Tabu Search Algorithm for Codebook Generation in Vector Quantization

Pasi Fränti, Juha Kivijärvi* and Olli Nevalainen*

Department of Computer Science,
University of Joensuu, P.O. Box 111
FIN-80101 Joensuu, FINLAND
Email: franti@cs.joensuu.fi

*Department of Computer Science
University of Turku
Lemminkäisenkatu 14 A
FIN-20520 Turku, FINLAND

Abstract

A tabu search algorithm is proposed for the codebook generation in vector quantization. The key question is the definition of neighboring solution. Making random modifications to the current solution is not sufficient alone. The proposed algorithm first makes non-local changes to the codebook which is then fine-tuned by the generalized Lloyd algorithm (GLA). For a set of gray scale images, the new algorithm was better than GLA alone, and its results were comparable to simulated annealing. For binary images, the tabu search approach gave the best MSE-values.

Key words: Vector quantization, codebook generation, clustering problem, tabu search, image compression

1. Introduction

In the present paper we study the *codebook generation* in *vector quantization* (*VQ*). The aim is to find a set of representative vectors (called *codevectors*, or *codebook*) for a given *training set* minimizing the average distortion between the training set and the codebook. It is assumed that the vectors are mapped to their nearest representative in the codebook in respect to a *distortion function*.

Iterative algorithms are applicable to the problem. A common property of these algorithms is that they try several possible solutions (codebooks) to the problem and at each step a new solution is generated by making modifications to the current one. For example, the well known *generalized Lloyd algorithm* (GLA) [1, 2] (also referred as *k-means* algorithm due to [3]) starts with an initial solution, which can be chosen arbitrarily. The solution is then improved iteratively using two optimality criteria in turn until a local minimum is reached. This localized search strategy is based on a greedy heuristic where the optimization function is never allowed to increase. Thus, the algorithm will get stuck to the first local minimum, although it may be far from the global optimum.

Tabu search [4] is an alternative approach to the *localized search* described above. It includes the following potential improvements:

- The new solution is chosen among several candidates.
- The search is allowed to make suboptimal moves.
- The use of tabu list prevents local loops.

The use of several candidates directs the search towards the highest improvement in the optimization function value. Making suboptimal moves, on the other hand, allows the search to continue past local minima. Tabu search algorithm also uses so-called *tabu list* 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 codebook generation resembles the *clustering problem*, where the objects of clustering are *K*-dimensional *training vectors* in an *Euclidean space*. The aim of the VQ-algorithm, however, is to create a codebook; the clusters are of secondary importance. Tabu search approach for the clustering problem was proposed in [5]. This algorithm has several drawbacks when applied to the codebook generation in VQ. Some of them originate from the different nature of the source data. The number of the objects, the dimensions of the source data and the number of clusters to be created are all significantly larger in VQ than in most clustering applications.

Here we consider the following training set types: (1) 4×4 pixel blocks taken from gray-scale images (8 bits per pixel), see Fig. 1; (2) the same blocks after quantization into two levels according to the mean value of the block; (3) 4×4 blocks taken from binary images (the eight standard CCITT test images). The training sets are summarized in Table 1. This kind of data are typical in image compression [6, 7, 8]. The number of codevectors to be formed is fixed to 256 in the present study.

The huge size of the search space makes the codebook generation problem less of a combinatorial nature. Therefore an algorithm applicable to this kind of data has to contain more deterministic reasoning to direct the search than randomizing alone. Furthermore, it seems unlikely that the use of tabu list would have a major effect on the algorithm because the generation of new candidates is heavily based on randomizing (also due to the size of the search space). Because of these matters the algorithm proposed in [5] needs major revisions when brought into the context of vector quantization.

In the present paper we propose a new tabu search algorithm for the codebook generation problem with the following ideas:

- Distance weighted algorithm for modifying the partitions.
- Codebook-based problem setup instead of partitioning-based.
- Inclusion of GLA iterations.
- Approximative tabu matching criterion.

These improvements contribute also to the more general clustering problem. The rest of the paper is organised as follows. We start in Section 2 by defining basic concepts of the problem and by giving a brief summary of the previous releated research. The tabu search algorithm and its parameters are discussed in Section 3 and then compared with the existing algorithms in Section 4. Test results appear for both image coding and standard cluster analysis data sets. Finally, conclusions are drawn in Section 5.

Bridge (256×256) *Camera* (256×256) *Couple* (256×256) *Lena* (512×512)

Fig. 1. Training set images.

Table 1. Training sets and their statistics.

