Design of Hybrids for the Minimum Sum-of-Squares Clustering Problem

Joaquín Pacheco[1], Olga Valencia

Department of Applied Economics, University of Burgos,
Plaza Infanta Elena s/n BURGOS 09001, SPAIN

Latest version: July 8, 2002

Abstract

A series of metaheuristic algorithms is proposed and analyzed for the non-hierarchical clustering problem under the criterion of minimum Sum-of-Squares Clustering. These algorithms incorporate genetic operators and Local Search and Tabu Search procedures. The aim is to obtain quality solutions with short computation times. A series of computational experiments has been performed. The proposed algorithms obtain better results than previously reported methods, especially with small numbers of clusters.

Keywords: Clusterization, Metaheuristics, Tabu Search, Genetic Algorithms, Memetic Algorithms, Hybrid Algorithms.

1. Introduction

Consider a set X = {x_1, x_2, ..., x_N} of N points in R^q and let m be a predetermined positive integer. The Minimum Sum-of-Squares Clustering (MSSC) problem is to find a partition of X into m disjoint subsets (clusters) so that the sum of squared distances from each point to the centroid of its cluster is minimum. Specifically, let P_m denote the set of all the partitions of X in m sets, where each partition $P \in P_m$ is defined as $P = (C_1, C_2, ..., C_m)$ and where C_i denotes each of the clusters that forms P. Thus, the problem can be expressed as:

$$\min_{P \in P_m} \sum_{i=1}^{m} \sum_{x_l \in C_i} \left\| x_l - \bar{x}_i \right\|^2 ,$$

where the centroid $\bar{x}_i$ is defined as

$$\bar{x}_i = \frac{1}{n_i} \sum_{x_l \in C_i} x_l , \qquad \text{with } n_i = |C_i| .$$

Equivalently the problem can be written as

$$\min \sum_{l=1}^{N} \left\| x_l - \bar{x}_{c(l)} \right\|^2 ,$$

where $c(l)$ is the cluster to which point x_l belongs.

The design of clusters is a well known exploratory Data Analysis issue called Pattern Recognition. The aim is to find whether a given set of cases X has some structure and, in if so, to display it in the form of a partition. This problem belongs to

[1] Corresponding author: Fax: +34 947 25 80 13; Tel: +34 947 25 90 21
E-mail addresses: jpacheco@ubu.es (Joaquín Pacheco), oval@ubu.es (Olga Valencia).

the area of Non-Hierarchical cluster design, which has many applications in Economics, Social and Natural Sciences. It is known to be NP-Hard, (Brucker, 1978).

Various exact methods for MSSC can be found in the literature (see, for example, Koontz et al., 1975 and Diehr, 1985), some of which, such as the method proposed by du Merle et al. (2000), have succeeded in resolving problems with up to 150 points. For larger-sized problems the use of heuristic algorithms is still necessary. The most popular are those based on Local Search methods, such as the well-known K-Means (Jancey, 1966) and H-Means (Howard, 1966) procedures. In a recent work, Hansen and Mladenovic (2001) propose a new Local Search procedure, J-Means, along with variants H-Means+ or HK-Means. In recent years algorithms using Metaheuristic strategies have been designed, such as Simulated Annealing (Klein and Dubes, 1989), Tabu Search (Al-Sultan, 1995), Genetic Algorithms (Babu and Murty, 1993) or most recently Variable Neighborhood Search or VNS (du Merle et al., 2000, and Hansen and Mladenovic, 2001).

A series of algorithms that is able to obtain good solutions in short times is proposed for this problem. Initially, a genetic algorithm is designed using Local Search methods, thus becoming a Memetic Algorithm. A simple procedure based on a Tabu Search method using binary trees is also suggested. This method demonstrates its capacity to improve solutions in very few iterations. The incorporation of this procedure into Memetic Algorithms yields Hybrid Algorithms. Finally, these Memetic and Hybrid Algorithms are analyzed and compared with other techniques. In all cases, the proposed techniques give adequate solutions, compared with other techniques, in reasonable time, especially with small values of *m*.

This paper is structured as follows: in the next section some of the already existing Local Search algorithms are described. In the third section, the Tabu Search method is presented. Section 4 considers the Genetic Algorithm in detail. In section 5 Memetic and Hybrid Algorithms are described. Section 6 presents the results of different computational experiments and compares the effectiveness and efficiency of the proposed algorithms with other recent techniques. Finally section 7 summarizes the contribution.

2. Principal Local Search Algorithms

Some important local search algorithms are described. These are the well-known H-Means and K-Means, their variants H-Means+ and HK-Means, as well as the more recent J-Means and J-Means+. In all cases, we begin by obtaining an arbitrary initial solution (or partition), $(C_1, C_2, ..., C_M)$.

