UNIVERSITY OF JOENSUU
DEPARTMENT OF COMPUTER SCIENCE
Report Series A

Dynamic local search algorithm for the clustering problem

Ismo Kärkkäinen and Pasi Fränti

Report A-2002-6

ACM I.4.2, I.5.3
UDK 519.712
ISSN 0789-7316
ISBN 952-458-143-4

Dynamic local search algorithm for the clustering problem

Ismo Kärkkäinen and Pasi Fränti

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

Tel. +358 – 13 251 7959; Fax. +358 – 13 251 7955

Abstract: Dynamic clustering problems can be solved by finding several clustering solutions with different number of clusters, and by choosing the one that minimizes a given evaluation function value. This kind of brute force approach is general but not very efficient. We propose a dynamic local search 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. The clustering is found by the combination of trial-and-error approach of local search. The algorithm finds the result 30 times faster than the brute force approach.

Keywords: clustering, number of clusters, algorithms, optimization, vector quantization.

1. Introduction

Clustering is an important 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 (Everitt 1992, Kaufman and Rousseeuw 1990, Jain et al. 1999). It aims at answering two main questions: *how many clusters* there are in the data set and *where* they are located. We define the problem here as *static clustering* if the number of clusters is known beforehand, and as *dynamic clustering* if the number of clusters must also be solved. Clustering is also needed for designing a *codebook* in vector quantization (Gersho and Gray 1992).

Static clustering problem can be solved by methods such as *Generalized Lloyd algorithm* (GLA) (Linde et al, 1980), *simulated annealing* (Zeger and Gersho, 1986), *deterministic annealing* (Rose et al. 1990, Hoffmann and Buhmann 1997), *genetic algorithm* (Fränti et al 1997, Fränti 2000), *agglomerative methods* (Ward 1963) among many others. *Randomized Local Search* (*RLS*) is a good choice for the clustering because of its competitive performance according to the results by Fränti and Kivijärvi (2000) in terms of optimizing the evaluation function value. Its simplicity makes it easy to generalize for the dynamic clustering problem, too.

The dynamic clustering problem can be solved by *Brute Force* (BF) approach as follows. First a suitable *evaluation function* is defined, which includes the number of clusters as a parameter. Any static clustering algorithm is then applied for all reasonable number of clusters. The resulting solutions are compared using the evaluation function, and the final result is the clustering that minimizes the evaluation function. Criteria such as *Davies-Bouldin index* (1979), and *variance-ratio F-test* (Ito, 1980) can be used among many others (Bezdek and Pal, 1998). The advantage of this kind *criterion-based* approach is that the existing algorithms for the static clustering can be utilized as such.

The Brute Force approach is general but inefficient. A more efficient solution is to perform *Stepwise local search*, in which the previous clustering (with m clusters) is utilized when solving the current one (with m+1 clusters), and by defining appropriate stopping criterion for the iterations (Kärkkäinen and Fränti, 2002). The Stepwise algorithm is more efficient but still uses unnecessary time for optimizing solutions with wrong number of clusters.

In this paper, we propose a more efficient approach called *dynamic local search* (DLS). It optimizes the number and the location of the clusters jointly. The main motivation is that most of the computation should be spent on solutions with the correct, or nearly correct number of clusters. Time should not be wasted for optimizing clustering that has very little relevance to the final solution. We first derive a set of basic operations: *cluster addition, cluster removal* and *cluster swapping*, and then study how the operations should be applied in order to achieve the correct clustering in most efficient way.

The main problem in the dynamic local search is the following. In static clustering, the RLS can solve the correct clustering starting from any initial solution. The longer the algorithm is iterated, the more likely the correct result is reached. In the dynamic approach, however, the optimization function can have local minima with the changes of M. The algorithm must therefore be able to reallocate more than one cluster at a time. This can be major source of inefficiency if not properly designed.

We study experimentally different search strategies in the dynamic local search. We will give a feasible solution that avoids the local minima, and is significantly faster than the Brute Force approach. We will show by experiments, that the proposed algorithm finds the correct number of clusters by about 30 times faster than the Brute Force approach. We also give a simple stopping criterion that indicates when the iterations can be stopped with high confidence that the correct clustering has been reached.

