Proceedings of International Conference on Image
Processing and Character Recognition

Computational Experience About Clustering Using Tabu Search*

Wu-Ja Lin and Ja-Chen Lin

Department of Computer and Information Science

National Chiao-Tung University

Hsinchu, Taiwan

gis82802@cis.nctu.edu.tw

Abstract

In this paper we present our experience about using tabu search (T.S.) to cluster data. Experimental results show that the T.S. approach is in general superior to the k-means method and simulated annealing (S.A.) method.

1 Introduction

Clustering has beed developed for several decades and it can be applied to many fields such as medical, behavioral and politics [1]. Clustering can be considered a process that partitions data into several clusters so that every point in the same cluster is similar to each other. One criterion that is commonly used to measure the goodness of clustering results is the sum of square error (SSE) defined as

$$\mathrm{SSE} = \sum_{S_j} \sum_{x \in S_j} \|x - c_j\|^2 , \qquad (1)$$

where x are data points, and c_j is the centroid of the generated cluster S_j. Many clustering methods [2] developed so far intended to minimize SSE. However, it is difficult in general to find the global minimum for SSE. An increasing trend is to apply some heuristic search methods [4][7] to accomplish it. T.S. is a heuristic search method and is receiving increasing attension for its good performance on the applications to many fields [3]. In this paper, we apply T.S. to clustering data and compare it with the k-means and S.A. method.

*This work was supported by National Science Council, Republic of China, under contract number NSC85-2221-E-009-012

2 A review of tabu search (T.S.)

Before introducing the proposed method, we briefly describe the tabu search method. It is a heuristic search method which not only avoids exploring previously searched paths but also tries to jump out of the traps of local optimum [8] [9]. T.S. has received much attention in the solving of scheduling problems. Basically, T.S. is a neighborhood search method (stated below) [10] and it has a memory structure to store the moves (Step 3) previously made in the searching process. Two rules, *restriction* and *aspiration*, are used in T.S. to direct the searching.

Neighborhood search procedure:

Step 1: Start searching from an arbitrarily chosen solution. Call this solution the current solution.

Step 2: Generate a set of solutions from the neighbors of the current solution.

Step 3: Move the current solution to a neighbor (chosen by a selection criterion). In other words, update the current solution using that neighbor.

Step 4: Repeat steps 2 $\sim$ 3 until a predetermined termination criterion is met. □

The restriction rule of T.S. stipulates that the new move in Step 3 must not be found in the memory. That is, it forbids the moves that had ever been made previously. If the move is found in the memory structure, one should choose another neighbor from the remaining ones whose moves are not forbidden and yields the next best value. Then move current solution to it. (Once the moves from the current solution to the remaining neighbors are all forbidden, one should re-generate another set of neighbors of the current solution and repeat the same examination to make the next move.) On the other hand, the aspiration rule states that the restriction rule could be overridden if the forbidden move can reach a solution which is better than the best one that had ever been found so far.

Note that the size of the memory structure used by T.S. determines the capacity of memorizing the his-

torical moves. The larger the size is, the more moves the memory can memorize. In practice, the memory size is not infinite. Therefore, the contents of the memory must be consecutively updated during the searching process so that the memory always stores the moves recently made. The major idea of using a memory structure in T.S. is to avoid the possibility of exploring searched solutions so that the diversity of the explored solutions can increase. This diversity is helpful for the effectiveness of finding the globally optimal solution. We adopt this strategy in our method to solve clustering problem. The details of our method are introduced in the next section.

3 The proposed T.S. method

We solve clustering problems by searching for a set of specific points (called class centers) and using these points to partition the given data sets. Each set of class centers will be called a possible solution. The major idea of our method is to apply T.S. to search for a good solution, i.e., a set of class centers that yields a good partition of data. Before applying T.S. to searching for a good solution, we first encode each possible solution as a binary string using the method described in [4]. In other words, given a set of d-dimensional data points, we subsampe the range $(\mathrm{MIN}_D, \mathrm{MAX}_D)$ of every dimension D ($D = 1, 2, \ldots, d$) into, say, 2^ζ values (the sampling step is $\frac{\mathrm{MAX}_D - \mathrm{MIN}_D}{2^\zeta - 1}$ and the subsampled values are MIN_D, $\mathrm{MIN}_D + \frac{\mathrm{MAX}_D - \mathrm{MIN}_D}{2^\zeta - 1}$, ..., MAX_D). Here MIN_D and MAX_D is the minimal and maximal value of input points on the Dth dimension. Therefore, we can use ζ bits to represent the 2^ζ subsampled values for each dimension, and hence, we can use $\zeta \times d$ bits to represent a d-dimensional point. When proceeding our searching to a better solution, we only consider data points (d-dimension) that can be represented with these $\zeta \times d$ bits. The details of the method are described in the following.

ALGORITHM

Step 1: Assign desired values to T.S. parameters: λ (number of candidate solutions), ω (memory size), P_r (the disturbing rate used to generate candidate solutions from current solution), TIN (total iteration number) and ζ(stated in the above paragraph). Moreover, set the content of T.S. memory unit to empty.

