UNIVERSITY OF JOENSUU
DEPARTMENT OF COMPUTER SCIENCE
Report Series A

Randomized local search algorithm for the clustering problem

Pasi Fränti[1] and Juha Kivijärvi[2]

Report A-1999-5
(revised 24.4.2000)

ACM I.4.2, I.5.3
UDK 519.712
ISSN 0789-7316
ISBN 951-708-845-0

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

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

Randomized local search algorithm for the clustering problem

(to appear in *Pattern Analysis and Applications*)

Pasi Fränti[1] and Juha Kivijärvi[2]

[1] *Department of Computer Science*
University of Joensuu
Box 111, FIN-80101 Joensuu
FINLAND
Email: franti@cs.joensuu.fi

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

Abstract: We consider clustering as a combinatorial optimization problem. *Local search* provides a simple and effective approach to many other combinatorial optimization problems. It is therefore surprising how seldom it has been applied to the clustering problem. Instead, the best clustering results have been obtained by more complex techniques such as *tabu search* and *genetic algorithms* at the cost of high run time. We introduce a new randomized local search algorithm for the clustering problem. The algorithm is easy to implement, sufficiently fast, and competitive with the best clustering methods. The ease of implementation makes it possible to tailor the algorithm for various clustering applications with different distance metrics and evaluation criteria.

Keywords: clustering, vector quantization, local search, combinatorial optimization, image processing, compression.

1. Introduction

Clustering, or *unsupervised classification*, is considered here as a *combinatorial optimization problem* where the aim is to partition a set of data objects into a predefined number of clusters. The objects with similar features should be grouped together and objects with different features to separate groups [1, 2]. Clustering has many applications in social sciences, numerical taxonomy, computer science and image processing. The size and dimensionality of the data is often very high, which makes manual processing practically impossible. A high quality computer based clustering is therefore needed.

The general clustering problem includes three subproblems: (i) selection of the evaluation function, (ii) decision of the number of groups in the clustering, and (iii) the choice of the clustering algorithm. We consider the last subproblem and assume that the number of clusters (groups) is fixed beforehand. The evaluation function depends on the application and the type of data objects. Minimization of intracluster diversity is widely used as a criterion and it is therefore applied here as well.

There are several established methods for generating a clustering [1, 2, 3]. The most cited and widely used method is the *k-means algorithm* [4]. It starts with an initial solution, which is iteratively improved using two optimality criteria in turn until a local

minimum is reached. The algorithm is easy to implement and it gives reasonable results in most cases. Unfortunately the algorithm makes only local changes to the original clustering and it gets stuck at the first local minimum. The quality of the final clustering is therefore highly dependent on the initialization.

Better results have been achieved by optimization methods such as *genetic algorithms* (*GA*) and *tabu search* (*TS*). A common property of these methods is that they consider several possible solutions at time. The *GA* and *TS* approaches give better clustering results, and they are less dependent on the initialization [5, 6]. A drawback of these approaches is their high run time. There are many candidate solutions, and each of them must be fine-tuned by the *k-means* algorithm. This makes the overall run time significantly higher than that of the *k-means*, and the methods are also more complex to implement.

We propose a new clustering algorithm based on the traditional optimization technique, *local search*. We show that *randomized local search* (*RLS*) gives competitive results to those of the *GA* and *TS* with a faster algorithm, and with considerably simpler implementation. The method is based on the ideas studied in [6] with the following enhancements and simplifications. The representation of solution and the neighborhood function are revised so that new candidate solutions can be generated more efficiently. The use of the time consuming *k-means* can be implemented much faster. The *tabu list* is also omitted. These modifications give remarkable speed-up to the algorithm without losing the quality of the clustering. The method offers therefore a good trade-off between the quality and the complexity of the algorithm.

The simplicity of the proposed algorithm makes it a suitable candidate for a wide variety of clustering applications. The main modifications in the solution are based on combinatorial changes in the clustering structure. This is independent on the chosen distance metric. The most critical part of the algorithm when changing the distance metric is the partition step, in which the data objects are classified to the existing clusters. The classification rule is an integral part of practically all clustering methods (directly or indirectly), and must therefore be modified according to the chosen distance metric and evaluation criterion. Besides that, the algorithm is expected to be independent on the changes in the application domain.

The rest of the paper is organized as follows. We start in Section 2 by giving a formal definition of the clustering problem, and by summarizing existing clustering algorithms in Section 3. The local search approach is then studied in Section 4. We consider the local search first in a wider context by defining the basic components of the algorithm. Several different design alternatives are discussed for each component. The randomized local search and its parameter setup are then introduced in Section 5. Test data sets are presented in Section 6, and the performance of the new algorithm is compared with the related clustering methods in Section 7. Conclusions are drawn in Section 8.

2. Clustering problem

We use the following notations:

N	Number of data objects.
M	Number of clusters.
K	Number of attributes.
X	Set of N data objects $X=\{x_1, x_2, \ldots, x_N\}$.
P	Set of N cluster indices $P=\{p_1, p_2, \ldots, p_N\}$.
C	Set of M cluster representatives $C=\{c_1, c_2, \ldots, c_M\}$.

The clustering problem is defined as follows. Given a set of N data objects (x_i), partition the data set into M clusters such that similar objects are grouped together and objects with different features belong to different groups. Partition (P) defines the clustering by giving for each data object the cluster index (p_i) of the group where it is assigned to. Each group is described by its representative data object (c_i).

***Object dissimilarity*:** Each object x_i has K attributes (x_i^k), which together form a *feature vector*. For simplicity, we assume that the attributes are numerical and have the same scale. If this is not the case the attributes must first be normalized. The dissimilarity between two objects x_1 and x_2 is measured by a distance function $d(x_1, x_2)$ between the two feature vectors.

***Evaluation of clustering*:** The most important choice in the clustering method is the objective function f for evaluating the clustering. The choice of the function depends on the application and there is no universal solution of which measure should be used. However, once the objective function is decided the clustering problem can be formulated as a combinatorial optimization problem. The task is to find such a partition P that minimizes f.