The rest of the paper is organized as follows. We first recall the solutions for the static clustering problem in Section 2, and then present the new algorithm in Section 3. Experiments are then performed in Section 4, and conclusions drawn in Section 5.

2. Clustering problem

The *clustering problem* is defined here as follows. Given a set of N data vectors $X=\{x_1, x_2, \ldots, x_N\}$, partition the data set into M clusters such that similar vectors are grouped together and dissimilar vectors to different groups. Partition $P=\{p_1, p_2, \ldots, p_N\}$ defines the clustering by giving for each data vector the index of the cluster where it is assigned to. In vector quantization, the output of the clustering is a codebook $C=\{c_1, c_2, \ldots, c_M\}$, which is usually the set of cluster centroids.

We assume that the data set is normalized and the clusters are spherical, so that some standard distance metric, e.g. *Euclidean distance*, can be applied. This allows us to estimate the goodness of a solution of M clusters by calculating the *mean square error* (MSE) of the distances from the data vectors to their nearest cluster centroid. After these presumptions, the clustering problem can be formulated and solved as a *combinatorial optimization problem*.

2.1 Algorithms for static clustering

A well-known clustering algorithm for minimizing MSE is the *Generalized Lloyd algorithm*, also known as the LBG due to Linde, Buzo and Gray (1980). It is simple to implement and it can be applied to any initial solution. It performs iterative repartition of the data vectors and recalculation of the cluster centroids until the solution converges, or when a predefined maximum number of

iterations has been performed. The pseudocode is shown in Figure 1. The advantage of the GLA is that it can be applied to the output of any other algorithm, either integrated or as a separate step.

Another simple clustering method is the *Randomized local search* (RLS) by Fränti and Kivijärvi (2000), which has been shown to outperform most of the comparative algorithms. The method is based on a trial-and-error approach as follows. New candidate solutions are generated by *random swap* operation, which reallocates a randomly chosen cluster to another part of the vector space. The new cluster is located to the position of a randomly drawn vector from the data set. The partition is then modified according to the change in the clustering structure, and few iterations of the GLA are applied as fine-tuning. The new solution replaces the previous one only if it decreased the error value. The longer the algorithm is iterated, the better is the clustering. The pseudocode is presented in Figure 2.

GLA(*X*, *C, Iterations*) return *C*, *P*
REPEAT *Iterations* TIMES
- Generate partition *P* by mapping all data vectors to cluster represented by the nearest centroid vector
- Generate *C* by calculating mean vector for each partition

Return *C*, *P*;

Fig. 1: Pseudocode for the GLA.

RLS(*X*, *C*) return *C*, *P*
$C_{best} \leftarrow C$;
Generate partition P_{best} using C_{best} by mapping all data vectors to cluster represented by the nearest centroid vector
REPEAT *Iterations* TIMES
- $C \leftarrow C_{best}$;
- Swap randomly chosen centroid to randomly chosen data vector
- Generate partition *P* using *C* by mapping all data vectors to cluster represented by the nearest centroid vector
- *C*, *P* ← GLA(*X*, *C*, 2);
- Update the best solution if new solution is better:
  IF MSE(*X*, *C*, *P*) < MSE(*X*, C_{best}, P_{best}) THEN
  $C_{best} \leftarrow C$; $P_{best} \leftarrow P$;

Return *C*, *P*;

Fig. 2: Pseudocode for the RLS.

2.2 Algorithms for dynamic clustering

In many cases, the number of clusters is not known beforehand but solving the correct number of clusters is part of the problem. The simplest approach is to generate solutions for all possible number of clusters M in a given range $[M_{min}, M_{max}]$, and then select the best clustering according to a suitable evaluation function f. This approach is referred here as *Brute Force* (BF) algorithm. It allows us to use any static clustering algorithm in the search. The pseudo code is given in Figure 3.