Training set	bits	# vectors	# different vectors
Bridge	8	4096	4096
Camera	8	4096	4095
Couple	8	4096	3934
Lena	8	16384	16384
Bridge2	1	4096	2813
Camera2	1	4096	3172
Couple2	1	4096	2119
CCITT	1	2052864	5987

2. Background

Let us consider training vectors in a K-dimensional Euclidean space. Denote the number of vectors in the training set by N. Vector quantization partitions the input space into M non-overlapping regions called *partitions* or *clusters* so that the input space is completely covered. A representative *codevector* is assigned to each partition. The aim of the codebook construction is to minimize the average distance (*distortion*) between the training vectors and their representatives. The question of the proper choice for the training set is not issued here. The motivation is merely to select the best possible codebook for a given training set regardless the way it has been chosen. The distance between two vectors X and Y is defined by:

$$d(X,Y) = \sqrt{\sum_{i=1}^{K} \left(X_i - Y_i\right)^2} \qquad \textbf{(1)}$$

The average distortion of the codebook is calculated by *mean square error* (MSE):

$$MSE = \frac{1}{KN} \sum_{i=1}^{N} d\left(X_i, f\left(X_i\right)\right)^2 \qquad \textbf{(2)}$$

where $f(X)$ is a mapping function from training vector X to its representative codevector. Thus, the mapping function $f(X)$ defines a partitioning of the training set. All test results given in the following will be MSE-values (per pixel) in case of gray-scale images (*Bridge*, *Couple*, *Camera, Lena*). For the binary images (*Bridge2*, *Couple2*, *Camera2*, *CCITT*) the distortion is expressed by the average number of distorted pixels per 4×4-block (varying from 0 to 16). This equals to $K \cdot MSE$.

A solution of the codebook generation problem can be defined by the pair (*partitioning*, *codebook*). A partitioning describes a mapping from the training set to a codebook, and the codebook describes the codevectors. These two depend on each other in an interesting way: if one of them is given, the optimal solution to the other one can be uniquely constructed. This is formalized in the following two optimality criteria [1]:

- *Nearest neighbor condition*: For a given codebook, the optimal partitioning of the training set can be obtained by mapping each training vector to its nearest codevector in the codebook in respect to the distortion function.

- *Centroid condition*: For a given partition, the optimal codevector is the *centroid* (average vector) of the vectors within the partition.

The nearest neighbor condition clearly defines the optimal partitioning. It can be easily constructed by mapping each training vector to the codevector that minimizes (2). This is computationally expensive operation which takes $O(NM)$ time. The centroid calculation, on the other hand, only takes $O(N)$ time.

Generalized Lloyd algorithm (*GLA*) [2] applies the two optimality criteria in turn. In the *partitioning step* the training set is partitioned according to the existing codebook. The optimal partitioning is obtained by mapping each training vector to the nearest codevector as defined in (1). In the *codebook step* a new codebook is constructed by calculating the centroids of the partitions defined in the partitioning step. The two optimality criteria guarantee that the new solution is always equal to or better than the previous one. The process is iterated until no improvement is achieved, or the number of iterations reaches a predefined limit.

GLA uses a greedy heuristic where the optimization function is never allowed to increase. *Simulated annealing* (SA) tries to remove this restriction by a stochastic relaxation technique. At each step random noise is added to codevectors [9], or to training vectors [10] so that the algorithm can jump over local minima. The amount of noise is gradually decreased at each iteration step and eventually, when the noise has been completely eliminated, the algorithm will converge to a local minimum, which hopefully is a good solution to the problem.

3. Tabu search algorithm

Tabu search (TS) is an alternative approach to GLA in the sense that suboptimal moves are allowed whereas GLA proceeds using the nearest neighbor and centroid conditions as optimality criteria. In other words, the new solution does not need to be better than the previous one. Tabu search algorithm also uses tabu list to force the search for expanding into new directions instead of local minima. Tabu list includes T previously visited solutions. The new solution in each iteration round is always the best *non-tabu* candidate solution. In the case where all candidates are in tabu list the current solution does not change but a new set of S candidates will be generated.

A tabu search algorithm is formalized in Fig. 2. The initial solution is chosen randomly. We also studied the algorithm starting with codebooks generated by GLA as an initial solution. The tabu search was seldom able to improve them, thus the random initialization seems to be a better choice. The distortion function e is here due to (2).

