Pattern Analysis & Applications (2000)3:348–357

Clustering by Adaptive Local Search with Multiple Search Operators

M. Gyllenberg[1,2], T. Koski[1,3], T. Lund[1] and O. Nevalainen[1,2]

[1]*Department of Mathematical Sciences, University of Turku, Finland;* [2]*Turku Center for Compute Science (TUCS), University of Turku, Finland;* [3]*Department of Mathematics, Royal Institute of Technology, Stockholm, Sweden*

Abstract: Local Search (LS) has proven to be an efficient optimisation technique in clustering applications and in the minimisation of stochastic complexity of a data set. In the present paper, we propose two ways of organising LS in these contexts, the Multi-operator Local Search (MOLS) and the Adaptive Multi-Operator Local Search (AMOLS), and compare their performance to single operator (random swap) LS method and repeated GLA (Generalised Lloyd Algorithm). Both of the proposed methods use several different LS operators to solve the problem. MOLS applies the operators cyclically in the same order, whereas AMOLS adapts itself to favour the operators which manage to improve the result more frequently. We use a large database of binary vectors representing strains of bacteria belonging to the family *Enterobacteriaceae* and a binary image as our test materials. The new techniques turn out to be very promising in these tests.

Keywords: Adaptation; Clustering; GLA; Local Search; Stochastic complexity

1. INTRODUCTION

Local Search (LS) has proven to be an efficient optimisation technique in clustering applications [1]. In the present paper, we propose two ways of organising LS, the Multi-Operator Local Search (MOLS) and the Adaptive Multi-Operator Local Search (AMOLS). Both techniques use several different LS operators to solve the problem.

In recent years, there has been great activity in the research of efficient solution methods for hard combinatorial optimisation problems (see Reeves [2] for a survey of methods). The exact solution of relatively small problem instances can sometimes be found by modern optimisation packages like CPLEX, or by tailored branch-and-bound methods due to the remarkable progress of these systems. There are still numerous problems (like clustering), where the exact solution is ruled out, at least for instances of practical relevance, and one has to resort to approximative search methods. These can be roughly categorised as construction heuristics, descent methods (including local search heuristics), genetic algorithms, tabu search and simulated annealing. The above classification is not strict, and hybrid methods are becoming more and more popular. In fact, the genetic algorithms belong to a larger set of evolutionary computation methods (containing genetic algorithms, genetic programming and evolutionary strategies) [3].

Local Search (LS), *simulated annealing* [4] and *Tabu Search* (TS) [5] type algorithms modify the current candidate solution by a search (modification or move) operator which looks for a better solution in a neighbourhood of the current solution. These algorithms are usually written to use only one modification operator [1,6].

Here, we investigate alternative methods of cycling a pool of various different LS operators (MOLS), and adaptively drawing random methods from the pool of available operators (AMOLS). Different LS operators have their advantages and drawbacks; a globally best optimiser does not exist. Here we refer to the 'no-free-lunch' theorem [7]. Usually, a deterministic approach leads to losses in robustness of the algorithm, i.e. the efficiency depends strongly upon the problem instance and the initial solution. On the other hand, most of the search steps in LS do not produce any enhancement in the value of the cost function. The use of multiple search operators of LS can be fruitful because (1)

Received: 8 November 1999
Received in revised form: 15 February 2000
Accepted: 11 September 2000

despite the determinism, several directions in the search space are examined, and (2) some operators still can exploit randomness, even if we have reached a local minimum point.

The idea of multiple search operators has previously been successfully applied in the design of *Adaptive Genetic Algorithms* (AGA). In this context, various different crossover techniques and different mutation probabilities are used in an adaptive way [8,9]. This gives us a robust technique which adapts itself to use those search operators which are best suited for the problem instance at hand.

In the present paper, we apply the above idea in a different way. While the adaptation in AGA is made for a population of solutions, in our LS-implementation we consider only one solution. This solution will be greedily improved by several different LS-operators. The benefit of this approach is the simplicity of the implementation. On the other hand, we loose the potential power of parallel search of GA.

The rest of the paper is organised as follows. Background theory is presented in Section 2. In Section 3 we present the different LS operators and MOLS algorithm in detail. Then we extend MOLS to its adaptive version (AMOLS). We discuss the data and the test procedures in Section 4, and in Section 5 we present the results. The paper ends with a short discussion.

2. PROBLEM FORMULATION

Let $\mathbf{x}^{(l)}$ be a d-dimensional binary vector, i.e. a vector with components $x_1, x_2, \ldots, x_d$, where $x_i = 0$ or 1 for $i = 1, \ldots, d$. Suppose that we have a set of t binary vectors $\chi = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(t)}\}$ to be clustered (or placed, partitioned) into k disjoint clusters $C_1, C_2, \ldots, C_k$. We define the centroid of the cluster C_j as $\hat{\theta}_j = (\hat{\theta}_{1,j}, \ldots, \hat{\theta}_{dj})$, where $\hat{0}_{ij} = t_{ij}/t_j$. Here t_j stands for the number of vectors in the cluster C_j, and t_{ij} the number of vectors in cluster j with the ith component being one. For each vector $\mathbf{x}^{(l)}$, the distortion from the centre of its cluster is defined by the squared l_2-distance (Minkowsky 2-metric)

$$\|\mathbf{x}^{(l)} - \hat{\theta}_j\|_2^2 = \sum_{i=1}^{d} (x_i^{(l)} - \hat{\theta}_{ij}^{(l)})^2 \tag{1}$$

It is common to assign a vector to the cluster for which Eq. (1) is minimised. Another method of clustering is based on the minimisation of *stochastic complexity* [10,11]. In this case, if we define a set of cluster weights $\hat{\Lambda} = (\hat{\lambda}_1, \ldots, \hat{\lambda}_k)$, where $\hat{\lambda}_j = t_j/t$ and we assign $\mathbf{x}^{(l)}$ to the cluster for which the codelength

$$\begin{aligned} &D(\mathbf{x}^{(l)}, \hat{\theta}_j) = \\ &-\sum_{i=1}^{d} ((1 - x_i^{(l)}) \log_2 (1 - \hat{\theta}_{ij}) \\ &+ x_i^{(l)} \log_2 \hat{\theta}_{ij}) - \log_2 \hat{\lambda}_j \end{aligned} \tag{2}$$

assumes its least value.