The choice of the evaluation function is a vital part of the clustering; several candidates were presented by Bezdek and Pal (1998). We consider two criteria. The first is *Davies-Bouldin index* (DBI) by Davies and Bouldin (1979). It has been applied by Bezdek and Pal (1998), Sarkar, Yegnanarayana and Khemani (1997), and Kärkkäinen and Fränti (2000). The DBI measures for each cluster the ratio of the intracluster distortion relative to the inter cluster distance of the nearest cluster. This is denoted here as the *mixture* of the two clusters j and k, and is calculated as:

$$R_{j,k} = \frac{MAE_j + MAE_k}{d(c_j, c_k)} \tag{1}$$

Here d is the distance between the cluster centroids, and MAE_j and MAE_k are the mean absolute errors within the clusters j and k. The higher the intracluster distortion and the closer their centroids, the higher is the index R.

The *mixture* of a cluster j is defined as the maximum mixture between cluster j and all other clusters. The overall DBI-value is then calculated as the average mixtures of the clusters:

$$DBI = \frac{1}{M}\sum_{j=1}^{M}\max_{j\neq k} R_{j,k} \tag{2}$$

The second criterion we use is the *Variance-ratio F-test* based on a statistical ANOVA test procedure (Ito 1980). We have modified the test so that we omit the checking of the obtained values against F-distribution and use the values directly instead.

In principle, any other function can also be used for guiding the search. If the evaluation function and the clustering algorithm are properly chosen, the Brute Force approach should find the correct solution but the algorithm will be slow.

BruteForce(X, M_{min}, M_{max}) return C, P
Generate random solution C_{best} with M_{min} clusters.
Generate C_{best}, P_{best} with RLS(X, C_{best});
FOR $m \leftarrow M_{min} + 1$ TO M_{max} DO
- Generate random solution C with m clusters
- Generate C, P with RLS(X, C);
- If new solution is better than the best one found so far, replace the best solution by the new one:
  IF $f(X, C, P) < f(X, C_{best}, P_{best})$ THEN
  $C_{best} \leftarrow C$; $P_{best} \leftarrow P$;

END FOR
Return C_{best}, P_{best};

Fig. 3: Pseudocode for the Brute Force algorithm.

The dynamic clustering problem can also be attacked by heuristic methods. For example, the *competitive agglomeration* by Frigui and Krishnapuram (1997) decreases the number of clusters until there are no clusters smaller than a predefined threshold value. The drawback is that the threshold value must be experimentally determined. *Divisive clustering* uses an opposite, top-down approach for generating the clustering. The method starts with a single cluster, and new clusters are then created by dividing existing clusters until a predefined stopping criterion is met. The divisive approach typically requires much less computation than the agglomerative clustering methods but are far less used because of inferior clustering results.

3. Dynamic local search

We next generalize the RLS method so that it solves both the number and the location of the clusters jointly. We refer this algorithm as *Dynamic Local Search* (DLS). The input are the data set (X), an initial solution (C, P), and the search range for the number of clusters (M_{min}, M_{max}). The algorithm applies elementary operations to the current solution and proceeds as the RLS, see Figure 4. The algorithm, however, have two main differences to the RLS. First, we have not only the random swap operation but also we may add or remove clusters. The second difference is that we must use an evaluation function that is able to evaluate and compare clustering solutions with different number of clusters.

3.1 Elementary operations

The current solution is modified by two different ways. The first way is to apply an operator that makes a radical change to the solution. This change does not necessarily result in a better solution in itself, but it allows the search to proceed away from local minimum. The following elementary operations are used for modifying the current solution:

- Cluster swapping,
- Cluster addition,
- Cluster removal.

DLS(X, M_{min}, M_{max}) return C, P
$C_{best} \leftarrow$ Random solution(X, M_{min});
Generate partition P_{best} using C_{best} by mapping all data vectors to cluster represented by the nearest centroid vector
REPEAT *Iterations* TIMES
- $C \leftarrow C_{best}$;
- Select operation randomly and apply it to the solution
- Generate partition P using C by mapping all data vectors to cluster represented by the nearest centroid vector
- C, $P \leftarrow$ GLA(X, C, 2);
- Update the best solution if new solution is better:
  IF $f(X, C, P) < f(X, C_{best}, P_{best})$ THEN
  $C_{best} \leftarrow C$; $P_{best} \leftarrow P$;

Return C, P;

Fig. 4: Pseudocode for the DLS.