- Generate an initial solution C_{INIT} using any existing algorithm.
- Set $C_{CURR} = C_{BEST} = C_{INIT}$.
- Iterate the following I times
  - Generate S candidate solutions C_i by making modifications to the current solution C_{CURR}.
  - Calculate the distortion values of the candidate solutions C_i.
  - If the best candidate is better than C_{BEST} then set it as C_{CURR} otherwise set the best non-tabu candidate as C_{CURR}.
  - If tabu list is full, remove the oldest solution.
  - Insert C_{CURR} to tabu list.
  - If $e(C_{CURR}) < e(C_{BEST})$ then set $C_{BEST} = C_{CURR}$.
- Output C_{BEST}.

Fig. 2. Sketch of a tabu search algorithm.

In the tabu search algorithm for the clustering problem in [5] the initial solution is chosen randomly. New candidate solutions are generated by modifying the partitions in a random fashion; each training vector is moved from its original partition to another randomly chosen partition with a probability P. In principle, this algorithm can also be applied to the codebook generation problem if the centroid condition is applied for selecting the codevectors. However, the algorithm is impractical for the codebook generation due to the nature of the data in vector quantization. A typical VQ data set has the following properties:

- A large number of objects (training vectors) in the source data (N).
- High dimensionality of the data set (K).
- Relatively large number of clusters to be formed (M).

These facts make the search space significantly larger than in typical clustering applications and an algorithm applicable to this kind of data has to contain more

deterministic reasoning to direct the search than randomizing alone. These matters will be considered in the following sections.

3.1 Representation of a solution

The two optimality criteria (nearest neighbor condition and centroid condition) imply two alternative approaches to codebook generation algorithms:

- codebook-based (CB)
- partitioning-based (PB)

In the *codebook-based* variant a solution is described by a codebook. The codevectors are represented by an M-length array of K-dimensional vectors, see Fig. 3. This is a natural way to describe the problem in the context of VQ. After all, the aim of the algorithm is to create a codebook; the partitions are of secondary importance. The partitioning (the mapping function), however, is needed when evaluating the distortion values of the solutions and it is calculated using the nearest neighbor condition.

In the *partitioning-based* variant a solution is described by a partitioning. It is expressed as an N-length array of integers in the range $[1..M]$ defining mapping from the training set to the codebook, see Fig. 3. The actual codebook is calculated from the partitioning using the centroid condition. The partitioning-based variant is commonly used in the clustering algorithms because the aim is to cluster the data with no regard to the representatives of the clusters (such as codevectors in VQ). It was also adopted in [5].

The search space is of the size $M^N=2^{N \log M}$ in the PB-variant and 2^{bKM} in the CB-variant. The former is the number of ways N training vectors can be clustered into at most M distinctive clusters. The latter originates from the number of ways a codebook of size M consisting of K-dimensional vectors can be constructed (b is the number of bits per pixel in a vector element). With the typical parameters in our test data (b=8, K=16, N=4096, M=256) the size of the search space is thus order of 10^{9864}. This is significantly larger than the size of the search space used in [5]: 10^{12} with parameters N=40, M=2.

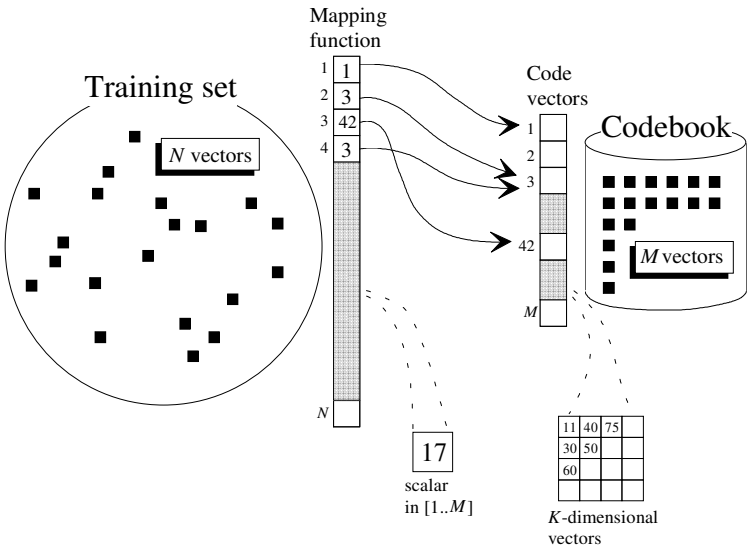

Fig. 3. Representation of a training set and a codebook.

3.2 Selection of the parameter values

The four main parameters of the tabu search algorithm are: probability threshold for the changes, the number of iterations, the number of candidate solutions and the size of tabu list.

The probability threshold (P) defines the amount of changes generated to the current solution in order to create new candidates. The best choice of P depends on the problem size, i.e. the training set size, codebook size and dimensionality of the source data. Our experiments indicate that P=5 % (used in [5]) is adequate for very small test data (N=500, M=16, K=2), see Fig. 2. This is, however, too large for our applications. A reasonable value for larger training sets (Table 1) seems to be as low as P=0.5 %.

