Pergamon

Pattern Recognition, Vol. 28, No. 9, pp. 1443–1451, 1995
Elsevier Science Ltd
Copyright © 1995 Pattern Recognition Society
Printed in Great Britain. All rights reserved
0031-3203/95 $9.50 + .00

0031-3203(95)00022-4

A TABU SEARCH APPROACH TO THE CLUSTERING PROBLEM

KHALED S. AL-SULTAN
Department of Systems Engineering, King Fahd University of Petroleum and Minerals, Dhahran 31261, Saudi Arabia

(*Received* 20 *June* 1994; *in revised form* 1 *December* 1994; *received for publication* 9 *March* 1995)

Abstract—In this paper we consider the problem of clustering m objects into c clusters. The objects are represented by points in an n-dimensional Euclidean space, and the objective is to classify these m points into c clusters such that the distance between points within a cluster and its center (which is to be found) is minimized. The problem is a nonconvex program that has many local minima. It has been studied by many researchers and the most well-known algorithm for solving it is the k-means algorithm. In this paper, we develop a new algorithm for solving this problem based on a tabu search technique. Preliminary computational experience on the developed algorithm are encouraging and compare favorably with both the k-means and the simulated annealing algorithms.

Clustering problem Tabu search k-means algorithm Simulated annealing Nonconvex programming Global optimum

1. INTRODUCTION

Cluster analysis could be defined as the process of separating a set of objects (patterns) into groups (classes and clusters) such that members of one group are similar, as much as possible, according to a predefined criterion. Cluster analysis has found applications in life, medical, behavioral and social sciences. Other applications also include earth sciences, information and decision sciences and engineering. For a summary of these applications, see Al-Sultan.[1]

In cluster analysis, the objects are represented by points in n-dimensional Euclidean space, where the elements of the vector are values for the attributes of the object and the objective is to classify these m points into c clusters such that a certain similarity measure is optimized. In this paper, we consider our similarity measure (function) to be the distance between points within a cluster and its center, and the objective is to minimize that function.

In this paper we develop an efficient algorithm based on a tabu search technique for the clustering problem. The paper is organized as follows: in Section 2 the statement of the problem and literature survey are presented. In Section 3 we summarize the tabu search technique. The statement of tabu search algorithm is presented in Section 4, and its implementation details are discussed in Section 5. Results and discussion are presented in Section 6. Finally, we give our conclusion in Section 7.

2. STATEMENT OF THE PROBLEM AND LITERATURE REVIEW

The clustering problem may be stated as follows: Given m patterns in R^n, allocate each pattern to one of c clusters such that the sum of squared Euclidean distances between each pattern and the center of the cluster (which is also to be found) to which it is allocated is minimized.

The clustering problem can be mathematically described as follows:

$$\left.\begin{aligned}&\text{Minimize}\quad J(w,z)=\sum_{i=1}^{m}\sum_{j=1}^{c} w_{ij}\|x_i-z_j\|^2\\ &\text{Subject to}\quad \sum_{j=1}^{c} w_{ij}=1,\quad i=1,2,\ldots,m\\ &\text{and}\quad w_{ij}=0\text{ or }1,\quad i=1,2,\ldots,m,\quad j=1,2,\ldots,c\end{aligned}\right\}\qquad\text{(CP)}$$

where:

c number of clusters (given);
m number of available patterns (given);
$x_i \in R^n$ location of the ith pattern (given), $i \in [1,2,\ldots,m]$;
$z_j \in R^n$ center of the jth cluster (to be found), $j \in [1,2,\ldots,c]$;
w_{ij} association weight of pattern x_i with cluster j (to be found);

where

$$w_{ij} = \begin{cases} 1 & \text{if pattern } i \text{ is allocated to cluster } j \\ & \forall i = 1,2,\ldots,m, \quad j = 1,2,\ldots,c \\ 0 & \text{otherwise} \end{cases}$$

z is an $m \times c$ matrix whose column j is z_j defined above;

w:$[w_{ij}]$ is an $n \times c$ matrix;

$\|x_i - z_j\|^2$ squared Euclidean distance between pattern x_i and center z_j of cluster j.

In this formulation, the squared Euclidean distance has been taken as the criterion function to be minimized.

Problem CP is a nonconvex program which possesses many local minima. The clustering problem has been investigated by many researchers. The major difficulty is that the solution of problem CP could possibly lead to local optima which are not necessarily global. The following is a summary of the work done so far on this problem:

- The k-means algorithm[2,3] is the most widely used method to solve the problem. However, it may fail to obtain the global solution of the problem. Selim and Ismail[4] derived a generalized convergence theorem for the k-means algorithm in which they characterized the conditions for local optimality of the solution obtained by that algorithm.
- Jensen[5] used dynamic programming to solve the above problem with the assumption that cluster centers coincide with some patterns. This approach is not optimal and it requires a large computer memory space and time.
- Vindo[6] and Rao[7] formulated the clustering problem as an integer linear program with the same assumption as in Jensen. This approach has the same disadvantages as in Jensen's approach.
- Duda and Hart[8] developed the Isodata algorithm to solve the clustering problem. Although the isodata algorithm is similar to the k-means algorithm in the sense that it may fail to obtain the global optimal solution, it is more complex and more expensive to implement. However, it has more options which are not needed in the solution of our problem.
- Cooper[9] was the first to formulate the clustering problem as a location–allocation problem. For the solution of the problem, Cooper suggested two heuristic which could achieve local optimal solutions.
- Selim[10] developed an algorithm that guarantees that a global solution will be obtained. The algorithm transforms the criterion function into an equivalent implicit concave function. Then the concept of convexity cuts is implemented to obtain the optimal solution. Although this algorithm is exact, it requires much computational time and a large memory. Since the cutting plane approach is known to be very slow for solving linear integer programming, the case will even be worse for the clustering problem which is a nonconvex program.
- Koontz *et al.*[11] employed the branch and bound approach to solve the problem. Diehr[12] modified Koontz algorithm to be more efficient. However, the algorithm is impractical for large problems. The Branch and Bound approach, although guarantees global optimality, is known to be a last resort for solving optimization problems because of the long and unpredictable time it takes to obtain the solution.
- Klein and Dubes[13] and Selim and Al-Sultan[14] have developed simulated annealing-based algorithms for solving the clustering problem. However, no comparison of the performance of their algorithms to the k-means algorithm have been provided.

In this paper, we develop an algorithm for CP based on a tabu search technique. We compare the performance of our algorithm with that of the k-means and the simulated annealing algorithms. However, we start by briefly describing the tabu search technique.

3. THE TABU SEARCH TECHNIQUE

The tabu search is a meta heuristic that can be used to solve combinatorial optimization problems. It is different from the well-known hill-climbing local search techniques in the sense that it does not become trapped in local optimal solutions, i.e. the tabu search allows moves out of a current solution that makes the objective function worse in the hope that it eventually will achieve a better solution.

The tabu search requires the following basic elements to be defined:

- *Configuration* is a solution or an assignment of values to variables.
- *A move* characterizes the process of generating a feasible solution to the combinatorial problem that is related to the current solution (i.e. a move is a procedure by which a new (trial) solution is generated from the current one).
- *Set of candidate moves* is the set of all possible moves out of a current configuration. If this set is too large, one could operate with a subset of this set.
- *Tabu restrictions*: These are certain conditions imposed on moves which make some of them forbidden. These forbidden moves are known as *tabu*. It is done by forming a list of a certain size that records these forbidden moves. This is called *tabu list*.
- *Aspiration criteria*: These are rules that override tabu restrictions, i.e. if a certain move is forbidden by tabu restriction, then the aspiration criterion, when satisfied, can make this move allowable.

Given the above basic elements, the tabu search scheme can be described as follows: start with a certain (current) configuration, evaluate the criterion function for that configuration. Then, follow a certain set of candidate moves. If the best of these moves is not tabu or if the best is tabu, but satisfies the aspiration criterion, then pick that move and consider it to be the new current configuration; otherwise, pick the best move that is not tabu and consider it to be the new current configuration. Repeat the procedure for a certain number of iterations. On termination, the best solution obtained so far is the solution obtained by the algorithm. Note that the move that is picked at a certain iteration is put in the tabu list so that it is not allowed to be reversed in the next iterations. The tabu list has a certain size, and when the length of the tabu reaches that size and a new move enters that list, then the first move on the tabu list is freed from being tabu and the process continues (i.e. the tabu list is circular). The aspiration criterion could reflect the value of the criterion (objective) function, i.e. if the tabu move results in a value of the criterion function that is better than the best known so far, then the aspiration criterion is satisfied and the tabu restriction is overridden by this.

In the above paragraph, we have outlined the basic steps of the tabu search procedure for solving combinatorial optimization problems. For more elaboration on the tabu search technique as well as a list of successful applications, see Glover.[15] In the next section, we develop a new algorithm for solving CP based on the above tabu search technique.

4. THE NEW ALGORITHM

In this section, we present our tabu search-based algorithm for problem CP. However, before we state our algorithm, we need to introduce some notation:

Let A be an array of dimension m whose ith element (A_i) is a number representing the cluster to which the ith pattern is allocated. Clearly, given A, all w_{ij}s are defined in the following manner:

$$w_{ij} = \begin{cases} 1 & \text{if } A_i = j \\ 0 & \text{otherwise} \end{cases}$$

for all $i = 1, 2, \ldots, m$, and $j = 1, 2, \ldots, c$.

For example, consider the following array A for $m = 10$, $c = 3$.

3	2	1	2	2	3	1	1	2	3

then the first pattern is assigned to the third cluster (i.e. $w_{11} = 0$, $w_{12} = 0$, $w_{13} = 1$), and the second pattern is assigned to the second cluster (i.e. $w_{21} = 0$, $w_{22} = 1$ and $w_{23} = 0$) and so on.