The cluster swapping replaces one cluster centroid by a randomly selected vector from the data set. This operator is the same as the random swap in the RLS. Cluster addition creates new clusters by randomly selecting vectors from the data set, and cluster removal removes randomly chosen clusters centroid, see Figures 5 and 6. These operations allow the algorithm dynamically adjust the number of clusters during the search. After any operation, two GLA-iterations are performed.

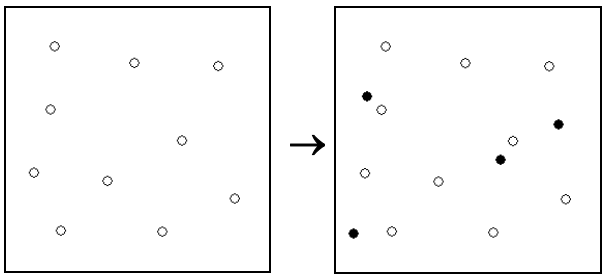


Fig. 5: Adding four random centroids.

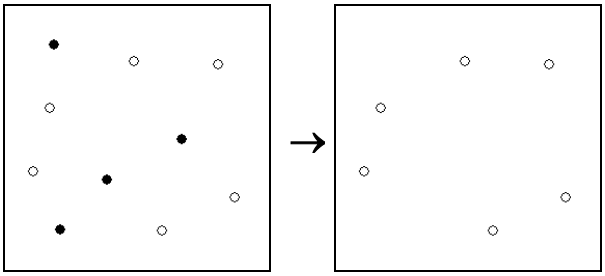


Fig. 6: Removing four random centroids

3.2 Amount of change

In the algorithm, we first select the operator to be applied, and the amount cluster centroids to be changed. In the selection, we use the following probabilities: cluster swap 50%, cluster addition 25%, and cluster removal 25%. Next we select the number of clusters that are added or removed. In the case of cluster swap, swapping of a single cluster is enough according to Fränti and Kivijärvi (2000). In principle, adding or removing only one cluster would also be enough to reach solution of any number of clusters. The evaluation function, however, may include local minima and, therefore, it is important to allow bigger changes take place during the search.

A simple approach to control the amount of change is to select the number of clusters to be added or removed (ΔM) randomly so that smaller changes have a higher probability to appear than bigger changes. This is because if we add several clusters at a time, the new solution can be practically random and we can loose all gain from the previous work made for improving the solution. Let M be the current number of clusters, and L the limit (M_{min} or M_{max}) that bounds the number of clusters, the amount of changes is:

$$\Delta M = 1 + \lfloor r^{\alpha} \cdot |M - L| \rfloor \tag{3}$$

where r is an random number evenly distributed in the range [0, 1[. The power α is a control parameter of the process. In the case of $\alpha = 1$ it gives even distribution. For larger values of α, the distribution is skewed towards small values. Figure 7 gives examples of the probability distribution for a few different values of α for maximum change of 20.

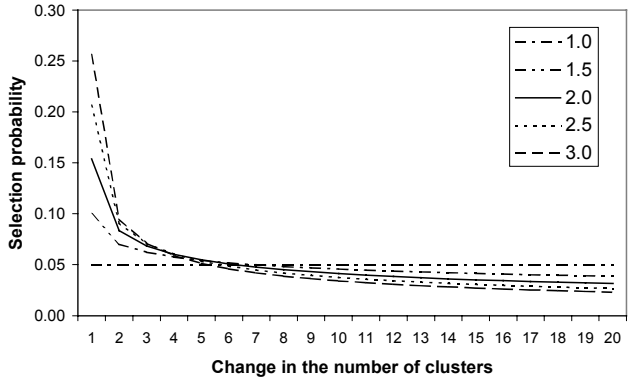


Fig. 7: Probability function for selecting the number of changes for the values α=1.0, 1.5, 2.0, 2.5, 3.0.

3.3 Stopping criterion

There still remains the question of how many iterations we must perform in order to find the correct number of clusters. If we iterate long enough we can expect to find the correct solution eventually. The necessary amount of iterations, however, is not known and it can vary from one data set to another.