A commonly used objective function is the *sum of squared distances* of the data objects to their cluster representatives. Given a partition P and the cluster representatives C, it is calculated as:

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

where d is a distance function. *Euclidean distance* is the most widely used distance function in the clustering context, and it is calculated as:

$$d\left(x_1, x_2\right) = \sqrt{\sum_{i=1}^{K} \left(x_1^i - x_2^i\right)^2} \tag{2}$$

Thereafter, given a partition P, the optimal choice for the cluster representatives C minimizing (1) are the cluster *centroids* calculated as:

$$c_j = \frac{\sum_{p_i=j} x_i}{\sum_{p_i=j} 1}, \quad 1 \le j \le M \tag{3}$$

The objective function (1) is implicitly applied in the *k-means* method. Hierarchical clustering methods, however, do not usually define the objective function directly. For example, agglomerative clustering methods measure the distance between the clusters, and use this criterion for selecting the clusters to be merged [7]. The most popular method, *Ward's method*, minimizes also (1) in each step of the algorithm [8]. For these reasons, we assume the sum of squared distances as the objective function in the rest of this paper.

***Number of clusters*:** In applications such as *vector quantization* [3], the number of clusters (M) is merely a question of resource allocation. It is therefore a parameter of the clustering algorithm. In some other applications the number of clusters must also be solved. The choice of the correct number of clusters is an important subproblem of clustering and should be considered separately. The decision is typically made by the researcher of the application area but analytical methods also exist [9, 10] for helping in the decision. A common approach is to generate several clusterings for various values of M and compare them against some evaluation criteria. In the rest of this paper we assume that the number of clusters is fixed.

3. Clustering algorithms

The clustering problem in its combinatorial form has been shown to be NP-complete [11]. No polynomial time algorithm is known to find the globally optimal solution, but reasonable suboptimal solutions are typically obtained by heuristic algorithms. Next we recall four groups of clustering algorithms.

K-means [4]: The algorithm (see Fig. 1) starts with an initial solution, which is iteratively improved until a local minimum is reached. In the first step, the data objects are partitioned into a set of M clusters by mapping each object to the nearest cluster centroid of the previous iteration. In the second step, the cluster centroids are recalculated corresponding to the new partition. The quality of the new solution is always better than or equal to the previous one. The algorithm is iterated as long as improvement is achieved. The number of iterations depends on the data set, and on the quality of the initial solution. Ten to fifty iterations are usually needed when starting from a random clustering.

```
K-means(X,P,C): returns (P,C)
 REPEAT
     FOR i:=1 TO N DO
         P_i ←FindNearestCentroid(x_i,C)
     FOR i:=1 TO M DO
         C_i ←CalculateCentroid(X,P,i)
 UNTIL no improvement.
```

Figure 1. Structure of the *k-means* algorithm.

The *k-means* algorithm is easy to implement and it gives reasonable results in most cases. However, the method itself is only a descent method and it gets stuck at the first local minimum. The quality of the final clustering is therefore highly sensitive to the

initialization and there is high variation between the results of different initializations. An advantage of the *k-means* method is that it can improve almost any existing solution not obtained by a *k-means* based method. The method is therefore highly applicable when integrated with other clustering methods as a local optimizer; see [5, 6, 12]. In the vector quantization context the method is known as *Generalized Lloyd algorithm* (GLA), or the *LBG* method due to [13].

***Agglomerative clustering*:** Another approach is to generate the clustering hierarchically. *Agglomerative methods* start by initializing each data object as a separate cluster. Two clusters are merged at each step of the algorithm and the process is repeated until the desired number of clusters is obtained. The clusters to be merged are always the ones that are closest to each other among all possible cluster pairs. Agglomerative clustering provides also a taxonomy (*dendrogram*) of the groups as a by-product.

There are several ways to measure the "distance" of two clusters: in a *single linkage method* (*nearest neighbor*) it is the minimum distance of the individual data objects belonging to the different groups, and in the *complete linkage method* (*furthest neighbor*) it is the maximum of these distances. *Ward's method* [8] selects the cluster pair that increases the objective function value least. This is a natural way for minimizing (1). In the vector quantization context, this method is known as *pairwise nearest neighbor* (*PNN*) due to [14].

The *Ward's method* is simple to implement and it usually outperforms the *k-means* and most of the other hierarchical clustering methods in the minimization of (1). The biggest deficiency of the algorithm is its slowness. It has generally been considered as an $O(N^3)$ time algorithm, which is a severe restriction in the case of large data sets. In a recent study, however, it was shown that the method can be implemented by an $O(\tau N^2)$ time algorithm where τ is significantly smaller than N in practice [15].

***Divisive clustering*:** Divisive clustering uses an opposite, top-down approach for generating the clustering. The method starts with a single cluster including all the data objects. New clusters are then created one at a time by dividing existing clusters. The splitting process is repeated until the desired number of clusters is reached. The divisive approach usually requires much less computation than the bottom-up approach of agglomeration, and it also provides taxonomy of the groups.

Despite its benefits the divisive clustering is far less used than the agglomerative clustering because of two main problems. The first problem is that there is no easy solution for defining the way the cluster should be split. The second problem is that partitions made in the earlier stages cannot be changed later. Inaccurate divisions can therefore be very harmful for the quality of the final clustering. Fortunately, these problems have been recently studied in the vector quantization literature and good solutions exist for both problems [16, 17].

The best known approach for the splitting is to use *principal component analysis* (PCA) [18]. The main idea is to calculate the principal axis of the data vectors in the cluster. The data objects are then classified by a (K-1)-dimensional hyperplane perpendicular to the principal axis. The optimal location for the hyperplane is obtained by considering each object as a tentative dividing point through which the hyperplane passes. This

technique can be implemented using an efficient $O(N\cdot \log N\cdot \log M) + O(N\cdot K^2\cdot \log M)$ time algorithm, see [17] for details.

The second problem can be solved by fine-tuning the partition boundaries after each split operation. The new (smaller) clusters due to the division may attract data objects from neighboring clusters. Wrong decisions in the early divisions can therefore be corrected by repartition. PCA-based divisive method with this kind of partition refinement gives comparable results to that of the *Ward's method* in the minimization of (1), with a faster $O(NM)$ algorithm [17]. The problem of the method is that the best results are obtained with a rather complicated algorithm, in comparison to the *k-means* and the agglomerative methods.

***Optimization methods*:** Various optimization methods, such as *local search*, *tabu search*, *stochastic relaxation*, *neural networks* and *genetic algorithms* have also been applied to the clustering problem [5, 6, 19, 20, 21]. A common property of these methods is that they consider several possible solutions and generate a new solution (or a set of solutions) at each step on the basis of the current one [22, 23]. The use of several candidates directs the search towards the highest improvement in the optimization function value. The key question is how to create new candidate solutions.