Algorithm 1. H-Means

```
Repeat
    (a) Calculate centroids $\bar{x}_i$, for $i=1....m$;
    (b) Reassign every point to its closest centroid ($\rightarrow$ new clusters: clusters are composed by the
        points assigned to the same centroid);
until convergence (that is, no modifications in the composition of clusters)
```

Algorithm 2. H-Means+

```
Repeat
    (a) Run steps (a) and (b) of H-Means (Algorithm 1);
    (b) If there are no modifications:
            (b.1) Verify if empty clusters exist (degeneration);
            (b.2) If there are $k$ empty clusters select the farthest $k$ points from their
                  centroids and insert them into the solution as $k$ new single-point clusters;
until convergence.
```

Algorithm 3. K-Means

```
Repeat
      (a) $\forall i = 1..m.$ $\forall$ $x_j \notin C_i$ calculate $v_{ij}$ (that is: change in the value of the objective
          function when $x_j$ is reassigned to $C_i$);
      (b) If $v_{i^*j^*} = \min v_{ij} < 0$ then reassign $x_{j^*}$ to $C_{i^*}$ ;
until $v_{i^*j^*} \geq 0$
```

Algorithm 4. J-Means

```
Repeat
      (a) $\forall$ $j = 1....N$ such that $x_j$ do not coincide with any actual centroid:
            (a.1) Add a fictitious cluster $C_{M+1}$ whose centroid is $x_j$ ;
            (a.2) $\forall$ $i=1....m$, calculate $J_{ij}$; that is: the change in the value of the objective function
                  when cluster $C_i$ is 'dissolved' and each one of its entities is reassigned to its
                  closest centroid cluster among the remaining ones (including $C_{M+1}$ and not
                  including $C_i$) ('Jump-move');
      (b) If $J_{i^*j^*} = \min J_{ij} < 0$ then undo $C_{i^*}$ and reassign the elements of $C_i$ to the closest centroid
          cluster among the remaining ones (including $C_{M+1}$ of centroid $x_{j^*}$), redefine $C_i = C_{M+1}$ and
          eliminate $C_{M+1}$.
until $J_{i^*j^*} \geq 0$.
```

Observe that H-Means+ simply modifies H-Means to prevent the algorithm from ending in a degenerate solution. This modification is very effective according to Hansen and Mladenovic's (2001) experiments.

The following formulas are obtained from Späth (1980) to simplify the calculations in K-Means. Let C_l be the cluster to which x_j belongs, then the value of v_{ij} is calculated as follows:

$$v_{ij} = \frac{n_i}{n_i + 1} \cdot \left\| \bar{x}_i - x_j \right\|^2 - \frac{n_l}{n_l - 1} \cdot \left\| \bar{x}_l - x_j \right\|^2 ,$$

and the centroids are easily updated as x_j is changed from C_l to C_i. It should be noted that a K-Means local optimum is also a local optimum for H-Means (and H-Means+), but the reverse is not necessarily true. This suggests using H-Means+ followed by K-Means and not the reverse. We shall denote this two phase heuristic by HK-Means.Finally, J-Means+ consists in applying HK-Means once every time 'Jump-move' is run in J-Means.

3. Tabu Algorithm

3.1. Description of a basic algorithm

Tabu Search (TS) is a strategy proposed by Glover (1989) and (1990). "Tabu Search is dramatically changing our possibilities of solving a host of combinatorial problems in different areas" (Glover and Laguna, 2002). This is a procedure that explores the solution space beyond the local optimum. Once a local optimum is reached, upward moves or those that worsen the solutions are allowed. Simultaneously, the last moves are marked as *tabus* during the following iterations to avoid cycling. Recent and comprehensive tutorials on Tabu Search, that include all types of applications, can be found in Glover and Laguna (1997) and (2002).

Next, we design a simple Tabu Search algorithm that uses the neighboring moves employed in K-Means. These moves consist at each step in the movement of an entity from a cluster to a different one. In order to avoid repetitive cycling when a move which consists in moving point x_j from cluster C_l to cluster C_i is performed, point x_j is prevented from returning to the cluster C_l for a certain number of iterations. Specifically, let's define

Matrix_tabu (l, j) = the number of the iteration in which point x_j leaves cluster C_l .

The Tabu Search method is described by Algorithm 5, where P denotes a initial solution with a value f. The parameter *Tabu_Tenure* indicates the number of iterations during which a point is not allowed to return to the leaving cluster. The parameter *max_iter* indicates the maximum number of unimproved iterations. After different tests, *Tabu_Tenure* was set as m.

```
Algorithm 5. Tabu Search
(a) Do Matrix_tabu(i,j) = - Tabu_Tenure, i =1..m, j = 1,..N
(b) Do niter = 0 and P* = P, f* = f and iter_better =0;
(c) Repeat
        (c.1) niter = niter + 1
        (c.2) Determine $v_{i*j*}$ = min {$v_{ij}$ / i =1..m. j = 1,...N ; $x_j \notin C_i$ verifying
                                  niter > Matrix_tabu (i,j) + Tabu_Tenure or
                                  f + $v_{ij}$ < f* ('aspiration criterion')}
        (c.3) Reassign $x_{j*}$ to $C_{i*}$ ;
        (c.4) Do Matrix_tabu (l*,j*) = niter (l* being the previous cluster of $x_{j*}$);
        (c.5) If f (the value of the current solution P) < f* then do: P* = P, f* = f and iter_better = niter;
    until (niter – iter_better > max_iter) or another termination criterion
```