Heuristic stopping criteria were designed by Kärkkäinen and Fränti (2002) for the static clustering. A criterion referred as *50-50 ratio* terminates the search when the improvement for the latter 50% of the iterations divided by the improvement made during the first 50% of the improvements drops below a given threshold value, and a given minimum number of iterations has been performed. This criterion worked well for the static clustering, and can also be applied in the DLS. We refer this as the *static 50-50 ratio*.

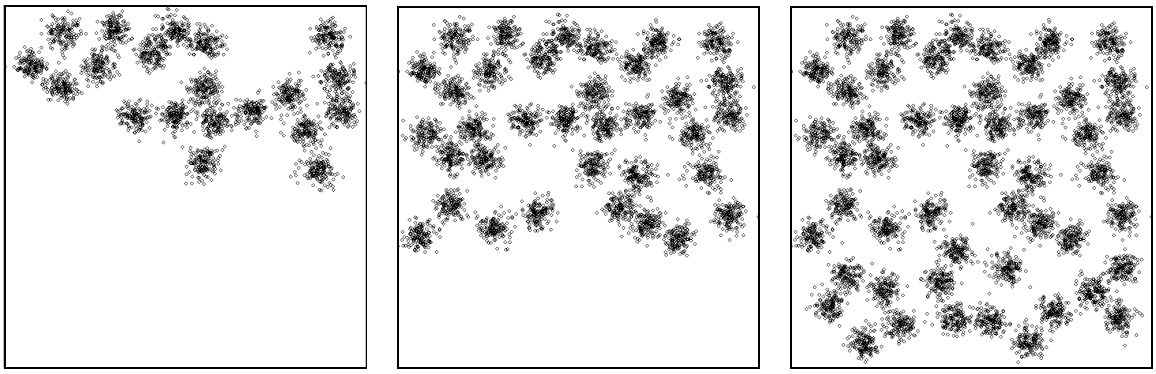
When applied with the DLS, we make a small modification to the *50-50 ratio* as follows. We let the algorithm iterate as long as the number of clusters keeps changing. When it settles, we start to monitor the improvement using the 50-50 ratio as described above. The number of clusters is defined to be settled when it has not changed for a predefined number of iterations (e.g. 200). If the number of clusters changes later, we again wait for the number of cluster to settle before start to monitor the stopping criterion again. We refer this as the *dynamic 50-50 ratio*.

4. Test results

We study the proposed algorithm with the following data sets:

- Data sets A_1, A_2 and A_3 (varying the number of clusters),
- Data sets S_1, S_2, S_3 and S_4 (varying spatial complexity),
- Data sets D_2, D_3, …, D_{15} (varying dimensionality).

The data sets A_1, A_2 and A_3 are two-dimensional sets with varying number of circular clusters (M=20, 35, 50) as illustrated in Fig. 8. The optimized evaluation function values for these data sets are shown in Figure 9. The data sets S_1 to S_4 are two-dimensional sets with varying complexity in terms of spatial data distributions, as shown in Figure 10. All have 15 clusters. The data sets D_2 to D_{15} have slightly less spatial complexity but higher dimensionality (varying from 2 to 15) and 9 clusters. The S and D data sets are the same as described earlier by Kärkkäinen and Fränti (2002).



Data set A_1, 3000 vectors | Data set A_2, 5250 vectors | Data set A_3, 7500 vectors

Fig. 8: Sample data sets A_1, A_2 and A_3.