Tabu search (*TS*) is a variant of the traditional local search, which uses suboptimal moves to allow the search to continue past local minima. A tabu list is used to prevent the search from returning to solutions that have been visited recently. This forces the search into new directions instead of stucking in a local minimum and its neighborhood. The algorithm in [6] generates new solutions by making random modifications to the current solution and fine-tuning it by two iterations of the *k-means*. This was shown to give high quality clustering results at the cost of high run time.

Genetic algorithms (*GA*) maintain a set of solutions (*population*). In each iteration the algorithm generates new solutions by genetic operations, such as *crossover* and *mutation*. Only the best solutions survive to the next iteration (*generation*). New candidates are created in the crossover by combining two existing solutions (*parents*). A simple approach uses random crossover and the *k-means* as a local optimizer but this can hardly perform any better than a simple local search with similar modifications. However, the *GA* has been shown to outperform all existing methods [5] when a well-defined deterministic crossover operation is applied. The drawback of the *GA* is that a large number of candidate solutions must be generated. This makes the overall run time significantly higher than that of the k-means and the hierarchical methods.

4. Design alternatives for local search

The structure of a local search algorithm is shown in Fig. 2. The algorithm starts with an initial solution, which is iteratively improved using neighborhood search and selection. In each iteration a set of *candidate solutions* is generated by making small modifications to the existing solution. The best candidate is then chosen as the new solution. The search is iterated a fixed number of iterations, or until a stopping criterion is met.

In a local search algorithm, the following design problems are to be considered:

- Representation of a solution
- Neighborhood function
- Search strategy

The representation of a solution is an important choice in the algorithm. It determines the data structures which are to be modified. The neighborhood function defines the way the new solutions are generated. An application of the neighborhood function is referred as a *move* in the neighborhood search. The neighborhood size is usually very large and only a small subset of all possible neighbors are generated. The search strategy determines the way the next solution is chosen among the candidates. The most obvious approach is to select the solution minimizing the objective function (1).

```
Generate initial solution.
REPEAT
      Generate a set of new solutions.
      Evaluate the new solutions.
      Select the best solution.
UNTIL stopping criterion met.
```

Figure 2. Structure of the local search.

4.1 Representing a solution

A solution could be represented and processed as a bit string (binary data) without any semantic interpretation of the content. New solutions would then be generated by turning a number of randomly chosen bits in the solution. However, this is not a very efficient way to improve the solution and it is possible that certain bit combinations do not represent a valid solution. It is therefore much more efficient to use a problem specific representation and operate directly on the data structures in the problem domain.

In the clustering problem there are two main data structures: the partition P of the data objects, and the cluster representatives C. In the context of minimizing (1) using Euclidean distance, the P and C depend on each other so that if one of them has been given, the optimal choice of the other one can be uniquely constructed. This is formalized in the following two optimality conditions [3]:

- *Nearest neighbor condition*: For a given set of cluster centroids, any data object can be optimally classified by assigning it to the cluster whose centroid is closest to the data object in respect to the distance function.
- *Centroid condition*: For a given partition, the optimal cluster representative minimizing the distortion is the *centroid* of the cluster members.

It is therefore sufficient to determine only P or C to define a solution although both of them must be generated in order to evaluate the solution. The preceding reasoning gives three alternative approaches for representing a solution:

- Partition: (P)
- Centroid: (C)
- Combined: (P, C)

The first approach operates with P and generates C (when needed) using the centroid condition. This is a natural representation in traditional clustering problems since the primary aim is to generate the partition. The approach is computationally fast requiring only $O(N)$ time. The problem is that only local changes may be generated to the solution by modifying the partitions.

The second approach operates with C and generates the partition using the nearest neighbor condition. This is a natural way to represent the solution in *vector quantization* applications since the aim is to create a *codebook* (corresponding to the set of cluster representatives). This approach is effective because the entire clustering structure may be revised through modifications of the cluster representatives. A drawback is that the generation of the partition is computationally expensive requiring $O(NM)$ time.

We take the third approach and maintain both P and C. The key point is that both data structures are needed for evaluating the clustering, and it would be computationally inefficient to recalculate either data structure from scratch in every step of the algorithm. Instead, the data structures of the existing solutions are utilized. We aim at achieving the power of the second representation (having only C) but avoiding its slowness.

4.2 Neighborhood function

The neighborhood function generates new candidate solutions by making modifications to the current solution. The amount of modifications must be small enough not to destroy the original solution completely, but also large enough so that the search may pass local minima. Some level of randomness should be included in the neighborhood function but it is unlikely that random modifications alone improve the clustering. A good neighborhood function is balanced between random and deterministic modifications.

The modifications should make global changes to the clustering structure and at the same time perform local fine tuning (as in the *k-means*) towards a local optimum. Global changes can be made by changing the location of the clusters through modifying the cluster representatives. These changes are needed for obtaining significantly different solutions from the current one. The purpose of local changes is to fine-tune the current solution towards a local optimum so that it is competitive with the existing

solution. In the following, we describe different ways to generate global and local changes.

4.2.1 Methods for global rearrangement

Random swap: A randomly chosen cluster is made obsolete and a new one is created. This is performed by replacing the chosen cluster representative c_j by a randomly chosen data object x_i:

$$c_j \leftarrow x_i \quad \big| \quad j = random(1, M), \;\; i = random(1, N) \tag{4}$$

The change is effective if the step is followed by partition refinement. The idea is due to [6] with the difference that a single swap is enough. Further changes in a single step may disturb the current clustering too much to be able to improve the solution. A single swap requires $O(1)$ time on average, and at most $O(M)$ time if duplicates are not allowed when selecting the new representative. We will use the random swap as the basic component in our neighborhood function.

Deterministic swap: A deterministic variant of the swapping method is proposed in [24]. The method removes the cluster whose absence decreases the quality of the clustering least. A new cluster is added in the vicinity of the cluster that has the largest distortion. The swap is performed as in the previous method by changing the cluster representatives. Partition refinement is thus needed after the swap. The method is computationally expensive requiring $O(NM)$ time due to the calculation of secondary partition for each data object.

Split-and-merge: The split-and-merge approaches in [12, 25] are similar to the preceding swapping methods but they operate with the partitions. At each step, two nearby clusters are merged and reallocated elsewhere by splitting another cluster. The clusters to be merged are optimally chosen. The split can be performed heuristically as in [25], or using principal component analysis as in [12]. The methods include the partition update implicitly. No explicit refinement is therefore needed. A single split-and-merge phase can be performed in $O(M^2)$ time [12] on average.