Following Gyllenberg et al [10], we define the stochastic complexity

$$\begin{aligned} SC = &\log_2 \left(\frac{t!}{t_1! \ldots t_k!} \right) \\ &+ \log_2 \binom{t + k - 1}{t} \\ &+ \sum_{j=1}^{k} \sum_{i=1}^{d} \log_2 \left(\frac{(t_j + 1)!}{t_{ij}!(t_j - t_{ij})!} \right) \end{aligned} \tag{3}$$

The first two terms on the right-hand side of Eq. (3) can be interpreted as the length of the prefix, i.e. the complexity of the clustering structure (cluster representatives). The last term of Eq. (3) is the complexity of describing the vectors with respect to the clustering (3). The minimisation of stochastic complexity as a function of k is an answer to the question of finding the optimal number of clusters, and is related to the principle of the minimum description/message length (MDL/MML) [10,12].

We consider two different ways of constructing a clustering, viz. minimisation of Eq. (1) over all sample vectors for $\{C_1, \ldots, C_k\}$, and minimisation of Eq. (3). The first alternative produces ball-like clusters, whereas the second method minimises the total complexity of the clustering.

For some operators used by the clustering algorithms, we need a simple measure for the distortion or incoherence of a single cluster. We chose to define the distortion of the cluster C_j as the average Hamming-distance to the rounded centroid

$$\mathbf{a}_j = (a_{1j}, \ldots, a_{dj}), \quad a_{ij} = \lfloor \hat{\theta}_{ij} + 1/2 \rfloor \tag{4}$$

The Hamming-distance of the vector $\mathbf{x}^{(l)}$ to the rounded centroid of C_j is defined by

$$\rho(\mathbf{x}^{(l)}, C_j) = \sum_{i=1}^{d} |x_i^{(l)} - a_{ij}| \tag{5}$$

and thus the distortion of a cluster C_j is the average distance over the vectors $\mathbf{x}^{(l)}$ assigned to the cluster, i.e. the distortion of C_j is given by

$$R_j = \frac{1}{t_j} \sum_{x \in C_j} \rho(\mathbf{x}, C_j) \tag{6}$$

3. ALGORITHMS

3.1. Search Operators

Let us suppose that we are given a set χ of t d-dimensional, binary data vectors $\chi = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(t)}\}$ and an initial clustering as defined by the k cluster centroids $(\hat{\theta}_{1j}, \ldots, \hat{\theta}_{dj})$. We next recall six local search operators which can be used for improving the clustering. These operators are known from the context of vector quantisation [13], but we have modified them to suit the minimisation of stochastic complexity.

SJ1: Split-and-join (variant 1)
Often it turns out to be profitable to code two very close clusters as a single one, and to use the centroid thus obtained to code a cluster at some other part of the vector

space. The SJ1 operator takes this into account by joining the two closest clusters according to the l_2-distance of centroids, and by splitting the most incoherent cluster according to the internal distortion (average Hamming-distance to cluster representative (6)).

RWO: Replace-worst
As noted above, a cluster with a high internal distortion is likely to be coded inefficiently when measured by stochastic complexity. The *replace-worst* heuristic draws a new random centroid from the input data set for the most incoherent cluster. Incoherence is measured by the internal distortion (6). Application of GLA after this search operator will re-map the vectors. Note that the random choice among the input data means that the new centroid favours the more dense regions of the d-dimensional space.

RSA: Replace-smallest
Small clusters are likely to be coded inefficiently, because a code word consumed to the coding of a small cluster could perhaps be used more efficiently for something else. Small clusters also emerge as a side-effect of other search operators. One possibility to fix this is to draw a new random centroid for the smallest cluster. Again, an application of GLA after this search operator re-maps the vectors.

RSW: Random-swap
Global and random changes of a cluster centroid may help the algorithm escape from a local minimum. The operator selects a random cluster and draws a new random centroid from the input data set for the chosen cluster. It may be that no other method is applicable any longer, but the random-swap still helps us to proceed towards a more promising region of the search space.

SJ2: Split-and-join (variant 2)
This operation joins the smallest cluster and its closest neighbour. The closeness of the clusters is measured by the l_2-distance (1) between the cluster centroids. After joining, the operator splits the most incoherent cluster according to the internal distortion (6).

CMO: Class-move
As noted for the random-swap operator, it is necessary to introduce randomness in the search. On the other hand, the solution process may already be on the right track towards an advantageous region of the search space, so that it is only profitable to make minor changes. This is pursued in the *class-move* operator, which is the same as random-swap, the only difference being that the new centroid is drawn randomly from the same cluster which was chosen to be replaced. This operator moves the cluster into a random direction in the search space without destroying it completely.

3.2. Generalised Lloyd Algorithm

The GLA method [14,15] starts with an initial solution which can be chosen arbitrarily (it can also be an intermediate result of the search algorithm). The solution is then improved iteratively in a loop (Lloyd-iteration), where each item $\mathbf{x}^{(i)}$ in χ is checked against each of the clusters $\{C_1, \ldots, C_k\}$, and then $\mathbf{x}^{(i)}$ is assigned to the closest cluster. After all of the vectors have been assigned, the centroids and cost function are recalculated and the same process is repeated.

3.3. Multi-Operator LS

We first start by giving a very primitive organisation to the multi-operator search in this section: the MOLS algorithm applies a sequence of six LS operators (SJ1, RWO, RSA, RSW, SJ2 and CMO) iteratively. This simple algorithm thus applies the operators in a round-robin fashion. The initial classification in Step 1 is generated by taking k randomly chosen data vectors as cluster centroids, and by assigning the data vectors to the nearest clusters [16]. This approach is considered to be weaker than the widely used McQueen's method [17,18], but on the other hand, the final result of the LS does not depend strongly upon the quality of the initial solution, and the method of selecting the nearest cluster is fast.

Algorithm MOLS

Step 1. Draw k random initial centroids from the input data set;

Step 2. Run two iterations of GLA [15] with the l_2-metric. Let the result of the clustering be P_{best}, and calculate $\text{SC}(P_{\text{best}})$ by formula (3);

Step 3. Let the initial search operator be SJ1;

Step 4. Iterate the steps 4.1 to 4.5 *max-iter* times:

- *Step 4.1.* Apply the current search operator to P_{best} and let the solution be P_{mod};
- *Step 4.2.* Apply two iterations of GLA with the l_2-metric to the solution P_{mod};
- *Step 4.3.* Calculate $\text{SC}(P_{\text{mod}})$;
- *Step 4.4.* If $\text{SC}(P_{\text{mod}}) < \text{SC}(P_{\text{best}})$, then let the $P_{\text{best}} := P_{\text{mod}}$ and $\text{SC}(P_{\text{best}}) := \text{SC}(P_{\text{mod}})$;
- *Step 4.5.* Change the search operator to the next in the list (SJ1, RWO, RSA, RSW, SJ2, CMO) cyclically;

Step 5. Calculate $\hat{\Lambda}$;

Step 6. Apply GLA to minimise D given by Eq. (2), P_{best} as the initial solution, until the result does not change.