3.2. The Use of a Binary Tree structure

Next, a strategy based on the use of binary trees (Williams, 1964) is proposed for storing and arranging the different possible moves, defined for each (i, j) pair, according to the corresponding value v_{ij}. For each node k of the binary tree, a given move (i, j) is located so that, in the leaving nodes, $2k$ to $2k$+1, two moves (i', j') and (i'', j'') are located that are either equal to or worse than move (i, j), (i.e., $v_{ij} \leq v_{i'j'}$ and $v_{ij} \leq v_{i''j''}$). Clearly, in any iteration, the best move will be located in the root node. The binary tree does not include the tabu moves. A binary tree can be programmed using a vector *HP* and a matrix *QP* :

- *HP*(k) : indicates move (i, j) located in node k of the tree
- *Q*(i, j) : indicates the number of the node where move (i, j) is found; if this move is not in the tree (i.e., $x_j \in C_i$ or *niter > Matrix_tabu (i,j)+ Tabu_Ternure*) then Q(i, j) = 0.

Figure 1 illustrates a binary tree. In this case *HP* must be equal to ((1,2), (2,4), (3,5), (2,1), (4,6)) and *QP*(1,2) = 1, *QP*(2,4) = 2, *QP*(3,5) = 3, *QP*(2,1) = 4, and *QP*(4,6) = 5.

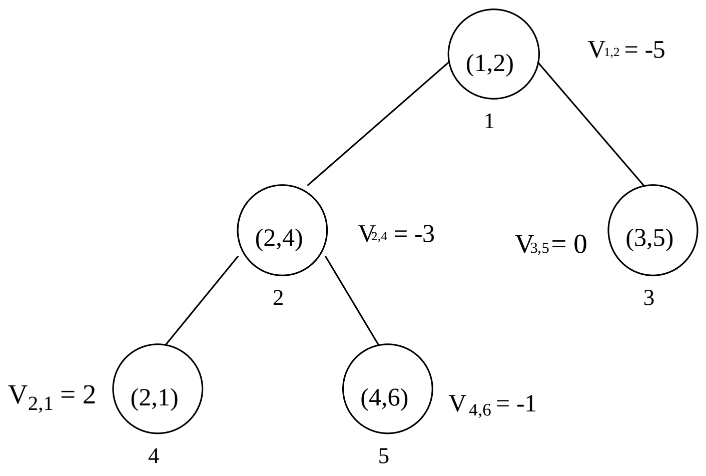

Fig. 1. An example of a binary tree to arrange the moves.

When a move defined for *i** and *j** is run, the positions of the moves with values located in the binary tree must be updated. Thus, move (*i**, *j**) in that iteration must be eliminated, and the move that is no longer tabu must be reincorporated in the tree. As the number of nodes of the binary tree is at most $N\cdot(m-1)$, each of these binary tree update operations can be implemented in $O(\log_2 (N\cdot m))$ time. As can easily be shown, the number of moves (*i*, *j*) that must be updated is on the order of $2\cdot N + \left(|C_{i^*}| + |C_{l^*}|\right)\cdot (m-3)$. Thus, the number of operations in each iteration is $O(N\cdot\log_2(N\cdot m))$. We denote by Tabu_Search_with Binary tree the procedure incorporating the above binary tree into the Tabu Search algorithm. Logically, the use of the binary tree does not affect the final result. However, important savings in calculation time are expected.

4. Genetic Algorithm

According to Goldberg (1989) "*Genetic Algorithms are search techniques based on the mechanics of natural selection and genetics*". These techniques are probably the best-known and widespread evolutionary algorithms. They were originally conceived by John Holland and described in the classic monograph "*Adaptation in Natural and Artificial Systems*" (Holland, 1975). This text has had a great influence on the later development of these techniques since the mechanisms described in it have long since been adopted as dogma.

They are based on a close analogy with the natural processes of the evolution of species. The base is an ensemble (population) of solutions on which are performed operations such as Selection, Crossover, Reproduction, Renovation or Mutation. An outline of how a Genetic algorithm operates is described by Algorithm 6:

```
Algorithm 6. Genetic Algorithm
Generate an initial population of solutions;
Repeat
    (a) Randomly select a subset of elements (even) of the population with a probability
        proportional to its goodness;
    (b) Cross reproduction: pair or cross these solutions (Parents) to give rise to new solutions
        (Children). From each pair of parents a new pair of 'child' solutions has to be generated;
```

(c) *Mutation: the solutions of the population can change some of its elements (genes) with a small probability;*
(d) *Substitute the worst solutions of the population with the new child solutions;*
until reaching some stop criterion

The solutions are represented by a set of *m* seed points $S = \{s_1, s_2, ..., s_m\} \subset R^q$. The corresponding partition $P = (C_1, C_2, ..., C_m)$ is the one resulting from assigning each point of X to its closest seed point.