4.2.2 Methods for local refinement

Optimal representatives: (due to the *k-means*) At any stage of the algorithm, the existing solution can be improved by recalculating the cluster representatives according to the centroid condition:

$$c_j \leftarrow \frac{\sum_{p_i = j} x_i}{\sum_{p_i = j} 1} \quad \forall \quad j \in [1, M] \tag{5}$$

This is a rather trivial operation and it takes only $O(M)$ time.

Optimal partition: (due to the *k-means*) At any stage of the algorithm, the existing partition may be improved by regenerating the partition according to the nearest neighbor condition:

$$p_i \leftarrow \arg\min_{1 \le j \le M} d\left(x_i, c_j\right)^2 \quad \forall \; i \in [1, N] \tag{6}$$

The partition optimality is important for achieving competitive solutions but it is also computationally expensive taking $O(NM)$ time.

Object rejection: The idea is to perform optimal partition considering only a single object at a time. We take any cluster j and find the optimal partition for the data objects in this cluster:

$$p_i \leftarrow \arg\min_{1 \le k \le M} d\left(x_i, c_k\right)^2 \quad \forall \; i \mid p_i = j \tag{7}$$

Effectively, the chosen cluster may "reject" the objects whose partition was not optimally chosen. The step requires $O(N)$ time on average since there are M distance calculations, and N/M objects to be checked on average.

Object attraction: The idea is similar to the object rejection. We take any cluster j and check the distance of every data object x_i to the centroid of their current cluster (p_i), and the distance to the chosen cluster (j). If we find out that a data object is closer to the chosen centroid then its partition is changed:

$$p_i \leftarrow \arg\min_{k = j \vee k = p_i} d\left(x_i, c_k\right)^2 \quad \forall \; i \in [1, N] \tag{8}$$

Effectively, the chosen cluster may "attract" new objects from its neighboring clusters. The step requires $O(N)$ time since there are only two distance calculations for each of the N objects.

Local repartition: Local repartition is a combination of the preceding object rejection (7) and object attraction (8) rules. It can be used as a partition refinement after the swapping methods. In the object rejection, the target cluster is chosen as the obsolete cluster that was removed in the swap. In the object attraction, the target cluster is chosen as the new cluster that was added in the swap. The motivation is that the original partition is a valid starting point for the new partition and unnecessary computations should not be wasted for full repartition. Furthermore, if the original partition was optimal (in respect to the previous C) the new partition is also optimal (in respect to the modified C). In this case, the local repartition performs optimal partition. The requirement of the optimality itself, however, is not necessary for the *LS* algorithm. The step requires $O(N)$ time in total.

Partition swapping: The idea of the random swapping in Section 4.2.1 can also be applied to partition refinement. Any data object i may be moved to another randomly chosen cluster:

$$p_i \leftarrow random(1, M) \tag{9}$$

This kind of blind swapping, however, is not effective unless the new cluster is a neighboring cluster. Even so, the partition swapping performs only local changes and it is therefore not a very interesting choice for the neighborhood function for local search.

4.3 Search strategy

The most obvious search strategy is the *steepest descent* method. It evaluates all the candidate solutions in the neighborhood and selects the one minimizing the objective function. With a large number of candidates the search is more selective as it seeks for the maximum improvement. An alternative approach is the *first-improvement* method, which accepts any candidate solution if it improves the objective function value. This is effectively the same as the steepest descent approach with the neighborhood size 1. The studies in [26] indicate that the first-improvement method would be a better choice in most real world problems. It is expected to find the optimum faster because there are less candidates to be evaluated, and it is also less likely to get stuck into a local optimum.

Suboptimal moves can also be allowed during the search by accepting solutions that do not improve the objective function value. The motivation of allowing suboptimal moves is to help the search to pass over local minima. *Stochastic relaxation*, for example, adds noise to the evaluation of the objective function. The amount of noise gradually decreases at each iteration step and eventually, when the noise has been completely eliminated, the search reduces back to normal local search. *Tabu search* uses a *tabu list* of previous moves and prevents the search from returning to solutions that have been visited recently.

In a typical clustering problem, however, the local optima are not a serious problem. The size and dimensions of the search space are usually very large. Local search with a large neighborhood and random sampling are therefore sufficient for preventing the search from getting stuck to a local minima. The key is to select a neighborhood function that includes randomness but is also capable for generating competitive candidates.

There is also an important group of deterministic neighborhood functions that generates only one candidate. Such methods are the *k-means* algorithm, the deterministic swap (Section 4.2.1), and the split-and-merge method (Section 4.2.1). Local search based on these neighborhood functions can gradually improve any solution until a local minimum is reached. In principle, these methods could be improved by repeating the search from several different starting points. However, the experiments in [5, 6, 12] have demonstrated that the repetition is not a successful approach for improving descent methods in general.

5. Randomized local search

Our design for the local search based clustering algorithm is described in Fig. 3. In the algorithm we combine the best ideas from the previous section so that the following three objectives would be satisfied. The main goal is to obtain a high quality clustering. Simple implementation and low computational complexity are the other criteria in the design.

Initialization: The initial solution is generated by taking *M* randomly chosen data objects as the cluster representatives. Optimal partition is generated using the nearest neighbor condition. This initialization is simple to implement and it distributes the clusters evenly all over the data space except to the unoccupied areas. However, practically any valid solution would be good enough because the algorithm is designed to be insensitive to the initialization.

Iterations: We apply the first-improvement strategy by generating only one candidate at each iteration. The candidate is accepted if it improves the current solution. This approach is simplest to implement and the use of larger neighborhood was not shown to give any improvement over the first-improvement approach. The algorithm is iterated for a fixed number of iterations.

Neighborhood function: A new candidate solution is generated using the following operations. The clustering structure of the current solution is first modified using the random swap technique. The partition of the new solution is then adjusted (in respect to the modified set of cluster representatives) by the local repartition operation. The cluster representatives are also updated according to the fine-tuned partition by calculating the optimal cluster representatives. Finally, the quality of the new solution is evaluated and compared to the previous solution. The three step procedure generates global changes to the clustering structure and at the same time it performs local fine-tuning for the partition and for the cluster representatives.