Given a set of w_{ij}s, the center of each cluster z_i can be computed as the centroid of the patterns allocated to that cluster or:

$$z_j = \frac{\sum_{i=1}^{m} w_{ij} x_{ij}}{\sum_{i=1}^{m} w_{ij}}. \tag{1}$$

Given the array A, and the centers z_j, one can compute the objective function $J(w, z)$ as:

$$J(w, z) = \sum_{i=1}^{m} \sum_{j=1}^{c} w_{ij} \|x_i - z_j\|^2. \tag{2}$$

Thus, from the above it is clear that for any array A (configuration), a specific value corresponds to the objective function which, for simplicity, is denoted by J. Therefore, we will depend in our tabu search algorithm on changing the configuration A, and consequently it will map into a value for the objective function J. We will operate with three arrays and their corresponding three objective function values.

Let A_t A_c and A_b denote the trial, current and best arrays and J_t J_c and J_b denote the corresponding trial, current and best objective function values, respectively.

We will always operate with a configuration which we call the current solution A_c and then through moves which were explained in the last section, we generate trial solutions A_t. As the algorithm proceeds, we also save the best solution found so far which is denoted by A_b. Corresponding to these arrays, we also operate with the objective function values J_c, J_t and J_b, respectively.

We are now ready to state our algorithm.

4.1. *Statement of the algorithm*

(1) *Initialization step.* Let A_c be an arbitrary solution, and J_c be the corresponding objective function value computed using equations (1) and (2). Let $A_b = A_c$, and $J_b = J_c$. Select values for the following parameters: MTLS (tabu list size), P (probability threshold), NTS (number of trial solutions) and let ITMAX be the maximum number of iterations. Let $k = 1$, let TLL (tabu list length) = 0 and go to step 2.

(2) Using A_c, generate NTS trial solutions $A_t^1, A_t^2, \ldots, A_t^{\mathrm{NTS}}$ (see the remark below) and evaluate their corresponding objective function values $J_t^1, J_t^2, \ldots, J_t^{\mathrm{NTS}}$, and go to Step 3.

(3) Order $J_t^1, J_t^2, \ldots, J_t^{\mathrm{NTS}}$ in an ascending order, and denote them by $J_t^{[1]}, J_t^{[2]}, \ldots, J_t^{[\mathrm{NTS}]}$. If $J_t^{[1]}$ is not tabu, or if it is tabu but $J_t^{[1]} < J_b$, then let $A_c = A_t^{[1]}$ and $J_c = J_t^{[1]}$, and go to step 4; otherwise let $A_c = A_t^{[L]}$, $J_c = J_t^{[L]}$ where $J_t^{[L]}$ is the best objective function of $J_t^{[2]}, J_t^{[3]}, \ldots, J_t^{[\mathrm{NTS}]}$ that is not tabu and go to step 4. If all $J_t^{[1]}, J_t^{[2]}, \ldots, J_t^{[\mathrm{NTS}]}$ are tabu go to step 2.

(4) Insert A_c at the bottom of the tabu list and let TLL = TLL + 1 (if TLL = MTLS + 1, delete the first element in the list and let TLL = TLL − 1). If $J_b > J_c$ let $A_b = A_c$ and $J_b = J_c$. If k = ITMAX, stop (A_b is the best solution found and J_b is the corresponding best objective function value); otherwise, let $k = k + 1$ and go to step 2.

4.2. *Remark*

Given a current solution A_c, one can generate a trial solution using several strategies. We use the following strategy: Given A_c and a probability threshold P, for $i = 1, m, \ldots,$ draw a random number $R \sim u(0, 1)$. If $u(0, 1) < P$, then $A_t(i) = A_c(i)$; otherwise draw randomly an integer $\hat{l}$ from the following set $\{l : l = 1, 2, 3, \ldots, c, l \neq A_c(j)\}$ and let $A_c(i) = \hat{l}$.

5. IMPLEMENTATION OF THE ALGORITHM

The algorithm presented in Section 4 has been coded in C, and tested on a IBM 386 machine on several test problems. We have used the following two sets of test problems.

5.1. *Random test problems* (*RS*).

These are randomly generated test problems in R^2 where the data are generated within spheres of the same radii. The overlap among clusters is controlled through changing the radius of the spheres. The following classes of test problems are used:

RSO: Clusters are overlapping.
RST: Clusters are touching each other.
RSS: Clusters are nontouching (separated).

5.2. *Standard test problems* (*SS*).

These are standard data sets.[16]

SS1: gives 89 postal zones in Bavaria (Germany), and their three attributes as their surface area in squared kilometers, their populations, and the density of the population [see reference (16), pp. 91, 92]

SS2: gives the same 89 postal zones Bavaria (Germany), but with four attributes which are: the numbers of self-employed people, civil servants, clerks and manual workers [see reference (16), pp. 103, 104].

As it is clear from its statement, the tabu search algorithm has three parameters which need to be assigned values before the execution of the algorithm. These parameters are:

MTLS: tabu list size
P: probability threshold
NTS: number of trial solutions

We have performed our implementation in two phases. In the first phase, an extensive parametric study has been performed to find the best parameter settings. These best values were then used in the second phase for testing the performance of the algorithm on various random and standard problems.