The 'non-goodness' of each solution of the population will be given by the value of the objective function *f* of the corresponding partition. The selection probability of a given solution S_i will be proportional to $fmax - f_i$; where f_i is its non-goodness value and *fmax* is the maximum value of f_i of this population. Once selected, the parent solution pairs are crossed as illustrated in the following example. Let *m* = 3 and let two parent solutions be $S^1 = \{A_1, B_1, C_1\}$ and $S^2 = \{A_2, B_2, C_2\}$, then the matrix of distances between the *seeds* of S^1 and S^2 is calculated assuming the results shown in Table 1.

Table 1
Example of distances between seeds

$S^1\downarrow$ $S^2\rightarrow$	A_2	B_2	C_2
A_1	7	9	6
B_1	3	5	4
C_1	5	4	9

Then the seeds of S^2 are rearranged according to their proximity to S^1 as follows: starting from A_1 the closest seed of S^2, C_2, is chosen and positioned in the first place. The column of C_2 is eliminated for consecutive choices. Next, out of the remaining seeds of S^2 the one closest to B_1 is selected, A_2 in this case, and its column is eliminated. Finally, the only remaining seed B_2 is selected. In this way, the elements of S_1 and S_2 are arranged as follows:

$$S^1 \rightarrow (A_1, B_1, C_1) \text{ and } S^2 \rightarrow (C_2, A_2, B_2).$$

Subsequently, an integer value between 1 and *m*-1 is randomly selected as an intersection point (*cross_pt*), (in the example *cross_pt* = 1), and two child solutions S^3 and S^4 are generated as shown in Tables 2 and 3.

Table 2
Parent Solutions

$S^1 \rightarrow$	A_1	B_1	C_1
$S^2 \rightarrow$	C_2	A_2	B_2

Table 3
Child Solutions

$S^3 \rightarrow$	A_1	A_2	B_2
$S^4 \rightarrow$	C_2	B_1	C_1

The aim of pairing the elements S^1 and S^2 according to their closeness is to prevent entities or seeds that are close to each other from forming the same child solution. Experience shows, in general, that if there are very close pairs of seeds-points of a solution *S*, the corresponding partition is poor (Cano, 1999). A series of tests has been performed to verify the efficiency of this arrangement process. First, *N* = 1000 is set and random values of X between 0 and 100 are uniformly generated. The results clearly show the positive effect of this arrangement process for every value of *m* considered, (*m* = 10, 20, … 150).

Every seed of every generated child solution can change or mutate with a small probability, *mut_p*. To decide whether a given seed changes, a random value in the interval (0,1) is uniformly generated. If this value is less than *mut_p* the change takes place: the seed is randomly placed in a point in the range of defined values for X (Mutation). This mutation process supplies diversity to the process and prevents the search from getting boxed into a zone around the local minimum.

Once an even number of child solutions is generated (equal to the number of selected parents), the worst solutions of the population are deleted and replaced by the child solutions such that the number of elements remains constant (Renovation).

Each iteration (Parent Selection $\rightarrow$ Crossover or Reproduction $\rightarrow$ Mutation $\rightarrow$ Renovation) is also denoted as ‘generation’. The algorithm ends when a given number of generations without improvement (*max_iter*) occurs, or when a certain preset time elapses. Besides the parameters mentioned before, *p_mut* and *max_iter*, two other parameters are used: *n_pop*, the number of solutions that constitute the population, and *n_sel*, the number of solutions selected as parents.

5. Memetic and Hybrid Algorithms

Memetic Algorithms are also population-based methods and have been demonstrated to be faster than Genetic Algorithms for certain classes of problems, (Moscato and Laguna, 1996). In brief, they combine local search procedures with crossing or mutating operators; due to their structure some researchers have called them Hybrid Genetic Algorithms, Parallel Genetic Algorithms (PGAs) or Genetic Local Search methods. The method is gaining wide acceptance particularly for the well-known problems of combinatorial optimization. A recent tutorial can be found in Moscato, (2002).

In our case, the proposed Memetic Algorithm acts like the Genetic Algorithm described in section 4, except Local Search procedures are added each time a new solution is generated. The Local Search procedures can be any of those described above: HK-Means, J-Means or J-Means+. When a new solution is generated, these processes act on the corresponding partition leading to a new improved partition *P*. The set *S* of seed points of this new partition *P* is defined as the set of centroids of the clusters in *P*. It must be noted that, after running HK-Means or J-Means+, the new partition *P* will not vary if we reassign the points of X to the closer seeds (centroids in this case), because *P* is the local optimum with respect to H-Means. In other words, the partition corresponding to *S* continues to be *P*.

In many cases the incorporation of Local Search methods improves the performance of the original genetic algorithms, although sometimes this has the effect of accelerating the process towards a local minimum. The use of higher values in the mutation probability (*p_mut*) is recommended to ensure a certain degree of diversification to avoid this problem.

The Tabu Search procedure with only a few iterations demonstrates its capacity to improve solutions, even ‘good’ local optima (see section 6.2). Therefore, we propose modifying the Memetic proposed algorithm by appending Tabu Search (with *max_iter* = 50 to avoid excessive computation time) to the local search procedure. The aim is to obtain a Hybrid method that gives better results than the original Memetic one.