K-means iterations: The random swap modifies the clustering structure by changing one cluster per iteration. However, even a single swap is sometimes too big change in comparison to the amount of local fine-tuning. The local refinement can therefore be enhanced by applying the *k-means* algorithm on each iteration. This would direct the search more efficiently by resulting in better intermediate solutions but it would also slow down the algorithm. In practice, the *k-means* algorithm can be implemented much faster within the *RLS* algorithm because of two reasons: (i) using the grouping technique of [27] most of the distance calculations can be avoided when *k-means* is applied to a solution that is already reasonably close to a local optimum; (ii) only two *k-means* iterations is usually enough [5, 6].

Demonstration: A single step of the algorithm is illustrated in Fig. 4 for a data set containing 15 distinctive but somewhat mixed clusters. In the original solution (Fig. 4a), there are two incorrect placements of clusters where there are clearly separate sets of data points partitioned into one clusters. At the same time, there are two smaller partitions at places where there should not be any. Next we demonstrate how the local search algorithm is capable of correcting one of the two misplacements of the clusters.

The algorithm proceeds by removing the top rightmost cluster (Fig. 4b) and by creating a new cluster centroid at a randomly chosen place, which happens to be not so far away. The objects in the obsolete cluster are repartitioned into their neighboring clusters, and the new cluster is created by attracting objects from the neighboring clusters (Fig. 4c). Note that the random swapping only partially solves the problem. Although it successfully removes a false cluster, it puts the new centroid into a place where another cluster is already located. Nevertheless, this is good enough because the *k-means* iterations are now capable of making the necessary fine-tuning by moving the other cluster centroid down to the place where the new cluster should be created (Fig. 4d). The new solution is better than the original one but further iterations of local search are still required to correct the other incorrect placement of a cluster.

Computational complexity: The bottleneck of the algorithm (without the *k-means*) is the local repartition phase. It requires $O(N)$ time originating from the distance calculations. The expected number of distance calculations is N in the object rejection phase, and $2N$ in the object attraction phase. In addition to that, the evaluation of the objective function value requires N distance calculations summing up to $4N$ distance calculations in total. If the *k-means* iterations are included the number of distance calculations is $O(NM)$.

***RLS* algorithm 1:**

$C \leftarrow$SelectRandomDataObjects(M).
$P \leftarrow$OptimalPartition(C).

REPEAT T times
    $C^{new} \leftarrow$RandomSwap(C).
    $P^{new} \leftarrow$LocalRepartition(P,C^{new}).
    $C^{new} \leftarrow$OptimalRepresentatives(P^{new}).
    IF $f(P^{new}, C^{new}) < f(P, C)$ THEN
        $(P, C) \leftarrow (P^{new}, C^{new})$

***RLS* algorithm 2:**

$C \leftarrow$SelectRandomDataObjects(M).
$P \leftarrow$OptimalPartition(C).

REPEAT T times
    $C^{new} \leftarrow$RandomSwap(C).
    $P^{new} \leftarrow$LocalRepartition(P,C^{new}).
    K-means(P^{new},C^{new}).
    IF $f(P^{new}, C^{new}) < f(P, C)$ THEN
        $(P, C) \leftarrow (P^{new}, C^{new})$

Figure 3. Two variants of the local search algorithm. The difference between the algorithms is that the *RLS*-2 algorithm performs two *k-means* iterations instead of the *OptimalRepresentatives*-operation. The number of iterations is fixed to T=5000 in the rest of the paper.