6. RESULTS AND DISCUSSION

As mentioned in the previous section, the tabu search algorithm has three parameters. In this section, we discuss the theoretical implications of the values of these parameters, and the best values obtained for them by our extensive parametric study.

6.1. *Tabu list size* (*MTLS*)

The tabu list embodies one of the functions of the short term memory, which enables the algorithm to have some memory of the history. In our case, it tells the algorithm not to reverse the last MTLS moves, where MTLS is the size of the tabu list. Therefore, the size of the tabu list here affects how much memory one would like the algorithm to have. A large tabu list size allows more diversification or forcing new moves to be generated which take solution to far points. A small list size, on the other hand, makes the algorithm more forgetful, and therefore allows movement within the region or allows intensification to happen (not forcing it because this means more control on the moves). Glover[15] suggests using a tabu list size in the range $[\frac{1}{3}n, 3n]$, where n is related to the size of the problem. In our case, n is the number of patterns.

6.2. *Probability threshold* (*P*)

This parameter controls the shake-up that is performed on a certain solution to produce a neighbor. The higher the value of P, the less shake-up is allowed and consequently the closer the neighbor (the solution obtained after the move) to the current solution, and *vice versa*. In our case, we have found that $P = 0.95$, or very little shake-up for the move produces better results. This means that we allow more intensification in the neighborhood to generate the next best move. One should remember that the tabu list will record this move and the search may not return to it in the next few iterations or it may be noise to explore the close neighborhood fully before one leaves it. (Figs. 1 and 2 show a sample the current and trial objective function values obtained by the algorithm versus number of iterations for several values of P, respectively).

6.3. *Number of trial solutions* (*NTS*)

This parameter controls the number of trial moves to be generated from a current one to decide on the next move. Clearly, the larger the value of NTS the better, simply because one has more choices to select from. However, this is done at the expense of more computational effort. Therefore, if one were to decide *a priori* on a certain amount of computational effort to be made by the algorithm, then larger values of NTS at each iteration means less number of iterations are made, or intensification is made at the expense of less diversification and *vice versa*.

We have tested our algorithms for various values of MTLS, P and NTS on 10 problems of each type of test problems and for various combinations of m and c. Tables 1–3 are just examples of the extensive parametric study that has been performed. It is clear that the algorithm performs better as the probability threshold increases with the best results obtained at $P = 0.95$. We have also found that the best values for both MTLS and NTS are in the range 15–20. We might add that the three values for each parameter in Tables 1–3 are the best values of the parameters found after an initial parametric study that has included various other values for them. The algorithm, in general, obtains good results for a wide range of parameter values, i.e. it is relatively robust.

Tables 4–6 compare the objective function values (average of 10 problems for each combination) obtained by our algorithm for the random test problems to those obtained by our coding of the k-means algorithm.[2] Tables 7 and 8 compare the objective function values obtained by our algorithm for the standard test problems to those obtained by our coding of both the k-means algorithm and the simulated annealing.

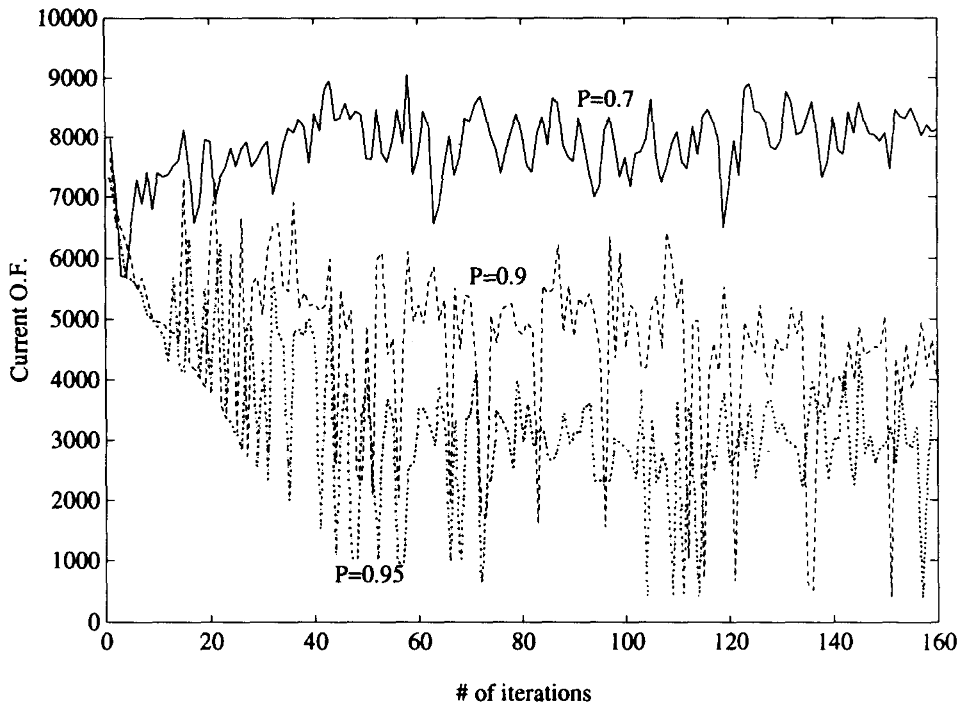

Fig. 1. Effect of probability threshold value *P* on the values of the current objective function.