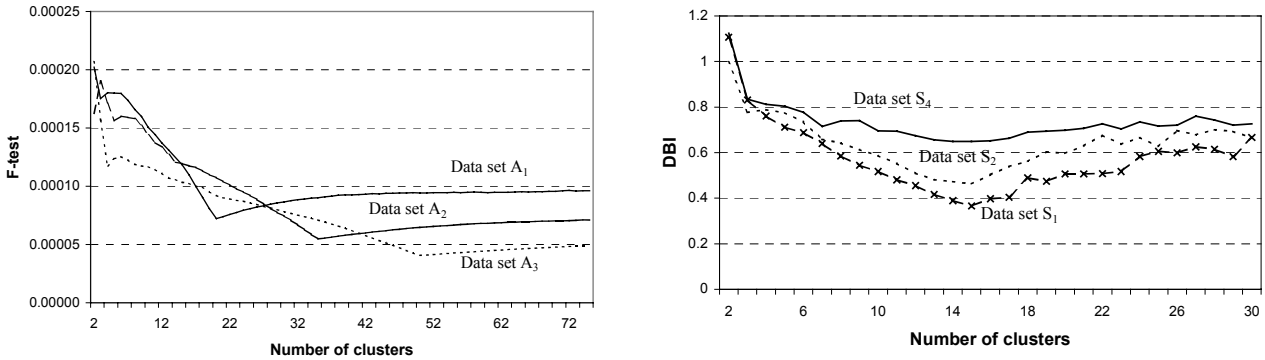
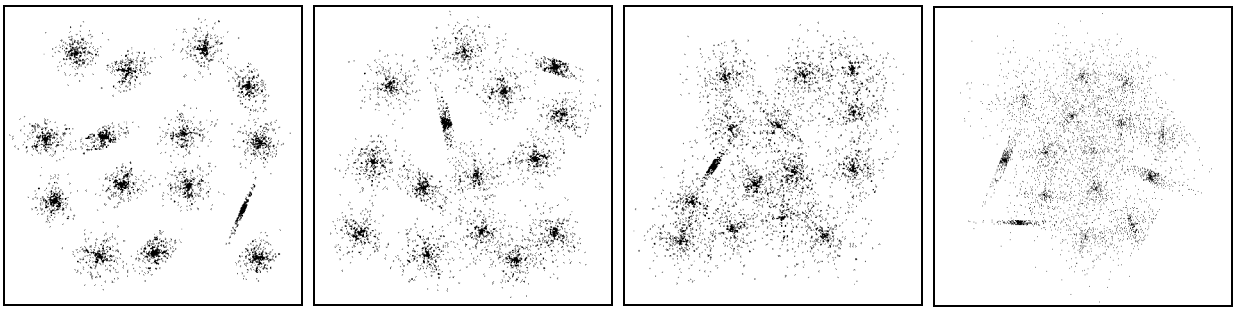


Fig. 9: Evaluation function values for the optimized clustering solutions for the data sets A_1, A_2 and A_3 using F-test (left), and for data sets S_1, S_2 and S_4 using DBI (right).



Data set S_1 Data set S_2 Data set S_3 Data set S_4

Fig. 10: Two-dimensional data sets with varying complexity in terms of spatial data distributions. The data sets have 5000 vectors scattered around 15 predefined clusters with a varying degree of overlap.

4.1 Comparison with Brute Force algorithm

We first study how much faster the DLS can find the correct solution than the Brute Force. For this purpose, we perform both the Brute Force and the DLS algorithms using a predefined search range from M_{min}=2 to M_{max}=75. The general test procedure was to fix the parameter setup and then repeat both algorithms 100 times starting from a different random solutions. The number of iterations in the Brute Force was varied from 100 to 500 per cluster count, corresponding to 7400 to 37000 number of iterations in total. The DLS algorithm was tested with 1000 and 2000 iterations, and by varying the control parameter from α=1.0 to 3.0. The number of GLA-iterations were fixed to two in both algorithms.

The results with the data sets A_1 to A_3 are summarized in Table 1 for the Brute Force, and in Table 2 for the DLS. As the number of clusters is known beforehand, we can calculate how many times the algorithms found the correct number of clusters. Comparison to the Brute Force shows that the DLS (with 1000 iterations) achieves similar success rate with approximately 3% of the amount of work needed by the Brute Force (37000 iterations). Moreover, the DLS obtains 100% success rate if the control parameter is set high enough (α=3.0), and is iterated sufficiently long (2000 iterations). Figure 11 gives three examples of how the best solution develops as the search proceeds in the DLS.

Table 1. Number of times (%) the correct clustering is found by *Brute Force*. The heading numbers are the number of iterations per each value of *M* (the total number of iterations).