The number of iterations (I) and the number of candidates generated in each iteration round (S) together induce the total number of trial solutions ($I \cdot S$). By modifying the ratio of these two parameters the nature of the search can be adjusted either towards depth-first search (increasing I and decreasing S) or towards breadth-first search (vice versa). Neither of these two approaches had the edge over the other in our experiments; the quality of the final solution seems to be relatively independent of I and S as long as $I \cdot S$ remains constant. Here we set the values I=500 and S=20.

In general, a small tabu list size (T) is usually sufficient. After all, the size of tabu list is limited by the number of iteration rounds because only one solution per iteration round is included in tabu list. Here we use T=20.

3.3 Definition of neighboring solution

The key question in the tabu search is the definition of the neighborhood of a solution, i.e. the way new candidates are generated. In the PB-variant the simplest method is to make random changes to the current solution, as proposed in [5]. A drawback of random changes is that a great number of useless solutions will be generated. In fact, the main effect of transferring a training vector to another randomly chosen cluster is that the centroid of the new cluster will be shifted towards the transferred vector. In general, the resulting codevectors are arbitrarily located in respect to each other, and they tend to concentrate around the centroid of the training set leaving the border areas sparsely occupied.

The search can be made more efficient if the new cluster for a training vector is chosen considering the distances to different clusters: the closer the cluster, the more likely it will be chosen. Here we apply the weighting function:

$$w_i = \frac{1}{d_i} \tag{3}$$

where d_i is the distance between the processed training vector and the corresponding codevector of the candidate cluster. The probability of selecting the *i*th cluster to be chosen is:

$$p_i = \frac{w_i}{\sum_j w_j} \tag{4}$$

In the CB-variant the randomizing can be implemented by the following methods:

- *Adding noise to codevectors*
- *Swapping codevector*

The first method (adopted from simulated annealing) has the problem that the changes are local. In the swapping method, a codevector is replaced (with probability *P*) by a randomly chosen training vector. The weighting function (4) could also be applied, but we prefer equal weighting in order to permit more radical, non-local changes in the codebook.

Both the CB-variant and the PB-variant (with a weighting function) outperform the original randomizing algorithm. Unfortunately, the results are still far from good, see Fig. 4 for an artificial trainiong set in two-dimensional space. Thus, additional refinements are needed in the tabu search algorithm.