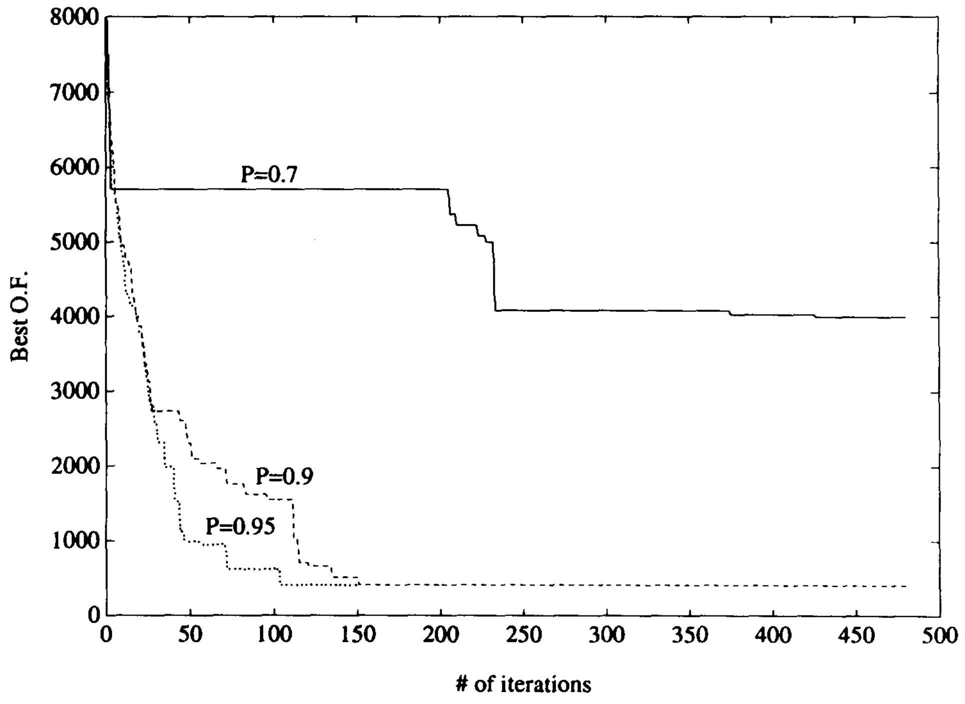

Fig. 2. Effect of probability threshold value *P* on the values of the best objective function.

The following points are clear from the tables:

(a) Our algorithm outperforms the k-means algorithm in almost all cases and by a factor ranging from 10 to 99.5%.

(b) In general, for a certain number of patterns m, the larger the number of clusters c, the more pronounced the improvement of our algorithm over the k-means. This is of course clear in the standard test problems where we use the same data with different values of c ($c = 2, 3, 4, 5$). This trend is in general clear for the random problems, with few exceptions. The reason for these exceptions might be that we are not using the same data with different values of c (as in standard data sets), but rather the seed for the random number generator might be different from one problem to another, and hence generates different data. Our interpretation of this difference in performance is that the problem becomes more difficult and may have more local minima as c increases for the same value of m. The number of variables for the problem is mc, and therefore is proportional to the value of c for a fixed number of m. Being a local search method, the k-means is more likely to become stuck at a local minimum that

Table 1. Parametric study on test problems RSS, $c = 2, m = 40$

	MTLS 15		MTLS 20	
P	NTS = 15	NTS = 20	NTS = 15	NTS = 20
0.7	1847.58[(a)]	1843.10	1760.93	1652.47
0.9	529.31	529.31	529.31	668.02
0.95	529.31	600.60	529.31	529.31

(a)
$$\frac{\sum_{i=1}^{10} J_a^i}{10},$$
where J_a^i is the objective function value obtained for problem i by our algorithm.

Table 2. Parametric study on test problems RST, $c = 4, m = 60$

	MTLS 15		MTLS 20	
P	NTS = 15	NTS = 20	NTS = 15	NTS = 20
0.7	13974.8[(a)]	13559.17	12226.72	13206.64
0.9	5426.01	5104.26	5723.05	3773.02
0.95	3648.53	3653.94	3649.83	3773.10

(a)
$$\frac{\sum_{i=1}^{10} J_a^i}{10},$$
where J_a^i is the objective function value obtained for problem i by our algorithm.

Table 3. Parametric study on test problems RSO, $c = 2, m = 40$

	MTLS 15		MTLS 20	
P	NTS = 15	NTS = 20	NTS = 15	NTS = 20
0.7	7667.77[(a)]	7194.93	7609.74	7401.7
0.9	6466.56	6466.56	6474.77	6466.57
0.95	6464.8	6466.56	6657.97	6464.82

(a)
$$\frac{\sum_{i=1}^{10} J_a^i}{10},$$
where J_a^i is the objective function value obtained for problem i by our algorithm.