Iterations: (total)	100 (7400)	200 (14800)	300 (22200)	500 (37000)
Data set A_1	83	100	100	100
Data set A_2	26	65	83	98
Data set A_3	9	22	43	73

Table 2. Number of times (%) the correct clustering is found by DLS.

	1000 iterations				
	α=1.0	α=1.5	α=2.0	α=2.5	α=3.0
Data set A_1	80	99	100	100	100
Data set A_2	3	37	77	92	98
Data set A_3	0	4	26	49	75
	2000 iterations				
	α=1.0	α=1.5	α=2.0	α=2.5	α=3.0
Data set A_1	98	100	100	100	100
Data set A_2	26	87	99	100	100
Data set A_3	2	36	88	95	100

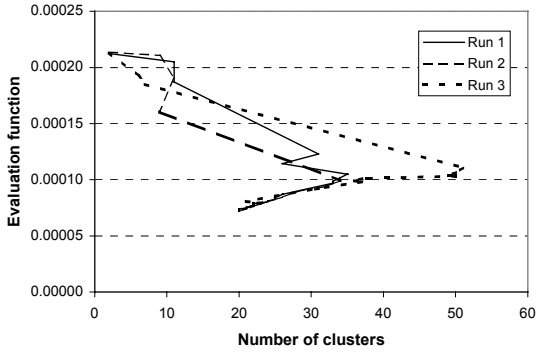


Fig. 11: Development of search for the data set A_1.
The results are for three different runs of the DLS algorithm.

4.2 Stopping criterion

From the previous results we can see that the DLS will find the correct number of clusters when iterated long enough. It would be useful for the algorithm to stop when the solution has stabilized. We test next whether the stopping criterion introduced in Section 3.3 can be reliably used with DLS. We set the minimal number of iterations to 400, and then apply the static 50-50 ratio with the threshold value of 10^{-5}. Results are shown in table 3.

We see that the *static 50-50 ratio* can be reliably used in the case of the first data set. In the case of the other two data sets, the results are slightly worse although the algorithm keeps iterating longer. It seems that the static 50-50 ratio works quite well but the optimal choice for the parameters is not trivial.

Table 3. Percentage the correct clustering is found by *DLS*.
The numbers in parentheses are the average number of iterations performed.

	α=2.0	α=2.5	α=3.0
Data set A_1	98 (861)	98 (670)	99 (626)
Data set A_2	93 (1932)	94 (1422)	90 (1295)
Data set A_3	91 (3479)	89 (2488)	98 (1906)

4.3 Comparison with other approaches

We next compare the proposed DLS algorithm with the following approaches:

- Stepwise with the GLA (Linde, Buzo and Gray, 1980)
- Stepwise with the LBG-U (Frizke, 1997)
- Competitive agglomeration ((Frigui and Krishnapuram, 1997)

The first two approaches integrates the GLA and the LBG-U with the Stepwise algorithm as described by Fränti and Kärkkäinen (2002). The LBG-U is also similar to that of the RLS but it uses deterministic choice for replacing the location of the cluster, whereas the RLS uses random choice. The competitive agglomeration was implanted so that it starts with 100 initial clusters, and then removes all clusters that are smaller than a predefined threshold value ε. The threshold value was experimentally optimized by varying it from 10^{-6} to 10^{-1}.

The results for the data sets A_1 to A_3, and S_1 to S_4 are summarized in Table 4 using both the F-test, and the DBI criteria. The proposed method (DLS) finds the correct clustering when F-test is used. The Stepwise with LBG-U works fine sometimes whereas the Stepwise GLA gives significantly worse success rate with all data sets. When DBI is used as the criterion, the results are worse with all variants. The relative performance of the different methods is similar to that of the F-test. The CA, on the other hand, consistently fails to find the correct clustering except in the case of easiest test set (S_1). It tends to remove too many clusters no matter of the parameter setup.

Table 4: The number of times the correct clustering was found (among 100 repeats).

