.

[OZ00] S.H. Ong and X. Zhao, On post-clustering evaluation and modification, *Pattern Recognition Letters*, 21(5): 365-373, May 2000.

[P80] T. Pun, A new method for gray-level picture thresholding using the entropy of the histogram, *Signal Processing*, 2(3): 223-237, July 1980.

[P97] G. Palubeckis, A branch-and-bound approach using polyhedral results for a clustering problem, *INFORMS Journal of Computing*, 9(1): 30-42, Winter 1997.

[PLL99] J.M Peña, J.A. Lozano and P. Larrañaga, An empirical comparision of four initialization methods for the *k*-means algorithm, *Pattern Recognition Letters*, 20(10): 1027-1040, October 1999.

[PMJ95] J.S. Pan, F.R. McInnes and M.A. Jack, VQ codebook design using genetic algorithms, *Electronic Letters*, 31(17): 1418-1419, August 1995.

[PP89] N.R. Pal and S.K. Pal, Entropic thresholding, *Signal Processing*, 16(2): 97-108, February 1989.

[PP93] N.R. Pal and S.K. Pal, A review on image segmentation techniques, *Pattern Recognition*, 26(9): 1277-1294, September 1993.

[PS85] F.P. Preparata and M.I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, 1985.

[R78] J. Rissanen, Modeling by shortest data description, *Automatica*, 14: 465-471, 1978.

[RGF90] K. Rose, E. Gurewitz and G.C. Fox, A deterministic annealing approach to clustering, *Pattern Recognition Letters*, 11(9): 589-594, September 1990.

[RK93] S.-W. Ra and J.-K. Kim, A fast mean-distance-ordered partial codebook search algorithm for image vector quantization, *IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing*, 40(9): 576-579, September 1993.

[RP92] V. Ramasubramanian and K.K. Paliwal, Fast *k*-dimensional tree algorithms for nearest neighbor search with application to vector quantization encoding, *IEEE Transactions on Signal Processing*, 40(3): 518-531, March 1992.

[RP97] V. Ramasubramanian and K.K. Paliwal, Fast vector quantization encoding based on *k*-d tree backtracking search algorithm, *Digital Signal Processing*, 7(3): 163-187, July 1997.

[RP00] V. Ramasubramanian and K.K. Paliwal, Fast nearest-neighbor search algorithms based on approximation-elimination search, *Pattern Recognition*, 33(9): 1497-1510, September 2000.

[RRS00] S. Ramaswamy, R. Rastogi and K. Shim, Efficient algorithms for mining outliers from large data sets, *ACM SIGMOD Record*, 29(2): 427-438, June 2000.

[S75] M.I. Shamos, Geometric complexity, *Proceedings of the 7th Annual ACM Symposium on the Theory of Computing*, Albuquerque, New Mexico, USA, 224-233, 1975.

[S78] G. Schwarz, Estimating the dimension of a model, *The Annals of Statistics*, 6: 461-464, 1978.

[S80] H. Späth, *Cluster Analysis Algorithms for Data Reduction and Classification of Objects*, Ellis Horwood Limited, West Sussex, UK, 1980.

[S90a] H. Samet, Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS, Addidon-Wesley, Readings, MA, 1990.

[S90b] H. Samet, The Design and Analysis of Spatial Data Structures, Addison-Wesley, Readings, MA, 1990.

[S91] R. Sproull, Refinements to nearest-neighbor searching in *k*-dimensional trees, *Algorithmica*, 6(4): 579-589, 1991.

[S96] P. Scheunders, A genetic Lloyd-Max quantization algorithm, *Pattern Recognition Letters*, 17(5): 547-556, May 1996.

[S00] M. Smid, Closest-Point Problems in Computational Geometry, *In Handbook on Computational Geometry* (editors J.-R. Sack and J. Urrutia), Elsevier Science, Amsterdam, North Holland, 2000.

[SC91a] T.-H. Su and R.-C. Chang, Computing the constrained relative neighborhood graphs and constrained gabriel graphs in Euclidean plane, *Pattern Recognition*, 24(3): 221-230, 1991.

[SC91b] T.-H. Su and R.-C. Chang, Computing the *k*-relative neighborhood graphs in Euclidean plane, *Pattern Recognition*, 24(3): 231-239, 1991.

[SC98] D.-F. Shen and K.-S. Chang, Fast PNN algorithm for design of VQ initial codebook, *Proceedings of SPIE Visual Communications and Image Processing '98*, San Jose, California, USA, 3309: 842-850, January 1998.