At Step 1 we select the k initial centroids among the t data vectors such that all of these are distinct. The algorithm then improves the initial solution by the GLA [15], also known as k-means and GBL (Step 2). After this, the main loop (Step 4) applies the various local search operators and GLA, moving in the direction of better solutions.

We determine the closeness with the l_2-metric in the main loop (Step 4) and with formula (2) in Step 6 of the algorithm, because we are minimising stochastic complexity. The number of GLA iterations is restricted to two in Step 4. In the final step, GLA is iterated until there is no change in the optimisation criterion.

3.4. Adaptive Local Search

Cyclical application of the LS operators in MOLS is, in the long run, very similar to a random choice of LS operators with uniform distribution. This observation helps us to enhance the multi-operator search. Depending upon the data, the initial values of the centroids and the state of the search, different search operators work better than others. On the other hand, when a certain operator is used in succession, its power may become exhausted. One way to overcome this shortcoming is to make the algorithm *adaptive*. This means that when one operator turns out to be successful (the application of the operator leads to a better value of the function representing the optimisation criterion, here stochastic complexity), the algorithm should use it more frequently. When doing this, we should still keep the option of switching to another search operator at a later stage of the solution process. Next, we propose a way of accomplishing this kind of strategy.

Let the initial distribution of the probabilities of using the different LS operators be

$$p^{(0)}(y) = \frac{1}{6}, \quad y = 1, \ldots, 6$$

Here y is an operator from the set {SJ1, RWO, RSA, RSW, SJ2, CMO}. This means that each operator is initially equally probable to be used. We define the indicator function

$$f^{(s)}(y) = \begin{cases} 1, & \text{if operation } y \text{ was successful (decreased Eq. (3)) at the iteration } s \\ 0, & \text{otherwise} \end{cases}$$

Let $n_y^{(s)}$ be the number of successes for operator y from the beginning up to iteration s. Initially, $n_y^{(0)} = 0$ for all y. In every iteration, the success counts are updated by

$$n_y^{(s+1)} = n_y^{(s)} + f^{(s+1)}(y)$$

and we denote the sum of successes by $n^{(s)} = \Sigma_{y=1}^{6} n_y^{(s)}$. Then we update the probabilities for using different operators by

$$p^{(s+1)}(y) = \begin{cases} p^{(s)}(y) & \text{if } f^{(s+1)}(y) = 0 \text{ for all } y \\ \dfrac{n_y^{(s)} + f^{(s+1)}(y) + \alpha}{n^{(s)} + 1 + 6\alpha} & \text{otherwise} \end{cases}$$

Because $f^{(t+1)}(y) = 1$ is only true for one y (= y^*),

$$\sum_{y=1}^{6} p^{(s+1)}(y) = \frac{n_{y^*}^{(s)} + 1 + \alpha + n^{(s)} - n_{y^*}^{(s)} + 5\alpha}{n^{(s)} + 1 + 6\alpha} = 1 \quad (7)$$

In this model, the parameter α controls the weight of the underlying uniform distribution. If α is small, adaptation happens faster. On the other hand, α should not be too small, because this implies that no $p^{(s)}(y)$ ever becomes zero. There is one drawback in this simple model; it has a long memory. Next we enhance this model to use a shorter memory, so that in a situation where a previously successful operator becomes inefficient, the model can adapt to the situation. We define a weight $w_y^{(s)}$ for each operator; initially, take

$$w_y^{(0)} = 0, \text{ for all } y$$

and denote the sum of the weights by $w^{(s)} = \Sigma_{y=1}^{6} w_y^{(s)}$.

The update values of the probabilities for using different operators is now given by

$$p^{(s+1)}(y) = \begin{cases} p^{(s)}(y) & \text{if } f^{(s+1)}(y) = 0 \text{ for all } y \\ \dfrac{w_y^{(s)} + f^{(s+1)}(y) + \alpha}{w^{(s)} + 1 + 6\alpha} & \text{otherwise} \end{cases} \quad (8)$$

We update the weights $w_y^{(s)}$ after each iteration by

$$w_y^{(s+1)} = w_y^{(s)} + f^{(s+1)}(y) - \beta(t) \quad \text{if } w_y^{(s)} + f^{(s+1)}(y) - \beta(s) \geq 0 \quad (9)$$

In this model, we have a new control parameter $\beta(s)$, which controls the length of the memory. If $\beta(s) = 0$ for all s, then $w_y^{(s)} = n_y^{(s)}$ for all s and y, and the new model is identical to the previous one. The larger the value of $\beta(s)$, the quicker the model forgets its past history. Usually, at the beginning of the clustering process, the operators have larger success rates than at the end. Thus, the model works better if $\beta(s)$ is larger for small s, and vice versa. By substituting $n_{y^*}^{(s)}$ with $w_{y^*}^{(s)}$ Eq. (7) also holds for the new model.

The new improved algorithm looks as follows:

Algorithm AMOLS

Step 1. Draw initially k random cluster centroids from the input data set;

Step 2. Let $s = 0$ and $p^{(s)}(y) = \frac{1}{6}$ for all y;

Step 3. Perform two iterations of GLA with the l_2-metric. Let the result of the clustering be P_{best} and calculate SC(P_{best}) by formula (3);

Step 4. Draw a random initial search operator amongst (SJ1, RWO, RSA, RSW, SJ2, CMO);

Step 5. Iterate the steps 5.1 to 5.7 until s = *max-iter*:

- *Step 5.1.* Apply search operator to P_{best} and let the solution be P_{mod};
- *Step 5.2.* Apply two iterations of GLA with l_2-metrics to the solution P_{mod};
- *Step 5.3.* Calculate SC(P_{mod});
- *Step 5.4.* If SC(P_{mod}) < SC(P_{best}), then let the P_{best} := P_{mod} and SC(P_{best}) := SC(P_{mod});
- *Step 5.5.* Increase s by one. Update probabilities $p^{(s)}(y)$ as defined by formula (8);
- *Step 5.6.* Update weights $w_y^{(s)}$ as defined by formula (9);
- *Step 5.7.* Draw a random operator amongst (SJ1, RWO, RSA, RSW, SJ2, CMO) weighted by distribution $p^{(s)}(y)$;

Step 6. Calculate $\hat{\Lambda}$;

Step 7. Apply GLA to minimise D given by (2) using P_{best} as an initial solution, until the result does not change.