6. Computational Results

Next the results of a set of computational experiments using the proposed algorithms are shown. For each case, the TSPLIB library file (Reinelt, 1991) with N = 1060 points is used with different numbers of clusters, m = 10, 20, 30, ..., 150. These test data were previously used in Hansen and Mladenovic (2001), where the best solution known for every value of m (except m = 40) is reported. These were obtained on a *SUN Ultra I System* workstation with 10 minutes computation time. All the tests in the current work are performed on a personal computer with a Pentium III 600 MHz processor.

6.1. Empirical Analysis of using the Binary Tree in Tabu Search

We present the results of a series of tests carried out in order to verify the reduction in computation time achieved by using the binary tree in Tabu Search. For each value of m, both Tabu Search *(TS)* and Tabu Search with Binary Tree *(TSB)* are run. Each uses the same initial solution and performs a fixed number (500) of iterations. The initial solution (SA) is obtained as follows: m centroids (with uniform distribution in the X value range) are randomly generated and each point is directly assigned to the closest centroid. The obtained computation times (in seconds) are shown in figure 2. It is clear that a reduction in computation time is achieved using the binary tree, especially as m grows (except for m =10). Thus, from now on, the TSB variant will be used when Tabu Search is employed.

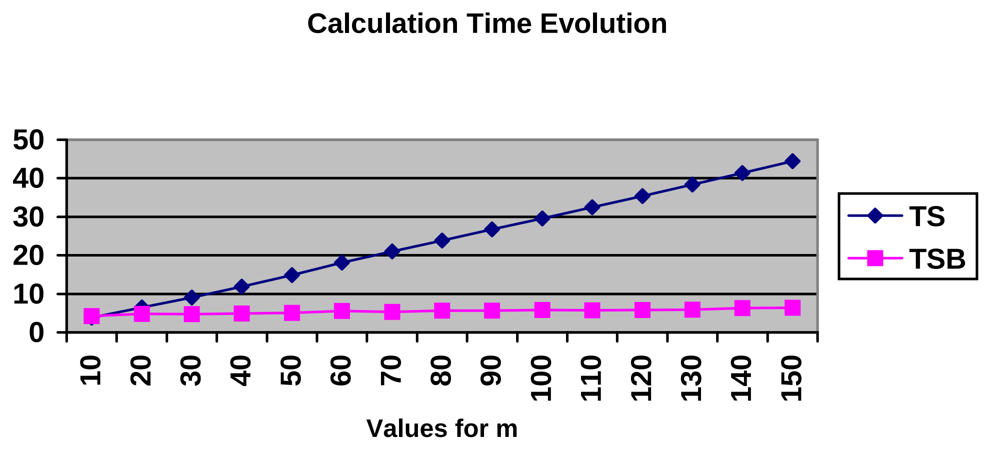

Fig. 2. Calculation Times of TS and TSB.

6.2. Tabu Search to improve Local Minima: Empirical Analysis

To demonstrate the efficiency of this *Tabu Search* method, a series of tests is run. In these tests the two-part solutions obtained by the greedy-random method - as proposed by Beltrán and Pacheco, (2001) - are improved by the J-Means+ algorithm. The solutions obtained by this Local Search method are in turn improved by the Tabu Search algorithm. In this case, Tabu Search uses the stop criterion *max_iter* = 50. The process is repeated ten times for each number of clusters m. The corresponding average percent deviation with respect to the best solution known is presented in figure 3.

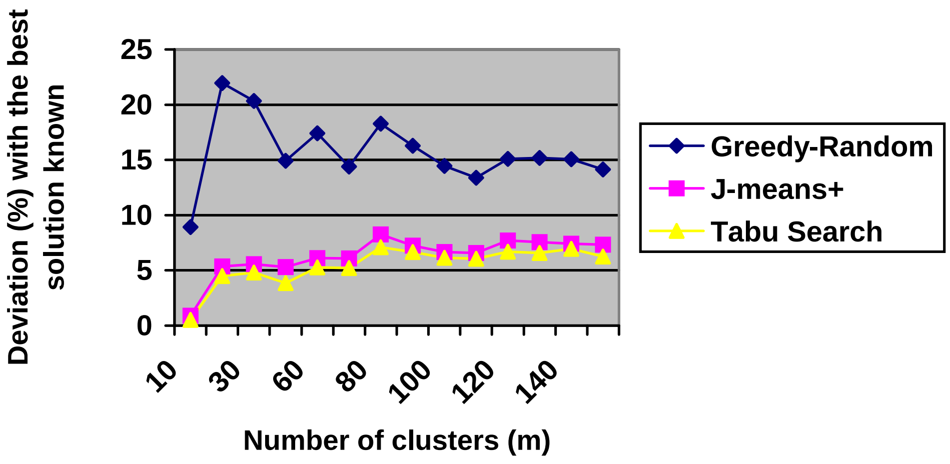

Fig. 3. Tabu Search Behavior

The following conclusions can be drawn from results of figure 3:

- As presented in Beltrán and Pacheco, (2001), the local optima obtained using greedy-random initial solutions are better than the local optima obtained with initial random centroids (SA).
- However, these local optima can still be slightly improved by exploring their neighboring regions with a simple Tabu Search method as suggested in section 3. This is interesting, since the local optima obtained by J-Means+ are local optima with respect to J-Means, H-Means, and K-Means. Thus, apparently, it would be very difficult to improve on them with solutions nearby. Moreover, the computation time employed by Tabu Search using the binary tree is very short. This allows their addition to other metaheuristics that use Local Search procedures, such as GRASP, VNS, Memetics, Scatter Search, to improve on the results and produce Hybrids of higher quality.