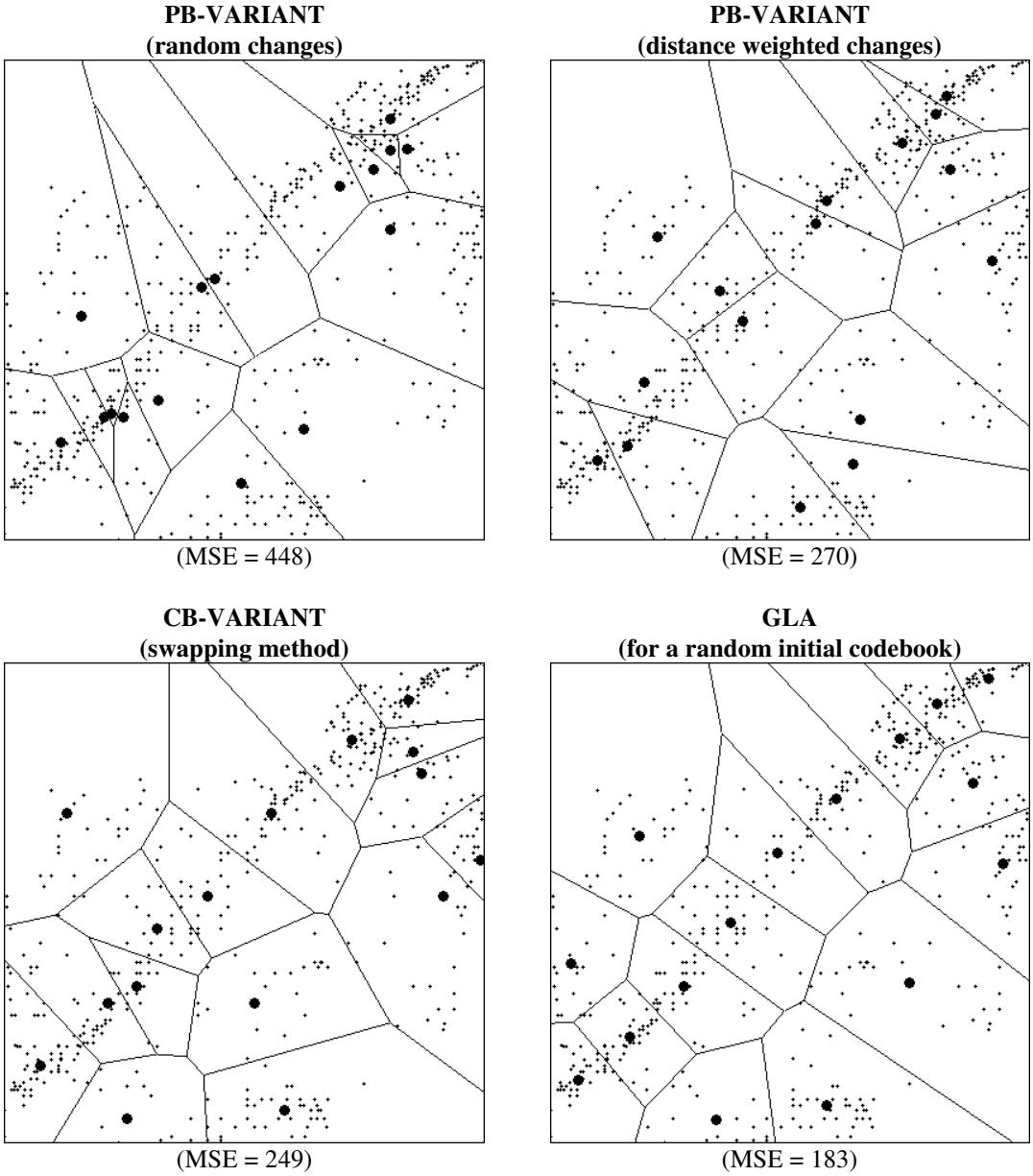

Fig. 4. Codebooks generated by different variants of the tabu search algorithm and by GLA. The small dots are the training vectors and the large ones are the codevectors. The partition boundaries (*Voronoi diagram*) have been calculated using the nearest neighbor condition.

3.4 Inclusion of GLA iterations

The most effective modification proposed here is the inclusion of GLA iterations. This means that each candidate solution is improved by feeding it to GLA which outputs the nearest local minimum. In a sense, the tabu search algorithm first makes non-local changes to the codebook which is then fine-tuned by GLA.

This modification has at least three potential improvements: (1) the search space is greatly reduced; (2) only reasonable solutions will ever be considered; (3) tabu list may become effective after all. The first two hypotheses are confirmed by the results shown in Fig. 5. GLA iterations give better results much faster. In fact, even the worst candidate is better than any of the candidates without GLA. The third hypothesis will be studied in the next section.

The main drawback of GLA is that the running time is (approximately) multiplied by the number of GLA iterations applied for each candidate solution. Fortunately, it is not necessary to iterate each candidate codebook until it reaches a local minimum. Instead, the number of GLA iterations can be limited to a very few, see Table 2. In the further tests the number of GLA iterations will be fixed to 2. Moreover, additional experiments revealed that the application of GLA outweighs the drawbacks of PB-variants and their results become competitive as well.

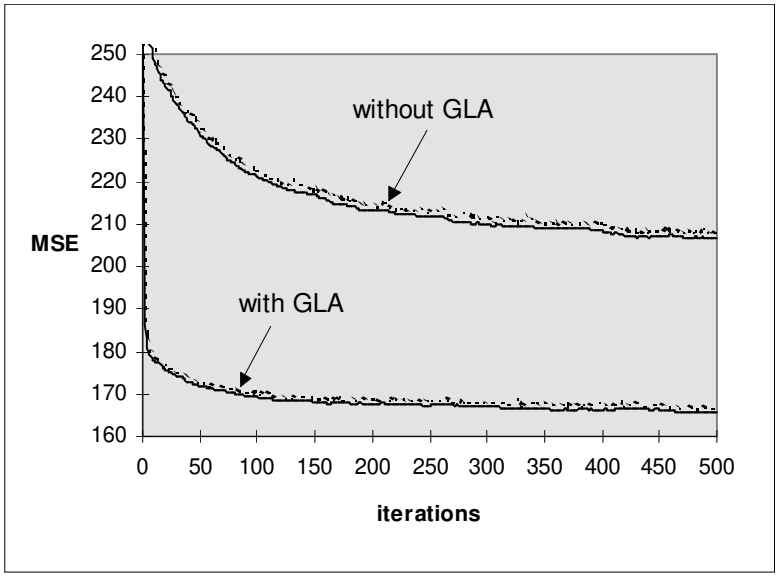

Fig. 5. Distortion of best (solid lines) and worst candidates (broken lines) vs. the number of iterations (for *Bridge*). CB-variant with swapping is applied.