3.5. Tabu Search

The tabu search algorithm [5] starts from a random initial solution, and then tries to improve it by making candidate moves in the neighbourhood of the current solution. If the best of the candidates is better than the current solution, we update the current one by the best candidate. Otherwise, we let the current solution be the best candidate from among those not in the so-called *tabu list*. The list has a limited size (like 20 in Fränti et al [5]), and the new current solution replaces the oldest one in the list.

The solution is expressed as a set of cluster centroids as in MOLS and AMOLS. (For a partition-based tabu search variant, see Fränti et al [5].) The initial classification is generated in the same way as with the MOLS and AMOLS methods, and fine tuned with GLA. In the trial classifications, a randomly chosen class is made obsolete, and a new one is generated by selecting a random data vector as the cluster centroid. Each data vector is then reassigned to the nearest class according to the l_2-metric. GLA is again applied to the solution.

It should be noted that the framework of tabu search leaves many degrees of freedom for the design and implementation of the algorithm (see Glover and Laguna [18] for a thorough discussion of the TS components and their role). These features include, among others the aspiration criterion (when to override a tabu restriction), the determination of tabu status on the basis of solution components (attributes), the determination of neighbourhood, the use of complementary tabu memory structures, intensification and diversification of the search, etc.

3.6. Comparison of Search Methods

The three search methods described above are variants of *neighbourhood search*. All start with an initial solution, and look at the neighbourhood of it in order to find a new more promising solution. The first two of these methods (MOLS and AMOS) are myopic, in the sense that they only move in the direction of locally better solutions. In a sense, they both are *descent methods*, but they are randomised because of the RWO, RSA, RSW and CMO operators. These operators perform a nondeterministic move in the search space. It can affect several clusters due to the application of GLA-steps, but still the effect is most probably rather limited. This is because the operators are applied to one or two clusters.

In contrast to the above, tabu search exploits the history, and is able to step in a direction with a worse cost function value. This increases the changes to cover a larger region of the search space. One can thus expect several different local minima to be found by the method. Perhaps this is one of the reasons why the results for TS have been excellent for a broad set of real applications in the literature. On the other hand, it is difficult to decide what would be the (standard or) best setting of the TS algorithm from among numerous possible options. It is therefore interesting to see whether a conceptually much simpler technique (like MOLS or AMOLS) can compete with this sophisticated technique. (At least we can, with a simple technique, use the extra time gained to repeat the same algorithm with different initial solutions, i.e., we may employ a so-called *iterated descent technique*.)

The AMOLS technique uses adaptation in the operator selection, which makes the method more robust and excludes much of the fine tuning. The same idea could be applied to TS, but then one has to reconsider the book-keeping of the adaptation parameters.

4. TEST PROCEDURE

We implemented the MOLS and AMOLS algorithms with the ANSI C language, and ran them with Digital UNIX on an AlphaStation 500/400.

The algorithms were applied to two types of test sets. The first test set was a binarised version of the picture 'Bridge' (see Fig. 1). It consists of 4096 16-bit vectors (2813 of these were unique). The 16-bit blocks were formed from 4×4 pixel areas of a grey scale image after a BTC-type quantisation into two levels, according to the mean value of the block [20].

The second test set consisted of data of 5313 strains of bacteria belonging to the *Enterobacteriaceae* family. The source of the material was the database of *Enterobacteriaceae* and *Vibrionaceae* compiled from 1972 to 1989 by the Enteric Bacteriology Laboratories, CDC, Atlanta, GA, USA. Each strain was characterised by a 47-bit binary vector. A detailed description of the data can be found elsewhere [21,22]. We call this data set Entero.

We applied the MOLS and AMOLS methods ten times to each of the test sets using 5000 iterations. We used 62 clusters for the bacterial data and 256 for the image data. In previous studies [1,22], we have shown that 62 is the optimal number of clusters for the bacterial data; on the other hand, 256 is a typical codebook size in image compression. We let $\alpha = 2.5$ and the function $\beta(s)$ be decreasing with respect to s (see Fig. 2).

For the purpose of comparison, we let the LS algorithm run with random-swap only (RSLS). We also performed repeated GLA tests, so that the running time was approximately the same (approximately one hour on a 400 MHz Alpha CPU) as for MOLS and AMOLS with 5000 iterations. This gave 400 GLA repetitions.

For comparison with TS with the standard settings of section 3.5 and some other well known methods (SOM = Self Organising Maps [23], SA = Simulated Annealing [4]) and PNN = Pairwise Nearest Neighbour [24], we modified the MOLS and AMOLS methods to minimise the distortion

$$\text{MSE} = \frac{1}{t} \sum_{l=1}^{t} \sum_{j=1}^{k} u_j^{(l)} \sum_{i=1}^{d} |x_i^{(l)} - a_{ij}| \qquad (10)$$