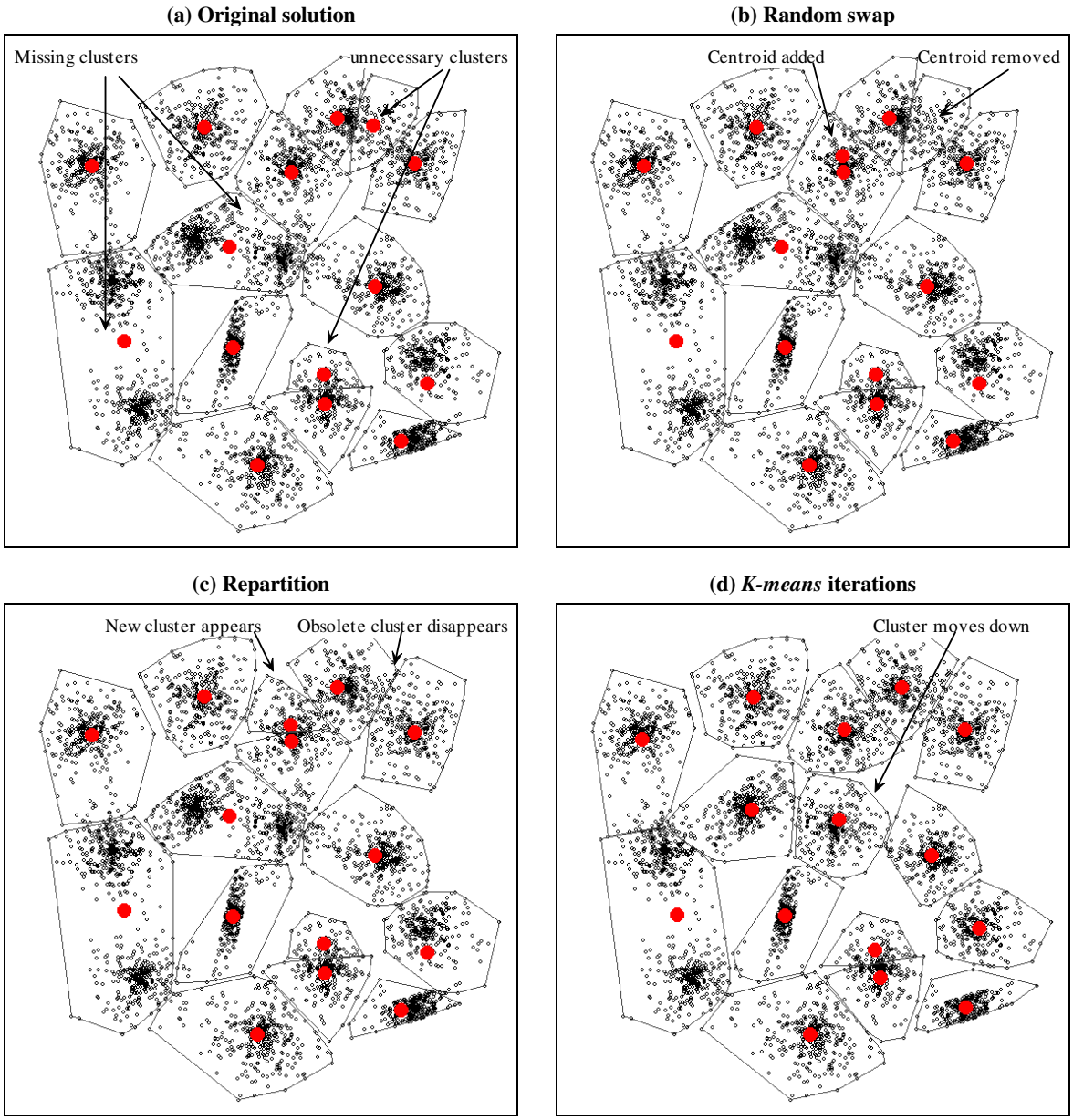

Figure 4. Illustration of the process for generating a new candidate solution by *RLS-2*. Clusters are illustrated by drawing convex hulls around them. The larger dots represent the cluster centroids.

6. Example data sets

The following data sets are considered: *Bridge*, *Bridge-2*, *Miss America*, *House*, *Lates mariae* and *SS2*. Due to our vector quantization and image compression background, the first four data sets originate from this context. We consider these data sets merely as test cases of the clustering problem.

In vector quantization, the aim is to map the input data objects (vectors) into a representative subset of the vectors, called *code vectors*. This subset is referred as a *codebook* and it can be constructed using any clustering algorithm. In data compression applications, reduction in storage space is achieved by storing the index of the nearest code vector instead of each original data vector. More details on the vector quantization and image compression applications can be found in [3, 28, 29].

Bridge consists of 4×4 spatial pixel blocks sampled from the image (8 bits per pixel). Each pixel corresponds to a single attribute having a value in the range [0, 255]. The data set is very sparse and no clear cluster boundaries can be found. *Bridge-2* has the blocks of *Bridge* after a BTC-like quantization into two values according to the average pixel value of the block [30]. The attributes of this data set are binary values (0/1) which makes it an important special case for the clustering. According to our experiments, most of the traditional methods do not apply very well for this kind of binary data.

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. The application of this kind of data is found in video image compression [31]. The data set is similar to the first set except that the data objects are presumably more clustered due to the motion compensation (subtraction of subsequent frames).

The fourth data set (*House*) consists of the *RGB* color vectors from the corresponding color image. This data could be applied for palette generation in color image quantization [32, 33]. The data objects have only three attributes (red, green and blue color values) but there are a high number of samples (65536). The data space consists of a sparse collection of data objects spread into a wide area, but there are also some clearly isolated and more compact clusters.

The fifth data set (*Lates mariae*) records 215 data samples from pelagic fishes on Lake Tanganyika. The data originates from a research of biology, where the occurrence of 52 different DNA fragments were tested from each fish sample (using *RAPD analysis*) and a binary decision was obtained whether the fragment was present or absent. This data has applications in studies of genetic variations among the species [34]. From the clustering point of view the set is an example of data with binary attributes. Due to only a moderate number of samples (215), the data set is an easy case for the clustering, compared to the first four sets.

The sixth data set is the standard clustering test problem SS2 of [35], pp. 103-104. The data set contains 89 postal zones in Bavaria (Germany) and their attributes are the number of self-employed people, civil servants, clerks and manual workers in these areas. The attributes are normalized to the scale [0, 1] according to their minimum and

maximum values. The dimensions of this set are rather small in comparison to the other sets. However, it serves as a school book example of a typical small scale clustering problem.

The data sets and their properties are summarized in Fig. 5. In the experiments made here, we will fix the number of clusters to 256 for the image data sets, 8 for the DNA data set, and 7 for the SS2 data set. The data samples of the binary set *Lates Mariae* are treated as real numbers in the range [0, 1].

Bridge	*Bridge-2*	*Miss America*	*House*	*Lates mariae*
(256×256)	(256×256)	(360×288)	(256×256)	(52×215)
K=16, *N*=4096	*K*=16, *N*=4096	*K*=16, *N*=6480	*K*=3, *N*=65536	K=52, N=215

Figure 5. Sources for the first five data sets.

7. Test results

We study the performance of the main *RLS*-variants (*RLS-1* and *RLS-2*) by generating clusterings for the six data sets introduced in Section 6. We are primarily interested in the objective function value and the time spent in the clustering process. Although the quality is the primary aim of the clustering the run time is also important. We use the following methods as the point of comparison:

- *Random clustering*
- *K-means* [4]
- *Ward's method* [8]
- *Tabu search* (*TS*) [6]
- *Genetic algorithm* (*GA*) [5]

Random clustering is generated by selecting M random data objects as cluster representatives and by mapping all the data objects to their nearest representative, according to the distance function d. The random clustering is used as the starting point in the *k-means*, *TS* and *GA* implementations. The parameter setup for the *TS* and *GA* methods are the best ones obtained in [5] and [6]. The *Ward's method* (or *PNN*) is a variant of the agglomerative clustering. We use the implementation presented in [15].

The time complexities of the *RLS-1*, *RLS-2*, *k-means* and the *Ward's* methods are $O(T_1N)$, $O(T_1NM)$, $O(T_2NM)$ and $O(\tau N^2)$, where T_1 and T_2 denote to the number of iterations in the *RLS* and in the *k-means* algorithms. In a single iteration the *RLS-1* algorithm is significantly faster than the *k-means* but the total number of iterations is much higher. The actual run times depend on the data set and implementation details. The *Ward's method* is asymptotically slower than the other methods but it is fast enough for smaller data sets to be competitive.

The clustering results are summarized in Tables 1, 2 and 3. The *k-means* is clearly the fastest (excluding the random clustering) but the quality is worst. The *TS* and *GA* methods are in the other extreme: they produce the best clustering results but the methods are significantly slower than the rest. The *RLS* algorithms offer good trade-offs. The *RLS-1* gives slightly better results than the *Ward's method* with a slightly faster algorithm. In the case of a very large data set (*House*) the difference in run time is remarkable. The results of the *RLS-2* method are comparable to those of the *TS* with much smaller run times. The results of the *GA* are sometimes better but at the cost of much higher run time.

We stated in Section 4 that the *RLS* algorithm is designed to be insensitive to the initialization. This can be verified in two ways: (i) by repeating the algorithm starting from different random initializations, and (ii) starting from the result of another clustering algorithm. The results in Fig. 6 show that there is very little variation between the different runs. The *RLS* algorithm is also capable to improve the clustering result of all the other algorithms tested. The *RLS-2* gives results of similar quality regardless of the method used in the initialization. The *k-means* method, on the other hand, is much more sensitive to the initialization.