Table 2. Distortion of codebooks (*Bridge*) with various number of GLA iterations. The results are averages of five test runs.

GLA iterations	Distortion (MSE)	Running time (hours:minutes)
0	206.46	7:08
1	167.20	12:23
2	165.53	17:56
4	164.91	27:29
8	164.94	38:00
∞	164.87	66:31

3.5 The use of tabu list

It is not evident that tabu list has a significant effect on the algorithm's overall performance. In fact, it is not even certain that tabu list is needed at all. Since the generation of new candidates is based on randomizing (and also due to the size of the search space) it is unlikely that the search will get stuck into a local minimum anyway. Experiments indeed show that tabu list is not utilized in the tabu search variants with standard parameter settings. The application of two GLA iterations does not remarkably change the situation for gray-scale images. For binary images, on the other hand, the search space is reduced enough so that tabu list becomes effective.

I order to make a stronger influence on the search te use of tabu list is intensified as follows. A candidate solution is declared tabu if an identical solution is found in tabu list. Instead of exact matches, we introduce an *approximative tabu matching* which allows a small error marginal, i.e. a candidate solution is declared tabu if it is "close enough" to any solution in tabu list. This inaccuracy improves the use of tabu list in cases where two solutions are effectively the same but only insignificant differences appear in the exact figures. The distance between two codebooks C_1 and C_2 is defined by the average distortion (2) of mappings $C_1 \rightarrow C_2$ and $C_2 \rightarrow C_1$. The best solution of every second iteration round (on average) is now declared tabu, with a threshold value of 0.5 (MSE units) for gray-scale images, and 0.005 for binary images.

In general, the approximate tabu matching had only marginal effect on the MSE-values. For binary images it slightly improved the results, but for gray-scale images the results got slightly worse. In overall, tabu list is of little use in the proposed algorithm. On average, the results are equally good with and without tabu list.

4. Comparison with existing algorithms

The computational complexity of different tabu search variants is summarized in Table 3. For each candidate solution we have to calculate both the partitioning (partitioning step) and the codevectors (codebook step) in order to obtain the fitness value of a solution. The calculation of optimal partitioning and optimal codebook take an $O(NM)$ and an $O(N)$ times respectively, see Section 2.

In the CB-variant, the codebook step is replaced by making random changes to the codebook. This takes only $O(M)$ time. Unfortunately the overall time complexity remains still $O(NM)$ due to the partitioning. The PB-variant with random changes performs the partitioning in $O(N)$ time which is also the overall time complexity of the algorithm. In the other PB-variant (with distance weighted changes) we have to calculate for each training vector the distances to all codevectors in order to obtain the weights. Fortunately only $P \cdot N$ training vectors will be processed (P=0.5 %) keeping the actual running time reasonable.

If GLA iterations are included, the time complexity of each variant is increased by $O(GNM)$, where G is the number of GLA iterations applied. As proposed in Section 3.4, the number of iterations can be limited to only two without causing any dramatic effects on the quality of the final solution.

To sum up, the time complexity of the CB-variant is of the same order with and without GLA iterations (assuming G=2). For the PB-variants, this is not the case. With random changes the algorithm is clearly faster, but the experiments have shown that this algorithm is practically useless without GLA iterations. The PB-variant with distance weighted changes is also much faster without GLA. However, it is able to produce reasonable results only for smaller problem sizes. Thus, a potential tabu search algorithm (TS) must contain GLA iterations if applied to the training sets of Table 1.

Table 3. The time complexity of the tabu search variants for one candidate solution. (*P=0.005)

	PB-variants:		**CB-variant:**
	random	**distance weighted**	
Partitioning step	$O(PN)$*	$O(PNM)$*	$O(NM)$
Codebook step	$O(N)$	$O(N)$	$O(PM)$*
GLA iterations	$O(GNM)$	$O(GNM)$	$O(GNM)$
Total - with GLA	$O(GNM)$	$O(GNM)$	$O(GNM)$
Total - w/o GLA	$O(N)$	$O(PNM)$*	$O(NM)$

So far we have considered only the time for a single candidate solution. There are, however, $I \cdot S$ solutions (I=500, S=20) to be generated in overall. This makes the above variants much slower than the other algorithms in our comparison. The performance of the tabu search algorithm is next compared with the following algorithms:

- GLA = Generalized Lloyd algorithm [2].
- SA = Simulated annealing [9].
- PNN = Pairwise nearest neighbor algorithm [11].