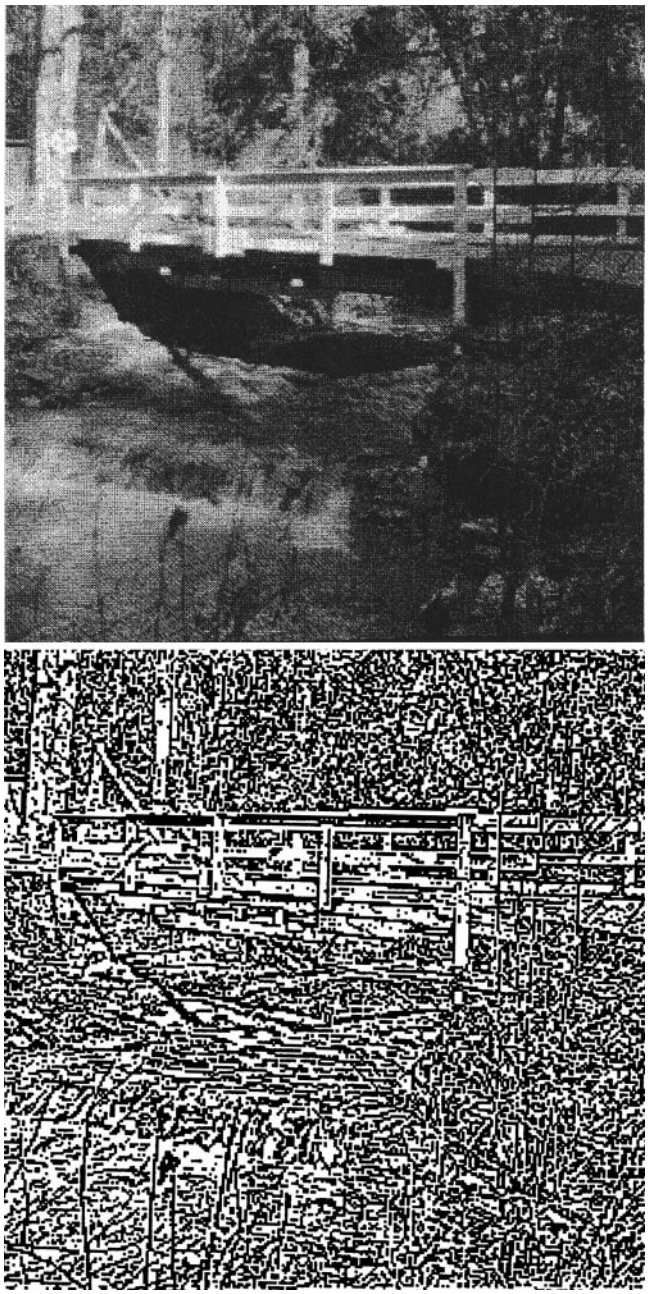

Fig. 1. Greyscale and binary versions of the 'Bridge' test image.

where $u_j^{(i)} = 1$ when $x^{(i)}$ belongs to the jth cluster, and zero otherwise. Here the cluster centroids $\mathbf{a}_1, \ldots, \mathbf{a}_d$ are the rounded Eq. (4). In this test, we used 10,000 iterations of MOLS and AMOLS, which roughly corresponds to the parameter settings of the TS. This comparison was only performed for the data set Bridge due to the availability of the results for TS, GLA, SA, SOM and PNN (see Fränti et al [5]).

Finally we studied the effect of different choices of $\beta(s)$. To do this, we sampled 1000 vectors from the Bridge data set, and applied the AMOLS algorithm to this sample. We used various different $\beta(s)$ functions for $k = 64$ clusters. We let $\beta(s)$ be a constant (0.0, 0.2 and 1.0) and a decreasing function of s (see Fig. 2). The value 0.0 means that AMOLS never forgets what it has learnt (i.e. it uses the first adaptive model presented in Section 3.4). On the other hand, the value 1.0 means that AMOLS forgets everything after one iteration step.

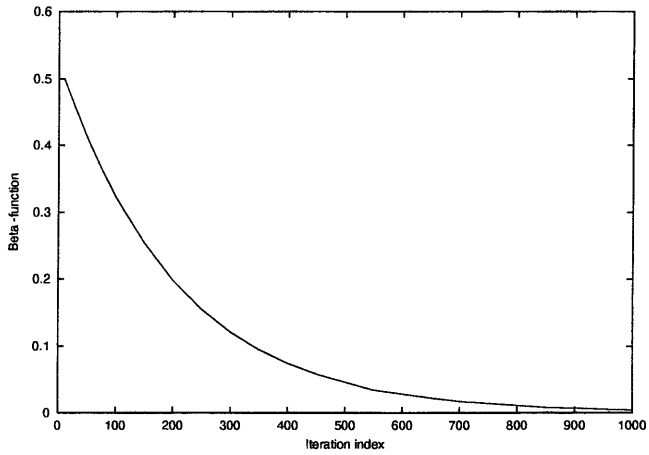

Fig. 2. The function $\beta(s)$ used in test runs for Bridge and Entero.

5. RESULTS

It is evident that, during the solution process, the power of some search operators is exhausted, and the MOLS should skip these and use other operators instead. The question is whether AMOLS can do any better. Figure 3 demonstrates the operation of the two algorithms for a typical run with Bridge. An 'x' is marked each time the algorithm finds a new, better value of SC. The figure shows that the adaptive pattern is already more clearly present at the beginning of the AMOLS process, whereas due to the cyclic application of the search operators, MOLS improves the solution quite evenly by all six operators. At the end of the process, patterns of improving methods look similar for AMOLS and MOLS, but AMOLS has the ability to alternate between different operators.

If we count the percentage of successful applications of the different operators, we can see that with MOLS, the profiles are similar with both data sets, whereas with AMOLS they are different. Figure 4 shows the relative frequencies of successful applications of different operators when counted over all of the repetitions. The figure shows that AMOLS can exploit the random-swap operator more effectively on the image data (the proportion of random-swap is 29% versus 8% of all successful applications). On the bacterial data, AMOLS exploits more replace-worst operator (46% versus 37% of successful applications). Both cases show the power of adaptation; different operators are more effective on different types of data sets.

Figure 5 shows the development of SC as a function of iteration index s for the best runs. We observe that the SC values of AMOLS are lower throughout the search with the