	F-test			DBI			
	DLS	Stepwise GLA	Stepwise LBG-U	DLS	Stepwise GLA	Stepwise LBG-U	CA
Data set S_1	100 %	22 %	94 %	100 %	14 %	94 %	70 %
Data set S_2	100 %	30 %	74 %	99 %	19 %	53 %	8 %
Data set S_3	100 %	39 %	80 %	81 %	28 %	72 %	0 %
Data set S_4	100 %	51 %	92 %	29 %	31 %	36 %	0 %
Data set A_1	100 %	8 %	44 %	73 %	4 %	54 %	1 %
Data set A_2	100 %	2 %	2 %	0 %	2 %	8 %	0 %
Data set A_3	100 %	0 %	2 %	0 %	0 %	0 %	0 %

The methods are also compared using the data sets D_2 to D_{15}. The results are summarized in Fig. 12. The main observation is that DLS clearly outperforms the other approaches, of which CA works also reasonably well for these data sets. The DLS results are independent of the dimensionality. The result of the CA, on the other hand, seems to have slight improvement when the dimensionality increases. This is most likely due to the fact that the clusters become more separable and thus, the spatial complexity decreases.

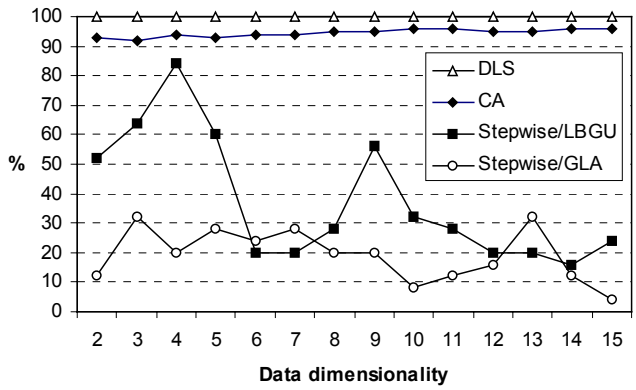


Figure 12. The number of times (%) the correct clustering is found by different algorithms as function of the dimensionality for the data sets D_2 to D_{15}.

5. Conclusions

Dynamic local search algorithm was proposed for finding optimizing both the number and location of the clusters jointly. The algorithm uses a set of basic operations, such as cluster addition, removal and swapping. The algorithm is significantly faster than a simple brute force approach, and it outperforms comparative methods in quality.

References

Bezdek JC, Pal NR, 1998. Some new indexes of cluster validity. *IEEE Transactions on Systems, Man and Cybernetics* 28(3): 302-315.

Davies DL, Bouldin DW 1979. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 1(2): 224-227.

Everitt BS, 1992. *Cluster Analysis*, 3rd Edition. Edward Arnold / Halsted Press, London.

Frigui H, Krishnapuram R, Clustering by Competitive Agglomeration. Pattern Recognition 1997; 30(7): 1109-1119.

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

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

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

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

Gersho A, Gray RM, 1992, *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, Dordrecht.

Hoffmann T, Buhmann J, 1997. Pairwise Data Clustering by Deterministic Annealing. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 19(1), 1-14.

Ito PK, 1980. Robustness of ANOVA and MANOVA Test Procedures. In: Krishnaiah PR (ed). *Handbook of Statistics 1: Analysis of Variance*. North-Holland Publishing Company, pp 199-236.

Jain A.K., Murty M.N., P.J. Flynn, 1999. Data Clustering: A Review. ACM Computing Surveys 31(3), 264-323.

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

Kärkkäinen I, Fränti P, 2000. Minimization of the value of Davies-Bouldin index. In Proceedings of the IASTED International Conference on Signal Processing and Communications (SPC'2000), 2000, pp 426-432.

Kärkkäinen I, Fränti P, 2002. Stepwise clustering algorithm for unknown number of clusters, University of Joensuu, Department of Computer Science, Technical Report, Series A, Report A-2002-5. (submitted)

Linde Y, Buzo A, Gray RM, 1980. An algorithm for vector quantizer design, *IEEE Transactions on Communications* 28(1), 84-95.

Rose K, Gurewitz E, Fox G, 1990. A deterministic annealing approach to clustering. *Pattern Recognition Letters* 11, 589-594.

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

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