6.3. Genetics and Memetics: Empirical Analysis

To compare the efficiency of the proposed Genetic and Memetic algorithms, a series of tests has been performed:

- the Genetic Algorithm proposed in section 4,
- Memetic-HK: the Memetic Algorithm proposed in section 5 with HK-Means as the Local Search method,
- Memetic-J: the same Memetic Algorithm with J-Means as the Local Search method.

For each number of clusters *m* the algorithms are run once, all employing the same population of initial solutions. To obtain the elements of these initial populations, partitions are randomly generated with the SA algorithm previously described. The centroids of the corresponding clusters are used as a set of seeds. A calculation time of 30 seconds is used for all algorithms as a stop criterion. The values of the parameters are the following: *n_pop* = 10, *n_sel* = 6, and *p_mut* = 0.3. A graph with the minimal percent deviations with respect to the best-known solution is shown in figure 4.

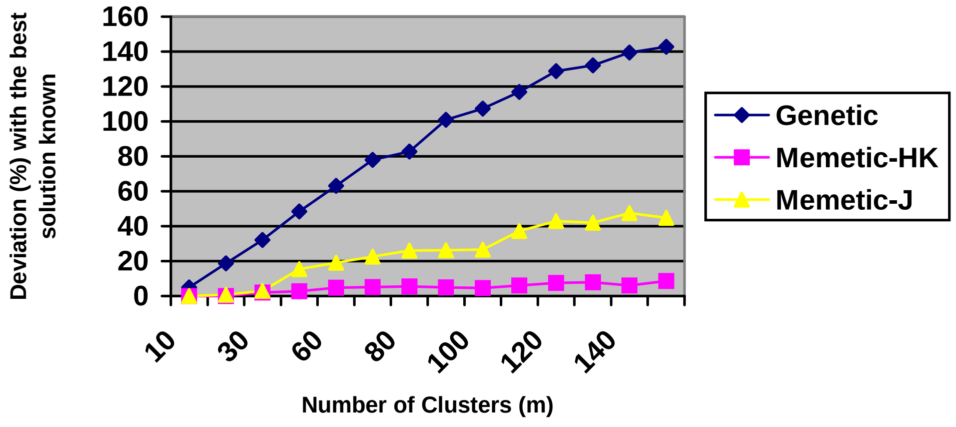

Fig. 4. Evolution of the minimal deviations

In the light of these results the following conclusions can be drawn:

- Among the evolutionary algorithms analyzed, the Memetic methods clearly perform better than the genetic ones, at least with this computational time. To obtain quality results, it is necessary to incorporate a local search procedure. Within the memetic methods, it is clear that HK-Means gives better results than J-Means. This is due to the poor functioning of the J-Means method using a start from partitions obtained by random seed points (generated by SA) as shown by Beltrán and Pacheco (2001). Therefore, in the following sections the Memetic-HK variant is used.

6.4. The proposed algorithms versus other recent strategies

To compare the efficiencies of the algorithms proposed in the previous sections and those of other recent strategies, a series of tests is performed. The following algorithms are tested:

Memetic-HK:	proposed in section 5
HybMem:	Memetic Algorithm to which Tabu Search is incorporated
GRASP-HK:	the GRASP Algorithm described in Beltrán and Pacheco (2000), with HK-Means as the Local Search method
GRASP-J:	the GRASP Algorithm with J-Means as the Local Search method
VNS-HK:	the VNS Algorithm, proposed in Hansen and Mladenovic (2001), with HK-Means as the Local Search method
VNS-J:	the VNS Algorithm, with J-Means as the Local Search method.

The GRASP strategy (*Greedy Randomize Adaptive Search Procedures*), although initially presented in the work of Feo and Resende (1989), has undergone development more recently than other metaheuristics. A thorough description can be found in a paper published in 1995 by the same authors, (Feo and Resende, 1995). *Variable Neighborhood Search* (VNS) is a recent metaheuristic method for solving combinatorial problems and is described in the works of Mladenovic (1995), Mladenovic and Hansen

(1997), and Hansen and Mladenovic, (1998). Both these recent strategies use Local Search procedures repeatedly.

Memetic-HK and HybMem use the same parameters as in section 5. For each value of *m*, the algorithms are run ten times. The Memetic-HK and HybMem algorithms use an initial population whose elements are generated as follows: partitions are generated in two phases with a greedy-random method described in Beltrán and Pacheco (2001). The centroids of the corresponding clusters are taken as sets of seeds. The VNS-HK and VNS-J algorithms use the best of these partitions as the initial solution. The two GRASP variants generate their own initial solutions. The running time is limited to 300 seconds for all the algorithms. In table 4 a series of statistics is presented corresponding to the solutions obtained for each algorithm. That is, the percent deviation with respect to the best-known solution (minimum and average). Also, the minimal deviations are presented in figure 5.