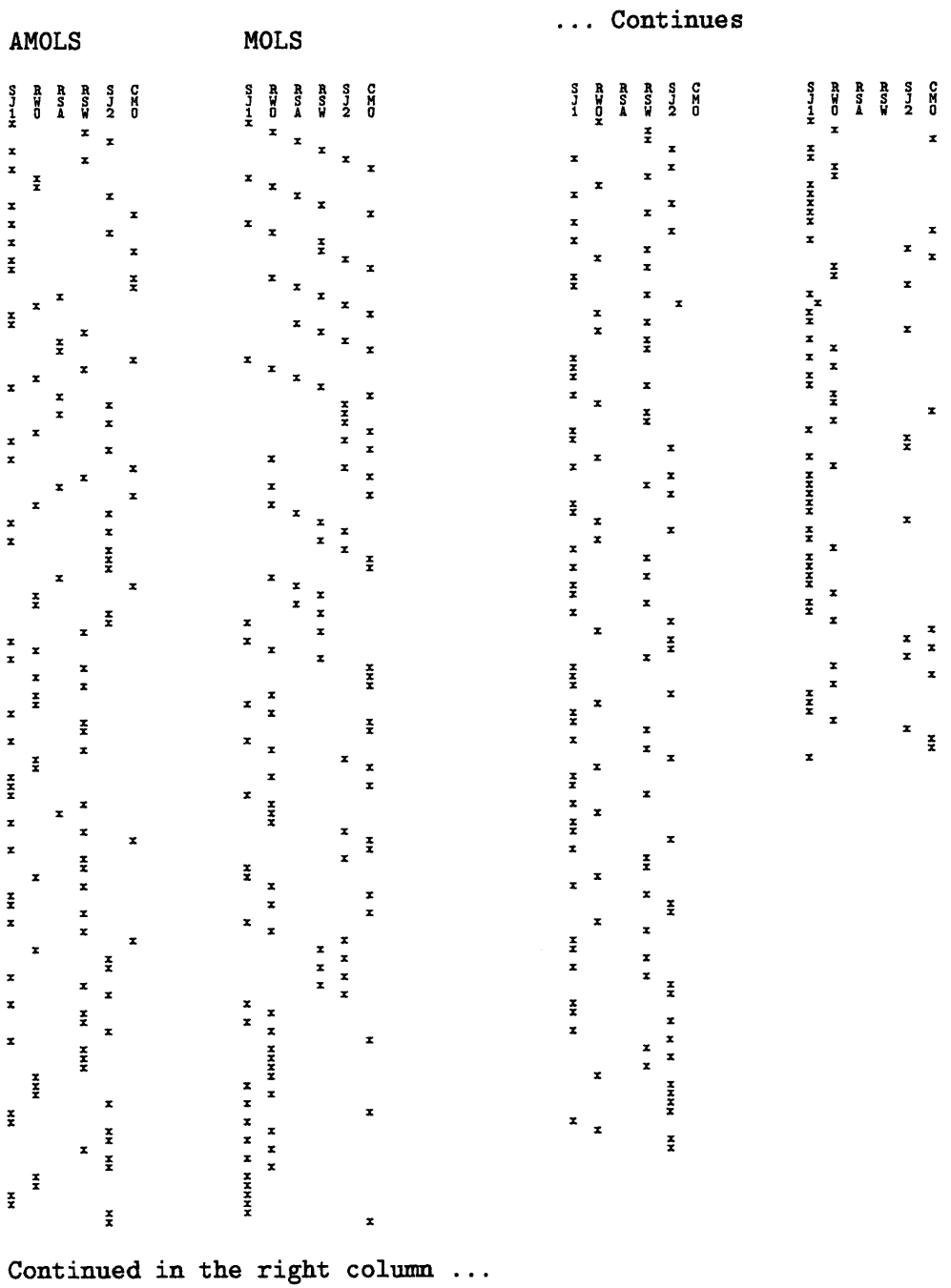

Fig. 3. Typical operator usage patterns of AMOLS (left) and MOLS for Bridge data set.

image data. This also holds for the bacterial data, approximately from iteration 1000 onwards. We also notice that successful iterations of AMOLS happen more in succession than with MOLS. Note that there is not much correlation between the SC values at the beginning and end of the two LS algorithms. This indicates that they concentrate on more promising areas of the search space.

Table 1 compares the average SC values of ten repetitions counted at a given iteration. The standard deviation is for the SC-values of the ten repetitions after 5000 iterations, and the 'best SC' gives the minimum of these SC-values. The last column shows the average number of successful applications of the operators. On the bacterial data, AMOLS had more successful applications of operators than MOLS (61% versus 49% on average). Also, the average SC (21.371) was statistically smaller than with MOLS (21.429) when tested with the *t*-test ($p = 0.0036 < 0.05$). The same holds for the image data. Here we have 226 successful applications for AMOLS against 204 of MOLS. The average SC for AMOLS (15.397) is statistically smaller ($p = 0.027$)

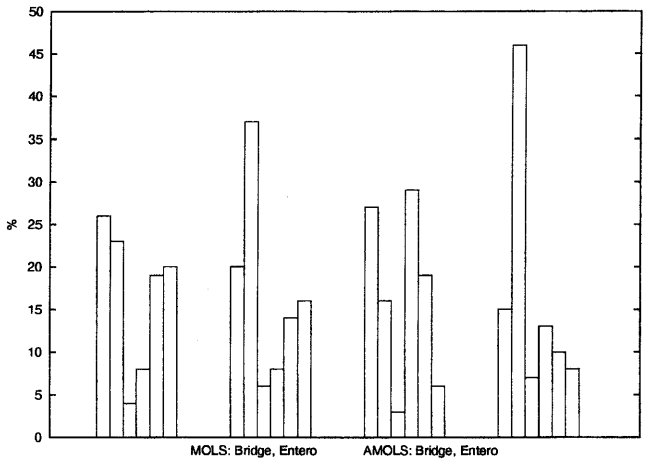

Fig. 4. Comparison of percentages of successful operator applications. There are four sets of six bars. The first set is for the MOLS Bridge, the second for the MOLS Entero, and the two next set for AMOLS. Six bars from the left are percentages per operation SJ1, RWO, RSA, RSW, SJ2 and CMO.

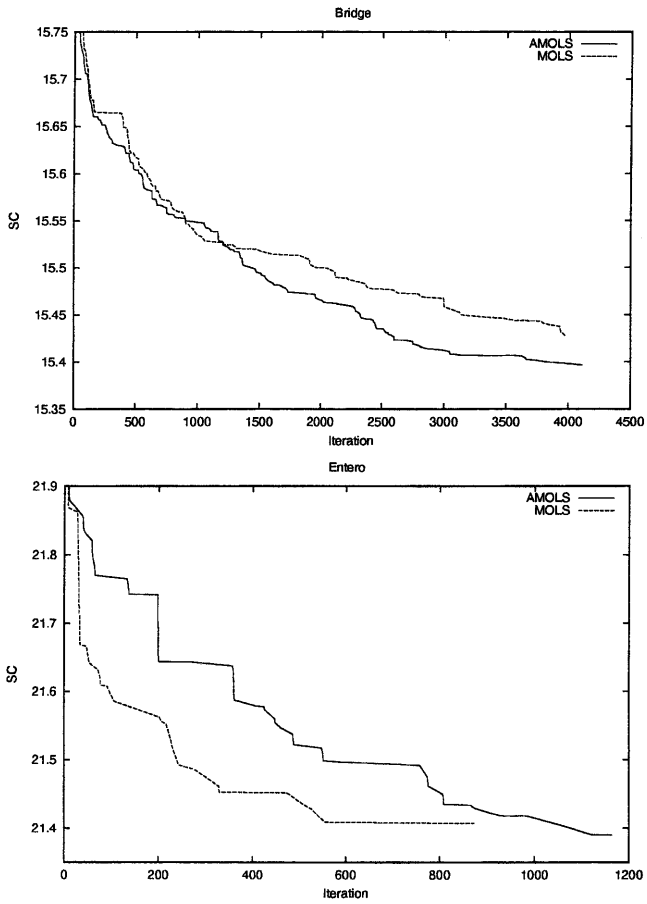

Fig. 5. Development of SC as a function of the iteration index (data plots of the best runs).

than for MOLS (15.417). The standard deviation of the SC values for the image data was almost the same for all of the algorithms, whereas with the bacterial data, AMOLS gave rise to the smallest standard deviation of the results.