Table 4. Comparison of results obtained by our algorithm and the k-means algorithm for test problems RSS

c	m	Our algorithm	k-means algorithm
2	10	102.83[(a)]	102.837[(b)]
		100.00[(c)]	100.00
	40	529.30	529.30
		100.00	100.00
3	15	157.03	275.39
		57.02	100.00
	60	796.86	1281.39
		62.19	100.00
4	20	217.99	439.66
		49.58	100.00
	40	507.74	2872.5
		18.44	100.00
5	20	189.15	1039.19
		18.99	100.00
	40	477.14	1426.67
		60.82	100.00

a
$$\frac{\sum_{i=1}^{10} J_a^i}{10},$$
where J_a^i is the objective function value obtained for problem i by our algorithm.

b
$$\frac{\sum_{i=1}^{10} J_k^i}{10},$$
where J_k^i is the objective function value obtained for problem i by the k-means algorithm.

c
$$\frac{\sum_{i=1}^{10}\left(\frac{J_a^i}{J_k^i}\right)}{10} \times 100\%.$$

Table 5. Comparison of results obtained by our algorithm and the k-means algorithm for test problem RST

c	m	Our algorithm	k-means algorithm
2	10	494.87	519.43
		96.57	100.00
	40	2531.01	2544.90
		99.50	100.00
3	15	750.64	750.64
		100.00	100.00
	60	3810.97	3829.75
		99.51	100.00
4	20	1010.76	1112.43
		93.31	100.00
	40	2362.58	2872.51
		85.59	100.00
5	20	844.89	1039.19
		84.59	100.00
	40	2159.57	2461.72
		89.23	100.00

is far from the global minimum as the problem becomes harder. This is not the case for our algorithm as explained later.

(c) For the standard data sets, our algorithm outperforms the k-means by a large factor and this difference in performance is more pronounced than for the random test sets. The reason is that the k-means is designed for spherical cluster shapes and our random

Table 6. Comparison of results obtained by our algorithm and the k-means algorithm for test problems RSO

c	m	Our algorithm	k-means algorithm
2	10	1235.36	1322.86
		94.74	100.00
	40	6464.82	6489.08
		99.65	100.00
3	15	1538.31	1815.21
		85.08	100.00
	60	8841.98	9296.26
		95.11	100.00
4	20	1908.33	2320.56
		83.10	100.00
	40	4557.96	4776.23
		95.90	100.00
5	20	1491.29	1814.87
	40	84.14	100.00
		3697.59	3920.25
		94.86	100.00

Table 7. Comparison of results obtained by our algorithm, the k-means algorithm and the simulated annealing algorithm for test problems SS1

c	m	Our algorithm	k-means algorithm	Simulated annealing
2	89	6.03×10^{11}	6.49×10^{11}	6.03×10^{11}
		92.81	100.00	92.81
3	89	3.64×10^{10}	3.64×10^{11}	3.64×10^{11}
		10.00	100.00	100.00
4	89	1.05×10^{11}	2.79×10^{11}	3.18×10^{11}
		37.63	100.00	114.22
5	89	6.78×10^{10}	2.60×10^{11}	7.82×10^{10}
		26.06	100.00	30.09

Table 8. Comparison of results obtained by our algorithm, the k-means algorithm and the simulated annealing algorithm for test problems SS2

c	m	Our algorithm	k-means algorithm	Simulated annealing
2	89	1.99×10^{10}	4.86×10^{10}	4.86×10^{10}
		40.94	100.00	100.00
3	89	1.74×10^{10}	3.60×10^{10}	1.74×10^{10}
		48.36	100.00	48.36
4	89	7.56×10^{9}	3.05×10^{10}	8.04×10^{9}
		24.77	100.00	26.34
5	89	5.95×10^{9}	2.95×10^{10}	6.47×10^{9}
		20.16	100.00	21.93