Table 4.
dmn*:* Minimal percent Deviation; ***dme****:* mean percent Deviation;

m	***Est.***	***Memetic***	***HybMem***	***GRASP-HK***	***GRASP-J***	***VNS-HK***	***VNS-J***
10	***dmn***	0	0	0.026	0.082	0	0
	dme	0	0	0.032	0.213	0.028	0.029
20	***dmn***	0	0	0.066	0.142	0	0.077
	dme	0.026	0	1.412	1.645	1.301	1.393
30	***dmn***	0.0	0.048	1.512	0.270	0.56	0.582
	dme	0.862	1.312	2.841	2.081	1.954	2.473
50	***dmn***	0.602	0.545	1.698	0.972	0.403	0.871
	dme	1.321	1.014	2.444	1.976	1.358	1.619
60	***dmn***	1.236	0.984	2.158	2.120	0.971	1.777
	dme	1.964	1.891	2.955	3.402	1.943	2.636
70	***dmn***	0.835	0.778	1.725	2.064	0.379	0.817
	dme	1.985	1.304	2.988	2.458	1.231	1.976
80	***dmn***	1.715	1.541	2.386	2.185	0.345	1.617
	dme	2.267	2.059	3.430	3.825	1.804	2.162
90	***dmn***	1.759	1.286	2.428	2.761	0.492	1.66
	dme	2.642	2.453	3.712	4.159	1.623	3.128
100	***dmn***	2.005	1.491	3.805	3.021	0.523	1.563
	dme	2.418	2.391	4.246	3.941	1.583	2.646
110	***dmn***	2.241	1.905	3.620	3.036	0.335	2.347
	dme	3.406	2.931	4.614	4.276	2.229	3.645
120	***dmn***	1.891	1.724	3.823	4.281	1.798	2.146
	dme	3.461	3.107	4.810	5.274	3.001	3.509
130	***dmn***	2.590	2.256	4.190	3.497	2.103	2.928
	dme	4.336	4.431	6.012	4.918	3.691	4.615
140	***dmn***	2.699	2.801	4.061	4.051	1.423	3.011
	dme	3.371	3.427	5.925	5.791	3.321	3.913
150	***dmn***	2.401	2.256	3.938	4.130	1.011	2.398
	dme	3.321	3.198	4.830	5.203	2.989	3.345

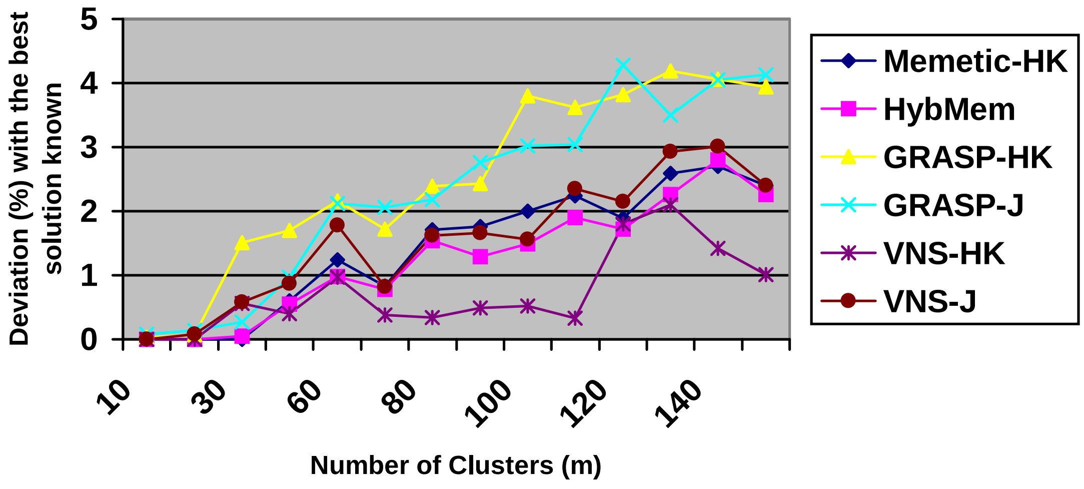

Fig. 5. Evolution of the minimal deviations (300'' run)

The following conclusions can be drawn from the previous results:

– Among the techniques analyzed, the two GRASP variants yield slightly worse results.
– Among the other four, Memetic-HK and HybMem obtain excellent results for a low number of clusters (until *m* = 50).
– For 60 or 70 clusters VNS-HK obtains the best results, both for the least and average values. Moreover, in almost all cases, Memetic-HK and HybMem yield better results than VNS-J, (particularly regarding average results).
– Finally, HybMem tends to give better results than Memetic-HK as the number of clusters increases.

7. Conclusions

Methods that give adequate solutions to the cluster design problem in short time have been proposed. The methods proposed have obtained solutions equal to the best-known solutions for small values of *m*, and only slightly worse for higher values of *m*.