The best SC produced by AMOLS for the bacterial data was 21.336, whereas MOLS produced 21.368. For the image data these values were 15.371 and 15.374, respectively. A closer investigation of the best results reveals that AMOLS produced a smaller number of small clusters than MOLS on the bacterial data: AMOLS produces no singleton clusters, while MOLS found four such clusters. On the other hand, the figures were just the opposite (45 and 35) for the image data.

When using only random-swap in our framework, the results were worse than those obtained by AMOLS and MOLS (SC was, on average, 21.604 with $\alpha = 0.057$ for Entero and 15.569 with $\alpha = 0.011$ for Bridge). If we look at the first 100 iterations, random-swap LS seems to work fine, but it does not succeed in any further iterations. However, RSLS gives better results than repeated GLA. Note that there was a considerably smaller number of successful iterations with RSLS than with AMOLS and MOLS. This suggests that other operators in AMOLS and MOLS do not exhaust the power of the random-swap operator.

Figure 6 illustrates the clustering results of the data set Entero by showing all 145 vectors (as rows of small white and black squares) belonging to genus *Salmonella*. Vectors appear in the figure in the order of their scientific name (left), and by the order in which they appear in the resulting clusterings of the MOLS (middle) and AMOLS (right) methods. The figure shows that clusters are not as clearly visible in the microbiological clustering as in the MOLS and AMOLS clusterings.

Table 2 compares a number of clustering techniques for the Bridge test set. The results for PNN, SOM, SA and TS are from Fränti et al [5]. The version of TS is highly optimised; this can be seen in its high quality MSE-value (1.27 in comparison to 1.32 of AMOLS). On the other hand, AMOLS performs well in comparison to the other methods, which have also been optimised.

Table 3 is a summary of test runs with different $\beta(s)$ settings (for ten repetitions) with a test data sample from Bridge. The decreasing $\beta(s)$-function of Fig. 2 gives the overall best results. The intuition is that, at the beginning of the search process, the operators are more successful and the learning process should use shorter memory there. With a value of 0.0, the method does not abandon an operator which has once been successful, and with a value of 1.0, AMOLS behaves more like MOLS. A compromise between these two extremes (like 0.2) works satisfactorily.

6. DISCUSSION

We have studied two versions of a multi-operator LS algorithm; non-adaptive and adaptive. Our study shows that the Multi-Operator Local Search (MOLS and AMOLS) is capable of finding very low cost clusterings. The AMOLS adapts quickly to favour SJ1, RWO, RSW and SJ2, whereas MOLS

Table 1. Summary of test runs (10 reptitions). The initial solution (0 iterations) stands for the situation after the initial application of GLA

Data	Method	Average SC				Standard deviation	Best SC	Average successes
		0	100	1000	5000			
Bridge	AMOLS	15.966	15.688	15.527	15.397	0.023	15.371	226
	MOLS	15.941	15.714	15.529	15.417	0.018	15.373	204
	LS	15.969	15.646	15.598	15.569	0.011	15.552	80
	RGLA				15.657	0.017	15.612	
Entero	AMOLS	23.225	21.764	21.495	21.371	0.022	21.335	61
	MOLS	23.461	21.769	21.513	21.429	0.049	21.368	49
	LS	23.227	21.809	21.714	21.604	0.057	21.462	19
	RGLA				21.809	0.135	21.508	

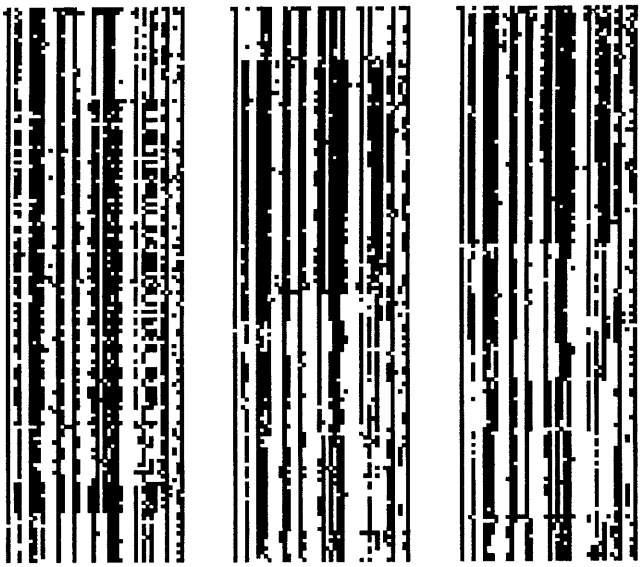

Fig. 6. Vectors belonging to the genus *Salmonella*. Left: vectors ordered by their scientific name; centre: the vectors in the order in which they have been found in the best classification by MOLS; right: by ALMOS.

Table 2. Comparison of clustering algorithm when minimising MSE of the clustering for Bridge. The results are averages of five runs

Method	MSE
MOLS	1.334
AMOLS	1.327
TS	1.27
PNN	1.33
SOM	1.39
GLA	1.48
SA	1.52

Table 3. The average SC and number of successes with different choices of $\beta(s)$ (for ten test runs). The function $f(s)$ is decreasing with s (see Fig. 2)

$\beta(s)$	Average SC	Average number of successes
$f(s)$	15.264	60
0.2	15.270	58
1.0	15.277	55
0.0	15.286	56

finds improvements most frequently with SJ1, RWO, SJ2 and CMO. For the bacterial and image test data sets, the behaviour is similar. Additionally, AMOLS used RWO more frequently on the bacterial data and RSW on the image data. The overall number of successful applications of the operators is greater for the adaptive methods for both data sets.

Although the quality of the results of MOLS and AMOLS is similar, AMOLS finds statistically better average clustering. Here, the absolute value of the difference to MOLS is small, but the standard deviation of the results is also very small. Comparison with other algorithms has revealed that AMOLS is also a very good method in minimising MSE, and not much behind a fine-tuned TS-algorithm.

Our test results indicate clearly that a carefully implemented multi-operator local search gives better results than the single operator approach. Due to its versatility, AMOLS is not only a very efficient, but also a multi-purpose, method; it has a good capability to adapt to different types of data. The extra computational cost of adding adaptation to MOLS is small, because the update of the probabilities (8) is very fast. This makes AMOLS a practical clustering algorithm for large clustering problems.