Overall, the *RLS* algorithms converge rather slowly and a relatively large number of iterations is therefore needed, see Fig. 7. In the case of *Bridge* and *House*, the *RLS-1* outperforms the *k-means* after about 1000 iterations, the *RLS-2* after 5 to 10 iterations. The *RLS-2* outperforms the *Ward's method* after about 500 iterations. From Fig. 7 it is also obvious that there is room for further improvement if more time is spent. Additional results have shown that the *RLS-2* outperforms the *GA* after about 10 hours of time for *Bridge*. The best result we have obtained is *161.33* after one million iterations (about 80 hours).

The use of more than one candidate seems not to be of any help. The current experiments confirm the previous results of [5, 6] indicating that the quality of the best clustering depends mainly on the total number of candidate solutions tested. However, if the run time is limited, the first-improvement approach was shown to be better than the use of a larger neighborhood with a limited number of iterations. Otherwise we did not observe any significant differences between the two approaches.

The last parameter of the *RLS* algorithm is the neighborhood function. We chose the random swap because it is the best compromise between quality, run time, and simplicity of implementation. In some cases the *split-and-merge* approach [12] would have been a better choice (similar results with a faster algorithm) but the method is much more complex to implement and it performs worse in the case of binary data sets. The *deterministic swap* [24] was not very promising being always slower and giving worse results than the random swap (with or without *k-means* iterations). The biggest deficiency of these two approaches is that they are deterministic descent methods seeking for the nearest local minimum. Unlike the random swap, they won't give significant improvement anymore after finding a local minimum.

Table 1. Performance comparison of various algorithms. The results of the *random clustering* are the best results of 100 test runs. The results of the *k-means* are averages of 100 test runs, *RLS* results of 10 test runs, *TS* and *GA* results of 5 test runs. *Ward's method* is a deterministic method and its result is always the same. The total number of tested candidates in the (*k-means*, *RLS*, *TS*, and *GA*) were (22, 5000, 10000, 2250).

	Bridge	***Miss America***	***House***	***Bridge-2***	***Lates mariae***	***SS2***
Random	251.32	8.34	12.12	1.51	0.120	1.76
K-means	179.68	5.96	7.81	1.48	0.071	1.14
Ward	169.15	5.52	6.36	1.44	0.063	0.34
RLS-1	169.67	5.44	6.20	1.28	0.063	0.31
RLS-2	164.64	5.28	5.96	1.26	0.063	0.31
TS	164.23	5.22	5.94	1.27	0.063	0.31
GA	162.09	5.18	5.92	1.28	0.063	0.31

Table 2. The standard deviations of the repeated test results of Table 1.

	Bridge	***Miss America***	***House***	***Bridge-2***	***Lates mariae***	***SS2***
Random	4.765	0.169	13.194	0.019	0.996	2.647
K-means	1.442	0.056	0.196	0.015	0.457	0.899
Ward	0.000	0.000	0.000	0.000	0.000	0.000
RLS-1	0.452	0.025	0.022	0.004	0.000	0.000
RLS-2	0.205	0.022	0.010	0.001	0.000	0.000
TS	0.345	0.012	0.010	0.001	0.000	0.000
GA	0.305	0.005	0.009	0.002	0.000	0.000

Table 3. Run time (min:s) comparison of various algorithms. It is noted that the parameter setup in the cases of *RLS*, *TS* and *GA* were optimized for large data sets. For example, the optimal clustering for the smaller sets (*Lates mariae*, *SS2*) could be found with a significantly less number of iterations.

	Bridge	***Miss America***	***House***	***Bridge-2***	***Lates mariae***	***SS2***
Random	0:01	0:03	0:06	0:01	0:00	0:00
K-means	0:25	0:58	4:53	0:07	0:00	0:00
Ward	9:48	13:32	177:54	2:12	0:01	0:00
RLS-1	5:28	8:25	14:22	3:31	0:48	0:03
RLS-2	35:40	34:23	53:10	6:26	1:33	0:09
TS	296:28	1149:19	2153:42	376:11	4:45	0:21
GA	135:02	535:59	904:47	236:43	2:05	0:11

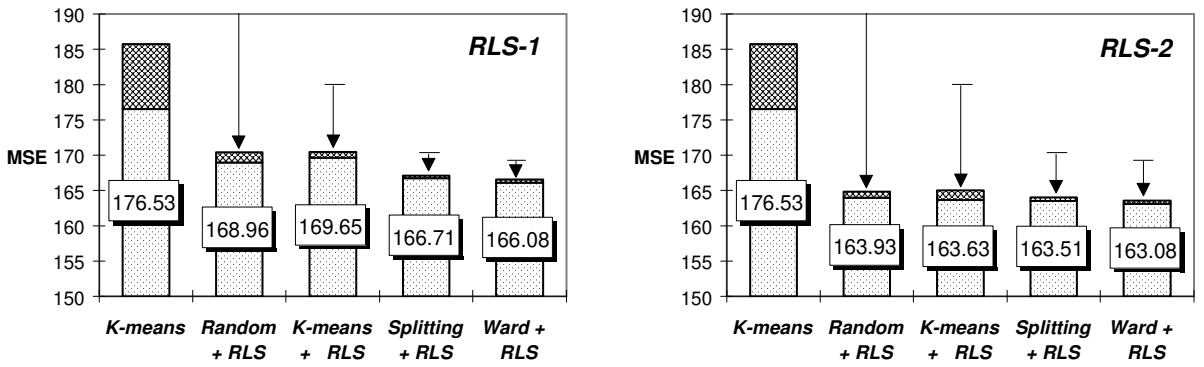

Figure 6. The effect of the initialization on the performance of the *RLS* algorithms (for *Bridge*). The methods were repeated 10 times each. The darker part in the columns represents the difference between the best and the worst results. The numbers in the boxes are for the best results. The arrows indicate the amount of improvement from the initial clustering.