Pairwise nearest neighbor (*PNN*) algorithm starts by initializing a codebook of size N where the codebook contains all the training vectors. Two codevectors (partitions) are combined and replaced by the centroid of the training vectors of the combined partition. The vectors to be combined are the ones that increase the distortion least. This step is repeated until the size of the codebook has decreased to M. There are two versions of the PNN algorithm: the $O(N^3)$-time algorithm, and the *fast PNN* which takes $O(N \cdot \log N)$ time. We use the former one.

In the simulated annealing a random noise vector is applied with a uniform probability distribution function in the range $[-t, t]$. A logarithmic temperature schedule decreases the temperature t by 1 % after each iteration step. The initial temperature t_0 is 50 for gray-scale images, and 1 for binary images. The inclusion of noise is stopped when t falls below 0.5; smaller temperature values have no effect because the pixel values are always rounded to the nearest integer. The time complexity of one SA iteration, $O(NM)$, is equal to that of GLA but the total number of iterations is greater for SA. Both algorithms use a random codebook as a starting point.

The quality of codebooks generated by various algorithms are shown in Table 4. The two TS algorithms are the PB-variant with distance weighted changes (TS-PB), and the CB-variant with swapping method (TS-CB). Both of them include two GLA iterations. The number of iterations was 500. Both TS variants were always better than pure GLA for the gray scale images, and comparable to SA and PNN. From these, SA gives slightly lower MSE-values than the tabu search. In the case of binary images the best methods seem to be the TS variants. The total running times (min:sec) of the algorithms (GLA, SA, PNN, TS-PB, TS-CB,) were (0:38, 13:03, 67:00, 739:00, 1076:00) on a Pentium/90 computer for *Bridge*.

Table 4. Performance comparisons of various algorithms. The results are averages of five test runs.

	TS-PB	**TS-CB**	**GLA**	**SA**	**PNN**
Bridge	165.85	164.23	179.68	**162.45**	169.15
Camera	73.31	71.87	122.61	**70.65**	70.90
Couple	26.51	25.53	40.36	**25.15**	25.91
Lena	56.64	54.80	59.73	**54.40**	56.44
Bridge2	1.28	**1.27**	1.48	1.52	1.33
Camera2	1.56	**1.53**	1.76	1.78	1.61
Couple2	0.90	**0.89**	1.06	1.12	0.95
CCITT	**0.17**	**0.17**	0.24	0.40	0.18

The efficiency of the algorithms is next compared by calculating the distortion of the best codebook as a function of running time spent. GLA and SA were repeated, each time starting from a new random codebook. The number of runs performed during an

18 hours time period was 1700 for GLA and 80 for SA. The variations in their results, however, are too small to produce any remarkable improvement when the algorithms were repeated, see Fig. 6. The corresponding number of iterations for TS-CB and TS-PB were 500 and 730. It is possible that TS-CB might reach the same level as SA (and probably beyond) if considerably more computing resources were used.

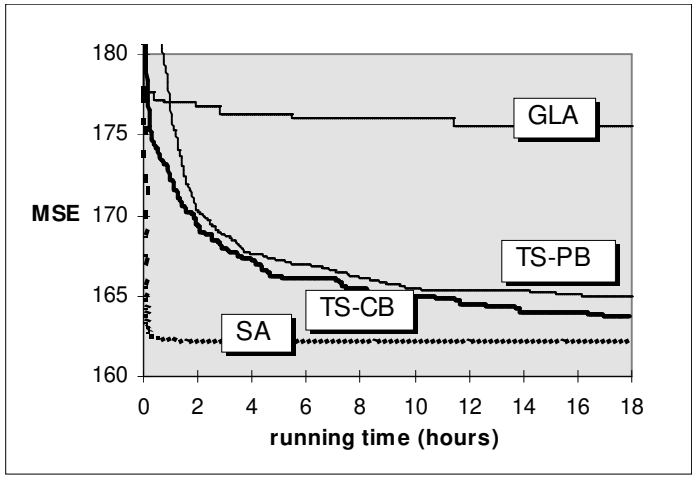

Fig. 6. The distortion of the best codebook found vs. running time (*Bridge*)

Finally, we tested the algorithms for the standard clustering test problems SS1 and SS2 of [12], see pages 91-92 and 103-104 correspondingly. The data sets contain 89 postal zones in Bavaria (Germany) and their attributes:

- SS1: surface area (km^2), population and density of population.
- SS2: the number of self-employed people, civil servants, clerks and manual workers.