[SO97] J. Shanbehzadeh and P.O. Ogunbona, On the computational complexity of the LGB and PNN algorithms, *IEEE Transactions on Image Processing*, 6(4): 614-616, April 1997.

[SR03] L.K. Saul and S.T. Roweis, Think globally, fit locally: unsupervised learning of low dimensional manifolds, *Journal of Machine Learning Research*, 4: 119-155, June 2003.

[SS73] P.H.A. Sneath and R.R. Sokal, *Numerical Taxonomy*, W.H. Freeman & Co, San Francisco, 1973.

[SS01] L.G. Shapiro and G.C. Stockman, *Computer Vision*, Prentice Hall, New Jersey, 83-91, 2001.

[SS04] M. Sezgin and B. Sankur, Survey over image thresholding techniques and quatitative performance evaluation, *Journal of Electronic Imaging*, 13(1): 146-165, January 2004.

[SSWC88] P.K. Sahoo, S. Soltani, A.K.C. Wang and Y.C. Chen, A survey of thresholding techniques, *Computer Vision, Graphics, and Image Processing*, 41(2): 233-260, February 1988.

[ST00] M. Sezgin and R. Taşaltín, A new dichotomization technique to multilevel thresholding devoted to inspection applications, *Pattern Recognition Letters*, 21(2): 151-161, February 2000.

[SWJ04] H. Sun, S. Wang and Q. Jiang, FCM-based model selection algorithms for determining the number of clusters, *Pattern Recognition*, 37(10): 2027-2037, October 2004.

[SYK97] M. Sarkar, B. Yegnanarayana and D. Khemani, A clustering algorithm using an evolutionary programming-based approach, *Pattern Recognition Letters*, 18(10): 975-986, October 1997.

[T80a] G.T. Toussaint, Pattern recognition and geometrical complexity, *Proceedings of the 5th International Conference on Pattern Recognition* (*ICPR'80*), Miami Beach, Florida, USA, 1324-1347, December 1980.

[T80b] G.T. Toussaint, The relative neighbourhood graph of a finite planar set, *Pattern Recognition*, 12(4): 261-268, 1980.

[T85] W.H. Tsai, Moment-preserving thresholding: a new approach, *Computer Vision, Graphics, and Image Processing*, 29(3): 377-393, March 1985.

[T99] L. Talavera, Feature selection as a preprocessing step for hierarchical clustering, *Proceedings of the 16th International Conference on Machine Learning (ICML'99)*, Bled, Slovenia, 389-397, June 1999.

[T00] L. Talavera, Dependency-based feature selection for clustering symbolic data, *Intelligent Data Analysis*, 4(1): 19-28, 2000.

[TC92] D.-M. Tsai and Y.-H. Chen, A fast histogram-clustering approach for multi-level thresholding, *Pattern Recognition Letters*, 13(4): 245-252, April 1992.

[TSL00] J.B. Tenenbaum, V. de Silva and J.C. Langford, A global geometric framework for nonlinear dimensionality reduction, *Science*, 290(5500): 2319-2323, December 2000.

[TY00] L.Y. Tseng and S.B. Yang, A genetic clustering algorithm for data with non-spherical-shape clusters, *Pattern Recognition*, 33(7): 1251-1259, July 2000.

[U82] R.B. Urquhart, Graph theoretical clustering besed on limited neighbourhood sets, *Pattern Recognition*, 15(3): 173-187, 1982.

[U83] R.B. Urquhart, Some properties of the planar Euclidean relative neighbourhood graph, *Pattern Recognition Letters*, 1(5-6): 317-322, July 1983.

[V86] E. Vidal, An algorithm for finding nearest neighbors in (approximately) constant average time, *Pattern Recognition Letters*, 4(3): 145-157, July 1986.

[V89] P.M. Vaidya, An O(*n* log *n*) algorithm for the all-nearest-neighbors problem, *Discrete & Computational Geometry*, 4: 101-115, 1989.

[V94] E. Vidal, New formulation and improvements of the nearest-neighbor approximating and eliminating search algorithm (AESA), *Pattern Recognition Letters*, 15(1): 1-7, January 1994.

[V95] J.M. Vilar, Reducing the overhead of the AESA metric-space nearest neighbor searching algorithm, *Information Processing Letters*, 56(5): 265-271, December 1995.

[VF04] O. Virmajoki and P. Fränti, Divide-and-conquer algorithm for creating neighborhood graph for clustering, *IAPR International Conference on Pattern Recognition (ICPR'04)*, Cambridge, United Kingdom, 1: 264-267, August 2004.

[VFK02] O. Virmajoki, P. Fränti and T. Kaukoranta, Iterative shrinking for generating clustering, *IEEE International Conference on Image Processing (ICIP'02)*, Rochester, New York, USA, 2: 685-688, September 2002.