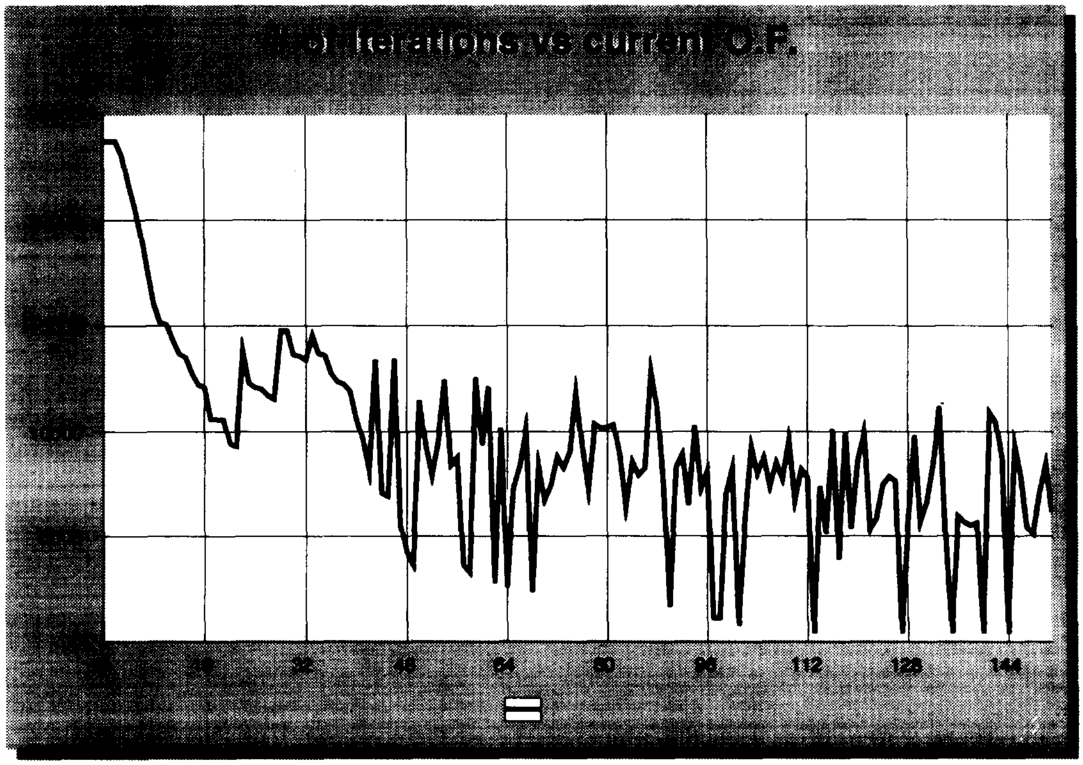

Fig. 3. Values of the current objective function vs number of iterations using the best parameter settings.

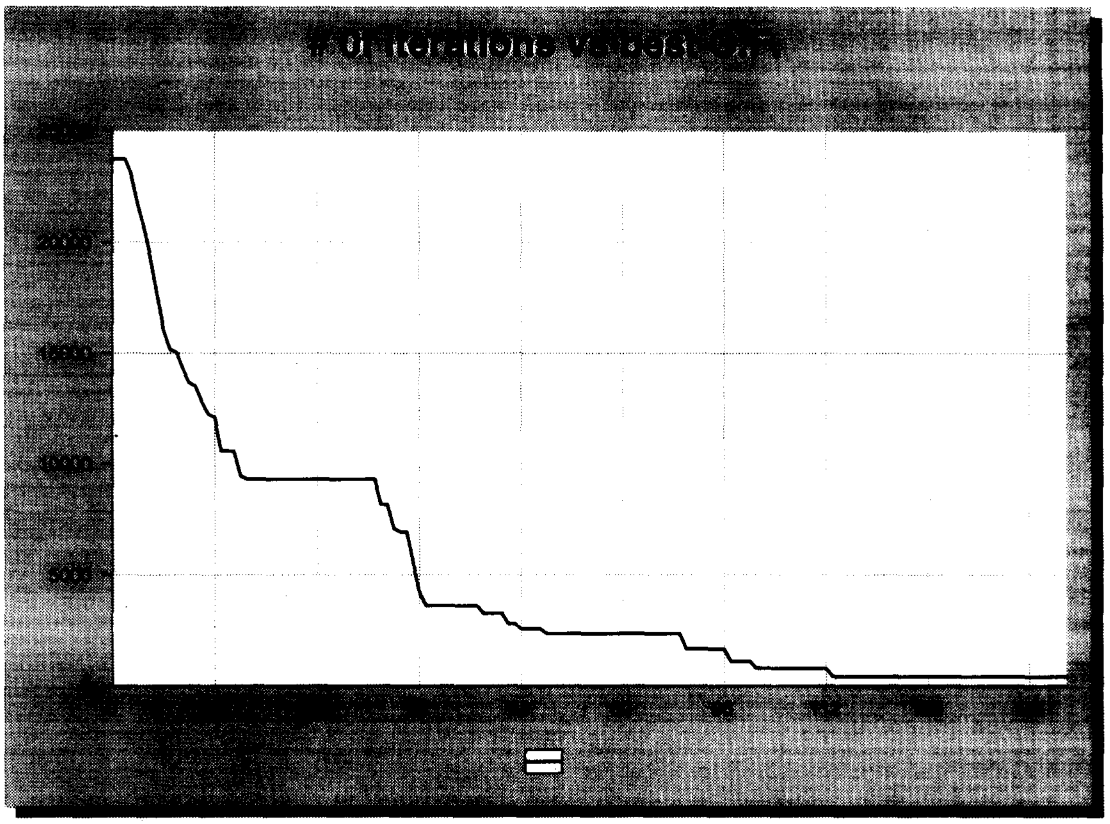

Fig. 4. Values of the best objective function versus number of iterations using the best parameter settings.

data are generated within spheres, while this is not the case for the standard data.