In order to cluster the objects according to these attributes their values must be scaled. We used the following scaling:

$$x' = \frac{x - \min(x_i)}{\max(x_i) - \min(x_i)} \qquad \textbf{(5)}$$

where x is the data value before and x' after scaling. The number of clusters varies from 2 to 32. Thus, the problem size becomes (N=89, M=2..32, K=3) for SS1 and (N=89, M=2..32, K=4) for SS2. The parameter setups were the same as for the gray scale images. Both GLA and SA were repeated 100 times and the best result was observed for each cluster size. This time the repetition improved significantly the results of GLA. SA, on the other hand, is more robust and its average result is not far from the best found. The best results are summarized in Fig. 7.

For all cluster sizes (except M=32) the results of TS-CB and SA are precisely equal. This implies that both algorithms (sometimes also GLA) have possibly found the global optimum. In addition to this, the results for the largest cluster size differ only in

the second decimal. In general, the conclusions made for gray-scale images hold rather well for SS1 and SS2, too. For example, the application of GLA iterations is vital and the use of tabu list is of little help. Moreover, the original partitioning-based variant (without GLA) with random modifications is inefficient.

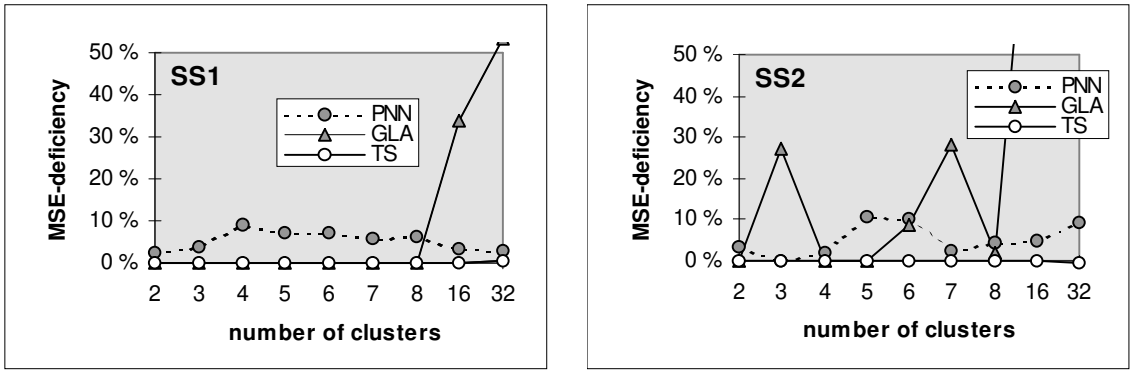

Fig. 7. Performance comparison for standard data sets SS1 and SS2. The values are the relative differences of the MSE-values when compared to the results of SA.

5. Conclusions

A tabu search algorithm for the codebook generation problem in VQ was proposed. The key question is the definition of neighboring solution. Two different approaches were studied: a codebook- and a partitioning-based. The most effective modification, however, was the inclusion of GLA iterations. In a sense, the tabu search algorithm first makes non-local changes to the codebook which is then fine-tuned by GLA. The inclusion of GLA iterations reduces the search space so that only reasonable solutions will ever be considered. The use of tabu list was of secondary importance.

In general, the results of the proposed algorithm were promising. The MSE-values were 20 % lower than those of GLA on average. For gray scale images, simulated annealing was slightly better (2-5 %), but in the case of binary images the tabu search gave clearly better. The main drawback of the algorithm is the high running time. If time is a critical factor, GLA and other faster methods (fast PNN for instance) are more attractive alternatives.

It was also noted that all tested algorithms were relatively robust to the random initialization. In case of image data, none of the algorithms was able to make remarkable improvement by repeating the search from a new starting point. This gives a slight edge to the tabu search approach since it did not converg. Thus, better results can be reached by increasing the computing resources.

References

[1] A. Gersho and R.M. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic Publishers (1992).

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

[3] 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).

[4] F. Glover and M. Laguna, Tabu search, in *Modern Heuristic Techniques for Combinatorial Problems* (ed. R.C. Reeves), 70-150, McGraw-Hill, Berkshire (1995).

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

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

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

[8] X. Wu and Y. Fang, Fast bintree-structured image coder for high quality subjective quality, *Proceedings Data Compression Conference*, Snowbird, Utah, 284-293 (1994).

[9] K. Zeger and A. Gersho, Stochastic relaxation algorithm for improved vector quantiser Design, *Electronics Letters* **25**, 896-898, (1989).

[10] J. Vaisey and A. Gersho, Simulated annealing and codebook design, *Proc. ICASSP*, 1176-1179, (1988).

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

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