In the former case, ($m \leq 50$), the new methods proposed obtain the best overall solutions from among those included in the comparison. In the latter case, they are only surpassed by the VNS algorithm proposed by Hansen and Mladenovic (2001), with HK-Means as the Local Search Procedure. However, the following is worth noting: in their work, VNS is used with J-Means and J-Means+ as a local search procedure, whereas in our work the use of J-Means in VNS leads to worse solutions than VNS with HK-Means and also to worse solutions than the methods we propose.

In future work the behavior of these new proposed methods and/or variants using long calculation times will be studied.

Acknowledgements

The authors are grateful to the Editor and two anonymous reviewers for helpful comments.

References

Al-Sultan, K.H., 1995. A Tabu Search Approach to the Clustering Problem. Pattern Recognition 28, 1443-1451.

Babu, G.P. and Murty, M.N., 1993. A Near-Optimal Initial Seed Value Selection in K-means Algorithm using Genetic Algorithms. Pattern Recognition Letters, 14, 763-769.

Beltrán, M. and Pacheco, J., 2001. Nuevos métodos para el diseño de cluster no jerárquicos. Una aplicación a los municipios de Castilla y León. Estadística Española. Instituto Nacional de Estadística. Vol. 43, nº 148, pp. 209-224.

Brucker, P., 1978. On the Complexity of Clustering Problems. Lecture Notes in Economics and Mathematical Systems 157, 45-54.

Cano, F.J., 1999. Análisis de clusters o de conglomerados. I Jornadas de Matemáticas. Burgos, Octubre 1999.

Diehr, G., 1985. Evaluation of a Branch and Bound Algorithm for Clustering. SIAM J.Sci.Statist.Comput., 6, 268-284.

du Merle, O., Hansen, P., Jaumard, B. and Mladenovic, N., 2000. An Interior Point Algorithm for Minimum Sum of Squares Clustering. SIAM Journal on Scientific Computing, 21(4):1485-1505.

Feo, T.A. and Resende, M.G.C., 1989. A Probabilistic heuristic for a computationally difficult Set Covering Problem. Operations Research Letters, 8, 67-71.

Feo, T.A. and Resende, M.G.C., 1995. Greedy Randomized Adaptive Search Procedures. Journal of Global Optimization, vol. 2, pp 1-27.

Glover, F., 1989. Tabu Search: Part I. ORSA Journal on Computing. Vol. 1, pp. 190-206.

Glover, F., 1990. Tabu Search: Part II. ORSA Journal on Computing. Vol. 2, pp. 4-32.

Glover, F. and Laguna, M., 1997. Tabu Search. Kluwer Academic Publishers, Boston.

Glover, F. and Laguna, M., 2002. Tabu Search, in Handbook of Applied Optimization, P. M. Pardalos and M. G. C. Resende (Eds.), Oxford University Press, pp. 194-208.

Goldberg, D.E., 1989. Genetic Algorithms in Search, Optimization and Machine Learning. Addison Wesley, Reading, Massachusetts.

Hansen, P. and Mladenovic, N., 1998. An Introduction to Variable Neighborhood Search. In S.Voss et al. (eds.), Metaheuristics Advances and Trends in Local Search Paradigms for Optimization, pp. 433-458, MIC-97, Kluwer, Dordrecht.

Hansen, P. and Mladenovic, N., 2001. J-Means: A new Local Search Heuristic for Minimum Sum-of-Squares Clustering. Pattern Recognition, 34 (2): 405-413.

Holland, J.H., 1975. Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor.

Howard, R., 1966. Classifying a Population into Homogeneous Groups. In Lawrence, J.R. (eds.), Operational Research in the Social Sciences. Tavistock Publ., London.

Jancey, R.C., 1966. Multidimensional Group Analysis. Australian J. Botany 14, 127-130.

Klein, R.W. and Dubes, R.C., 1989. Experiments in Projection and Clustering by Simulated Annealing. Pattern Recognition, 22, 213-220.

Koontz, W.L.G., Narendra, P.M. and Fukunuga, K., 1975. A Branch and Bound Clustering Algorithm. IEEE Transactions on Computers, C-24, 908-915

Mladenovic, N., 1995. A Variable Neighborhood Algorithm – A New Metaheuristic for Combinatorial Optimization. Abstracts of papers presented at Optimization Days, Montreal, p.112.

Mladenovic, N. and Hansen, P., 1997. Variable Neighborhood Search. Computers and Operations Research, 24, 1097-1100.

Moscato, P., 2002. Memetic Algorithms, in Handbook of Applied Optimization, P. M. Pardalos and M. G. C. Resende (Eds.), Oxford University Press, pp. 157-167.

Moscato, P. and Laguna, L., 1996. Algoritmos Genéticos, in Optimización Heurística y Redes Neuronales, DIAZ,A (coord). Paraninfo, Madrid.

Reinelt, G., 1991. TSPLIB: A Travelling Salesman Problem Library. ORSA Journal on Computing, 3, 376-384.

Späth, H., 1980. Cluster Analysis Algorithms for Data Reduction and Classification of Objects. Ellis Horwood, Chichester.

Williams, J.W.J., 1964. Algorithm 232: HeapSort. Commun. A.C.M. 7, 347.