[W63] J.H. Ward, Hierarchical grouping to optimize an objective function, *Journal of American Statistical Association*, 58(301): 236-244, March 1963.

[W91] X. Wu, Optimal quantization by matrix searching, *Journal of Algorithms,* 12(4): 663-673, December 1991.

[W04] E.W. Weisstein, Kissing number, From *MathWorld*—A Wolfram Web Resource, 2004. http://mathworld.wolfram.com/KissingNumber.html, 3.11.2004.

[WC04] S. Wu and T.W.S. Chow, Clustering of the self-organizing map using a clustering validity index based on inter-cluster and intra-cluster density, *Pattern Recognition*, 37(2): 175-188, February 2004.

[WCS01] C.-C. Wong, C.-C. Chen and M.-C. Su, A novel algorithm for data clustering, *Pattern Recognition*, 34(2): 425-442, February 2001.

[WG94] X. Wu and L. Guan, Acceleration of the LBG algorithm, *IEEE Transactions on Communications*, 42(2/3/4): 1518-1523, February/ March/April 1994.

[WH84] S. Wang and R.M. Haralick, Automatic multithresholding selection, *Computer Vision, Graphics, and Image Processing*, 25(1): 46-67, 1984.

[WL83] M.A. Wong and T. Lane, A *k*th nearest neighbor clustering procedure, *Journal of Royal Statistical Society*, 45(3): 362-368, 1983.

[WL93] Z. Wu and R. Leahy, An optimal graph theoretic approach to data clustering: theory and its application to image segmentation, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(11): 1101-1113, November 1993.

[WL00] K.-S. Wu and J.-C. Lin, Fast VQ encoding by an efficient kick-out condition, *IEEE Transactions on Circuit and Systems for Video Technology*, 10(1): 59-62, February 2000.

[WZ91] X. Wu and K. Zhang, A better tree-structured vector quantizer, *Proceedings of IEEE Data Compression Conference (DCC'91)*, Snowbird, USA, 392-401, April 1991.

[WZ93] X. Wu and K. Zhang, Quantizer monotonicities and globally optimal scalar quantizer design, *IEEE Transactions on Information Theory*, 39(3): 1049-1053, May 1993.

[X04] M. Xu, Delta-MSE dissimilarity in GLA based vector quantization, *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing* (*ICASSP'04*), Montreal, Canada, 5: 813-816, May 2004.

[Y93] P.N. Yianilos, Data structures and algorithms for nearest neighbor search in general metric spaces, *Proceedings of the 4th Annual ACM-SIAM*

Symposium on Discrete Algorithms (SODA'93), Austin, Texas, USA, 311-321, January 1993.

[Y99] P.-Y. Yin, A fast scheme for optimal thresholding using genetic algorithms, *Signal Processing*, 72(2): 85-95, 1999.

[Y00] P.N. Yianilos, Locally lifting the curse of dimensionality for nearest neighbor search (extended abstract), *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'00)*, San Francisco, California, USA, 361-370, January 2000.

[Y02] P.-Y. Yin, Maximum entropy-based optimal threshold selection using deterministic reinforcement learning with controlled randomization, *Signal Processing*, 82(7): 993-1006, July 2002.

[YC97] P.-Y. Yin and L.-H. Chen, A fast iterative scheme for multilevel thresholding methods, *Signal Processing*, 60(3): 305-313, 1997.

[YCC95] J.-C. Yen, F.-J. Chang and S. Chang, A new criterion for automatic multilevel thresholding, *IEEE Transactions on Image Processing*, 4(3): 370-378, March 1995.

[YG88] G. Yuan and M. Goldberg, A sequential initialization technique for vector quantizer design, *Pattern Recognition Letters*, 7(3): 157-161, March 1988.

[Z71] C.T. Zahn, Graph-theoretical methods for detecting and describing gestalt clusters, *IEEE Transactions on Computers*, C-20(1): 68-86, January 1971.

[Z01] B. Zhang, Generalized *k*-harmonic means – dynamic weighting of data in unsupervised learning, *Proceedings of the 1st SIAM International Conference on Data Mining*, Chicago, IL, USA, April 2001.

[ZG89] K. Zeger and A. Gersho, Stochastic relaxation algorithm for improved vector quantiser design, *Electronics Letters*, 25(14): 896-898, July 1989.

[ZRL97] T. Zhang, R. Ramakrishnan and M. Livny, BIRCH: a new data clustering algorithm and its applications, *Data Mining and Knowledge Discovery*, 1(2): 141-182, June 1997.