(d) Our algorithm has obtained results that are equal to those obtained by the simulated annealing algorithm in two out of the eight problem sets tested (see Tables 7 and 8), while it outperformed it in the remaining six sets and by a factor that is as large as 10% in some cases. More importantly is the fact that our algorithm is much more robust and consistent in its performance compared with the simulated annealing algorithm. The reason for this lack of robustness in the simulated annealing algorithm is its sensitivity to some parameters, especially the initial temperature and the cooling schedule while our algorithm is not too sensitive to parameters values. In the literature, some results of simulated annealing are deceiving because researchers usually report the best results obtained by the simulated annealing algorithm by using the best parameter settings for the data set which may be different from the settings used to obtain the best values for another set of data. An extensive initial parametric study should be performed on a large number of problems, and the values that seem to be best for most of the problems should then be picked. Then, fix these values for comparison purposes. This is what has been done in both simulated annealing and tabu search algorithms. (If this were not the right approach, then any result obtained by one of these algorithm is actually not a result of a single run of the algorithm, but rather several runs to find the best parameter settings and one run to obtain the solution using these parameter values, which is of course very costly).

(e) Fig. 3 is a typical graph of the values obtained for the current objective function values *versus* the number of iterations. It is clear that uphill moves are allowed and this is what makes the algorithm powerful in the sense that it does not become stuck at a local minimum, but rather allows some uphill moves in the hope that they will lead to better objective function values.

Figure 4 is a graph of the best objective function of values obtained *versus* the number of iterations. As is clear from the graph, the best value stays the same for some (random) number of iterations and suddenly drops when a downhill move is obtained.

7. CONCLUSIONS

In this paper, we have developed a new algorithm for solving the clustering problem which is based on the tabu search technique. The algorithm has been implemented and tested on various problems, and preliminary computational experience is very encouraging. As a matter of fact, our algorithm obtained results that are better than the well-known k-means and the simulated annealing algorithms for many test problems.

Acknowledgements—I would like to thank Mr M. M. Khan for many discussions and for his assistance in obtaining the computational results. The author is grateful to an anonymous referee for several comments that have improved the paper.

REFERENCES

1. K. S. Al-Sultan, Efficient global algorithms for Hand and Fuzzy Clustering, unpublished M.S. thesis, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, (1987).
2. E. Forgy, Cluster analysis of multivariate data: efficiency versus interpretability of classifications, *Biometrics* **21**, 768 (abstract) (1965).
3. J. B. McQueen, Some methods of classification and analysis of multivariate observations, *Proc. 5th Berkeley Symp. Mathemat. Statist. Probability*, pp. 281–297. University of California Press, Berkeley U.S.A.
4. S. Z. Selim and M. A. Ismail, K-Means-type algorithm: generalized convergence theorem and characterization of local optimality, *IEEE Trans. Pattern Anal. Mach. Intell.* **6**, 81–87 (1984).
5. R. E. Jensen, A dynamic programming algorithm for cluster analysis, *Appl. Statist.* **28**, 1034–1057 (1969).
6. H. D. Vindo, Integer Programming and the theory of grouping, *J. Am. Statist. Soc.* **64**, 506–519 (1969).
7. M. R. Rao, Cluster analysis and mathematical programming, *J. Am. Statist. Ass.* **66**, 622–626 (1971).
8. R. O. Duda, P. E. Hart, *Pattern Classification and Scene Analysis*. John Wiley & Sons, New York (1973).
9. L. Cooper, N-dimensional location models: applications to cluster analysis, *J. Regional Sci.* **13**, 41–54 (1973).
10. S. Z. Selim, A global algorithm for the clustering problem, *ORSA/TIMS*, San Diego (1982).
11. W. L. Koontz, P. M. Narendra and K. Fukunaga, A branch and bound clustering algorithm, *SIM J. Sci. Statist. Comput.* **6**(2), (April 1985).
12. G. Diehr, Evaluation of a branch and bound algorithm for clustering, *IEEE Trans. Comput.* **C24**(9), (September 1975).
13. R. W. Klein and R. C. Dubes, Experiments in projection and clustering by simulated annealing, *Pattern Recognition* **22**, 213–220 (1989).
14. S. Z. Selim and K. S. Al-Sultan, A simulated annealing algoroithm for the clustering problem, *Pattern Recognition* **24**(10), 1003–1008 (1991).
15. F. Glover, Artificial intelligence, heuristic frameworks and tabu search, *Manag. Decis. Econom.* **11**, 365–375 (1990).
16. H. Spath, *Cluster Analysis Algorithms for Data Reduction and Classification of Objects*. Ellis Horwood Limited, West Sussex, U.K. (1980).

About the Author—KHALED S. AL-SULTAN was born in Buridah, Al-Gassim, Saudi Arabia, in 1963. He received his B.S. and M.S. degrees from the King Fahd University of Petroleum and Minerals (KFUPM), Dhahran, Saudi Arabia, in 1985 and 1987, respectively. He earned his Ph.D. degree from the University of Michigan, Ann Arbor, U.S.A., in Industrial and Operations Engineering in 1990. Since 1990, he has been a faculty member in the Department of Systems Engineering at KFUPM, and currently he is the Chairman of the Department. Dr Al-Sultan is a registered professional engineer in the State of Michigan and is a senior member of the Institute of Industrial Engineers, the Operations Research Society of America and the Mathematical Programming Society. Dr Al-Sultan's research interests are in the areas of mathematical programming, cluster analysis, quality control and design and analysis of algorithms.