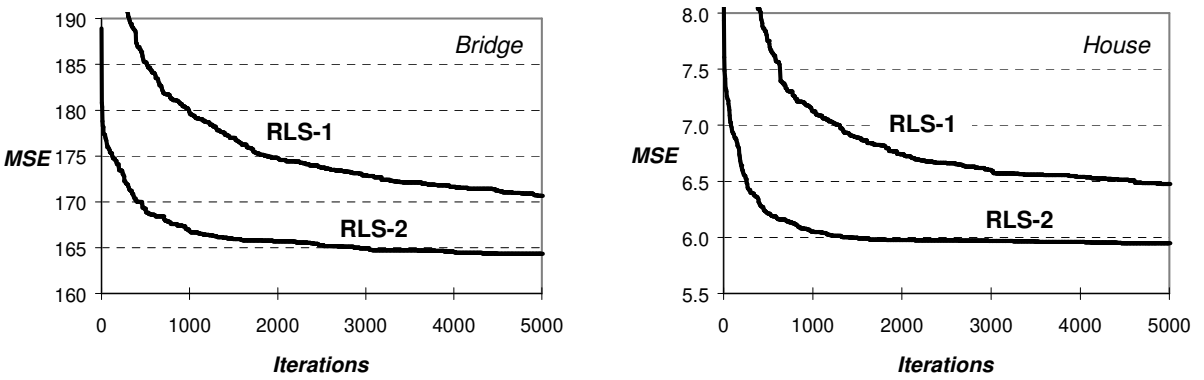

Figure 7. Quality of the clustering as a function of the iterations.

8. Summary

The key question of cluster analysis is how to partition a set of data objects into a given number of clusters. We consider the clustering as combinatorial optimization problem. The main goal is to obtain a high quality clustering but simplicity of implementation and low computational complexity are also important criteria in the design of the algorithm. The *k-means* and the *Ward's method* are the most widely used methods probably because of their easy implementation and reasonable results. The best clustering results, however, have been obtained by optimization techniques such as *tabu search* (*TS*) and *genetic algorithms* (*GA*).

We introduce a new clustering algorithm based on the traditional *local search*. The key questions in the design are the representation of a solution, the neighborhood function, and the search strategy. The neighborhood function is balanced between random and deterministic modifications. The modifications make global changes to the clustering structure and at the same time perform local fine tuning towards a local optimum. A large number of iterations is needed since the method converges slowly due to the randomized approach. The modifications, however, are rather simple and can be efficiently performed. A simple first-improvement approach is used as the search strategy.

Experiments show that the new *randomized local search* (*RLS*) approach outperforms the *k-means* and the *Ward's method*. The method gives also competitive results to those of the *GA* and *TS* with a faster algorithm, and with a much simpler implementation. It thus offers a good trade-off between the quality of the clustering and the complexity of the algorithm. The *RLS* algorithm is relatively insensitive to the initialization and was capable to improve the clustering result of all the tested algorithms. There was very little variation between different runs.

With simple modifications, the method can be implemented for other distance metrics and evaluation criteria. For example, *stochastic complexity* has been applied with the method as the evaluation function for the classification of bacteria [36]. The *RLS* algorithm can be applied to the more general case where the number of clusters must also be solved as well. A straightforward approach repeats the clustering with various number of clusters using suitable evaluation criterion where the number of clusters is a parameter. For more efficient implementation, the random swap operation itself can be modified by making the cluster removal and creation the basic operations. In this way, the number of clusters would be variable during the search and the same search strategy could be applied with a single run of the algorithm.

References

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

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

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

[4] J.B. McQueen, "Some methods of classification and analysis of multivariate observations", *Proc. 5th Berkeley Symp. Mathemat. Statist. Probability* **1**, 281-296. Univ. of California, Berkeley, USA, 1967.

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

[6] P. Fränti, J. Kivijärvi and O. Nevalainen, "Tabu search algorithm for codebook generation in VQ", *Pattern Recognition*, **31** (8), 1139-1148, August 1998.

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

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

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

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

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

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

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

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

[15] 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), May 2000. (in press)

[16] X. Wu and K. Zhang, "A Better Tree-Structured Vector Quantizer", *Proc. Data Compression Conference*, Snowbird, Utah, 392-401, 1991.

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

[18] S. Kotz, N.L. Johnson and C.B. Read, (editors) *Encyclopedia of Statistical Sciences*, **6**, John Wiley Sons, New York, 1985.

[19] K. Al-Sultan, A tabu search approach to the clustering problem, *Pattern Recognition* **28** (9), 1443-1451, September 1995.

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

[21] N.M. Nasrabadi and Y. Feng, "Vector quantization of images based upon the Kohonen self-organization feature maps", *Neural Networks*, **1** (1), 518, 1988.

[22] C. Reeves, *Modern Heuristic Techniques for Combinatorical Optimization Problems,* McGraw - Hill, 1995.

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

[24] B. Fritzke, "The LBG-U method for vector quantization −an improvement over LBG inspired from neural networks", *Neural Processing Letters*, **5** (1), 35-45, 1997.

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

[26] E.J. Anderson, "Mechanisms for local search", *European Journal of Operational Research*, **88** (1), 139-151, January 1996.

[27] T. Kaukoranta, P. Fränti and O. Nevalainen, "A fast exact GLA based on code vector activity detection", *IEEE Trans. on Image Processing*. (accepted for publication)

[28] N.M. Nasrabadi and R.A. King, "Image coding using vector quantization: a review". *IEEE Transactions on Communications*, **36** (8), 957-971, 1988.

[29] C.F. Barnes, S.A. Rizvi and N.M. Nasrabadi, "Advances in residual vector quantization: a review". *IEEE Transactions on Image Processing*, **5** (2), 226-262, February 1996.

[30] P. Fränti, T. Kaukoranta and O. Nevalainen, "On the design of a hierarchical BTC-VQ compression system", *Signal Processing: Image Communication*, **8** (11), 551-562, 1996.

[31] J.E. Fowler Jr., M.R. Carbonara and S.C. Ahalt, "Image coding using differential vector quantization". *IEEE Transactions on Circuits and Systems for Video Technology*, **3** (5), 350-367, October 1993.

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

[33] X. Wu, "YIQ Vector quantization in a new color palette architecture". *IEEE Transactions on Image Processing*, **5** (2), 321-329, February 1996.

[34] L. Kuusipalo, "Diversification of endemic Nile perch *Lates Mariae* (Centropomidae, Pisces) populations in Lake Tanganyika", East Africa, studied with RAPD-PCR. *Proc. Symposium on Lake Tanganyika Research*, 60-61, Kuopio, Finland, 1995.

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

[36] P. Fränti, H.H. 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*. (accepted for publication)