Acknowledgements

This work has been supported by Academy of Finland, The Swedish Natural Science Research Council (NFR), and by the Knut and Alice Wallenberg Foundation.

References

1. Fränti P, Gyllenberg HG, Gyllenberg M, Kivijärvi J, Koski T, Lund T, Nevalainen O. Minimizing stochastic complexity using local search and GLA with applications to classification of bacteria. Biosystems 2000; 57:37–48
2. Reeves RC, ed. Modern Heuristic Techniques for Combinatorial Problems. McGraw-Hill, 1995; 70–150
3. Hoffmeister F, Bäck T. Genetic self-learning. In: Varela FJ, Bourgine P, eds, Toward a Practice of Autonomous Systems: Proc First European Conf Artificial Life (ECAL '92), Paris, December 11–13 1991; 227–235
4. Vaisey J, Gersho A. Simulated annealing and codebook design. Proc. ICASSP 1988; 1176–1179
5. Fränti P, Kivijärvi J, Nevalainen O. Tabu search algorithm for codebook generation in vector quantization. Pattern Recognition 1998; 31(8):1139–1148
6. Fränti P, Kivijärvi J. Random swapping technique for improving clustering in an unsupervised classification. Proc 11th Scandinavian Conf on Image Analysis (SCIA '99), Kangerlussuaq, Greenland, 1999; 407–413
7. Wolpert DH, Macready WG. No free lunch theorems for optimziation. IEEE Trans on Evolutionary Computation 1997; 1(1):67–82
8. Hinterding R, Michalewicz Z, Eiben AE. Adaptation in evolutionary computation: A survey. IEEE Int Conf on Evolutionary Computation, Indianapolis, April 13–16, 1997; 65–69
9. Magyar G, Johnsson M, Nevalainen O. An adaptive hybrid genetic algorithm for the 3-matching problem. IEEE Transactions on Evolutionary Computation 2000; 4
10. Gyllenberg M, Koski T, Verlaan M. Classification of binary vectors by stochastic complexity. Multivariate Analysis 1997; 63:47–72
11. Rissanen J. Stochastic Complexity in Statistical Inquiry. World Scientific, 1989
12. Bischof H, Leonardis A, Sleb A. MDL principle for robust vector quantization. Pattern Analysis and Applications 1999; 2:59–72
13. Kaukoranta T, Fränti P, Nevalainen O. Reallocation of GLA Codevectors for Evading Local Minimum. Turku Center for Computer Science, TUCS Technical Report No 25, 1996
14. Cherkassky V, Mulier F. Learning from Data: Concepts, Theory and Methods. Wiley, 1998
15. Linde Y, Buzo A, Gray RM. An algorithm for vector quantizier design. IEEE Trans on Commun 1980; 28:84–95
16. Forgy E. Cluster analysis of multivariate data: Efficiency vs. interpretability of classifications. Biometrics 1965; 21:768
17. McQueen JB. Some methods of classification and analysis of multivariate observations. Proc 5th Berkeley Symposium in Mathematics, Statistics and Probability 1967; 1:281–296
18. Penã JM, Lozano JA, Larranãga P. An empirical comparison of four initialization methods for the k-means algorithm. Pattern Recognition Letters 1999; 20:1027–1040
19. Glover F, Laguna M. Tabu search. In Reeves RC, ed, Modern Heuristic Techniques for Combinatorial Problems. McGraw-Hill, 1995; 70–150
20. Fränti P, Kaukoranta T. Binary vector quantizer design using soft centroids. Signal Processing: Image Commun 1999; 14: 677–681
21. Farmer III, JJ, Davis BR, Hickman-Brenner FW, McWhorter A, Huntley-Carter GP, Asbury MA, Riddle C, Wahten-Grady HG, Elias C, Fanning GR, Steigerwalt AG, O'Hara M, Morris GK, Smith PB, Brenner DJ. Biochemical identification of new species and biogroups of *Enterobacteriaceae* isolated from clinical specimens. J Clinical Microbiology 1985; 21:46–76
22. Gyllenberg HG, Gyllenberg M, Koslei T, Lund T, Schindler J, Verlaan M. Classification of *Enterobacteriaceae* by minimization of stochastic complexity. Microbiology 1997; 143:721–732
23. Nasrabadi NM. Vector quantization of images based upon the Kohonen self-organization feature maps. Neural Networks 1988; 1:518–518
24. Equitz WH. A new vector quantization clustering algorithm. IEEE Trans Acoustics Speech Signal Process 1989; 1:1568–1575

Mats Gyllenberg received his Doctor of Technology degree from the Helsinki University of Technology. He has been Professor of Applied Mathematics at the Luleå University of Technology (1989–1992) and at the University of Turku since 1992. He was a visiting scientist at the Center for Mathematics and Informatics in Amsterdam (1984–1985), and a visiting professor at Vanderbilt University, Nashville Tennessee (1985–1986), the National Center for Ecological Analysis and Synthesis, Santa Barbara, California (1996), the University of Utrecht (1997) and at the Chalmers University of Technology, Gothenburg (1998). He is presently a Member of the Research Council for Natural Sciences and Technology Academy of Finland, and the Vice President of the European Society for Mathematical and Theoretical Biology. His research interests include mathematical population dynamics, theoretical ecology and mathematical taxonomy.

Timo Koski is Professor of Mathematical Statistics at the Royal Institute of Technology, and acting as a part time Professor of Applied Mathematics at the University of Turku. He has previously worked as an associate professor with Luleå University of Technology, Luleå, Sweden, and took his PhD at the Abo Akademi University in 1986. His scientific interests are in probability theory and bioinformatics.

Tatu Lund received a MSc degree in computer science from the University of Turku, Finland in 1997. Since 1995 he has been working at the Department of Mathematical Sciences, Institute of Applied Mathematics at University of Turku, Finland as an assistant. He has also been a PhD student of ComBi (Graduate School in Computational Biology, Bioinformatics, and Biometry) since 1998. His research interests include classification algorithms, Bayesian learning and neural computing.

Olli Nevalainen received his MSc and PhD degrees in 1969 and 1976, respectively. From 1972 to 1979, he was lecturer in the Department of Computer Science, University of Turku, Finland, where from 1979 to 1999 he was an Associate Professor, and since 1999 a Professor. He lectures in the area of data structures, algorithm design and analysis, compiler construction and operating systems. His research interests are algorithm design, including image compression, vector quantisation, scheduling algorithms and production planning.

Correspondence and offprint requests to: T. Lund, Department of Mathematical Sciences, University of Turku, FIN-20014, Turku, Finland.