Step 2: Randomly generate λ solutions for the given clustering problem and use the one with the lowest SSE, say, solution S as the current solution. Note that we represent every solution in binary form and calculate the corresponding SSE by setting the solution as initial seeds of k-means method and then run ten iterations for the k-means method.

Step 3: Disturb current solution S to generate λ candidate solutions, say, S^1, S^2,...,S^λ. Here we generate every candidate solution by the following method: to decide every bit value of every candidate solution, we randomly generate a real number $\delta \in (0, 1)$. If $\delta \geq P_r$ we set the bit value identical to the corresponding bit value of S; otherwise, take complement bit value.

Step 4: Calculate the SSEs of all candidate solutions S^1, S^2,...,S^λ, and choose the one possesses the best SSE. Let S^b be the best one. Let ΔS be a number defined by $\Delta S = (S^b)$ XOR (S), where XOR denotes exclusive or. (ΔS is used to keep the bit positions of S^b on which the values are different from that of S.)

Step 5: Check whether ΔS is stored in the T.S. memory unit. If not, the move ΔS (which means currently the move from S to S^b) is said to be not forbidden, and we proceed to Step 7. Otherwise, go to Step 6.

Step 6: If the SSE of S^b is better than that of the best solution we have ever found so far, restores ΔS (of S^b) so that it is the newest stored value in T.S. memory unit and go to Step 8. Otherwise, choose the best solution from the remaining candidate solutions $\{S^k | 1 \leq k \leq \lambda, k \neq b\}$ whose move (from S to S^k) are not forbidden. For convinence, we still use S^b to denote this solution and compute ΔS by the formula listed in Step 4. (If the moves from S to all S^k in the remaining candidate solutions are forbidden, however, go to Step 3.).

Step 7: Store ΔS in T.S. memory unit. If the number of stored values exceeds ω, remove the oldest one before memorizing ΔS.

Step 8: Move S to S^b (i.e., set $S = S_b$) and remember the best solution met so far.

Step 9: Repeat Steps 3 $\sim$ 8 until a pre-determined iteration number TIN is reached. □

In the above, we use ΔS to represent the move from one possible solution to another one (see Step 4) and we use further a memory unit to keep track of the moves made recently (see Step 7). Whenever we want to make a move, we always check whether this move had been taken recently (see Step 5). We usually avoid repreating those moves made recently (see Steps 5 and 6). The reason is that this restriction usually reduce the chance of being trapped in a local minimum. However, this restriction can be violated in a special case – that is, it is allowed to take a forbidden move when this move can reach a solution better than the best one that we had ever met (see Step 6). This exception is reasonable and is also called the aspiration criterion in the T.S. method [3].

Applying the above steps, we continuously searching for a good neighbors of current solution and use a memory unit to direct our searching (on moving the current solution to the next one). Although this searching process could not guarantee a global minimum, we found it usually reach good solutions. (The performance of our method is listed in Section 4.)

4 Parameter setting

The parameters used in our system include λ, ω, ζ, TIN and P_r. The choice for the values of λ, ζ and TIN

is a trade-off between solution goodness and computation effeciency. The larger the values of λ, ζ and TIN are, the more solutions could be explored, and thus the more chance to obtain the optimal solution. However, the searching space will also grow accordingly. The size of ω determines the capability of memorizing the history of moves and a special case of $\omega = 0$ makes the tabu search become the neighborhood search method. On the other hand, a large value of P_r means the differences between the current solution and each generated candidate solution is large. If P_r is too large, it may make the tabu search behave like the random search. Awaring of the characteristics of these parameters, we use several data sets to test various(36) distinct parameter settings (with P_r=0.3, 0.2, 0.1, 0.05, λ=10, 15, 20 and ω=3, 10, 15) and we found that P_r=0.05, λ=15 or 20 and ω=10 or 15, usually yields good results. (Here we use ζ=4 and TIN=100.)

5 Experimental results and conclusion

We used three real data sets, namely, GFB [5], Iris [6], and 8ox [6], to examine the performance of our method. The GFB data set contains 89 postal zones in Bavaria (Germany), and their four attributes as self-employed people, civil servants, clerks and manual workers. In other words, this data set contains 89 4-dimensional data points. The Iris data set consists of 150 samples of four variables each. The variables are sepal length, sepal width, petal length, and petal width of various species of iris. Moreover, the 8ox data set contains 15 handwritten characters '8', 15 handwritten characters 'o', and 15 handwritten characters 'x'. Each character is represented by eight measured features. Namely, the 8ox data set contains 45 8-dimensional data points. The parameters we used were $P_r = 0.05$, $\lambda = 15$, $\omega = 10$, $\zeta = 4$ and TIN=100. Moreover, we also included the results of k-means and Simulated Annealing for comparisons. All the results were measured in terms of SSE and shown in Tables 1, 2 and 3. From these tables, it can be seen that our method is superior to the k-means method and the S.A. method excepts the 4-class case in Table 2 in which the SSE of the k-means is 57.2 whereas ours is 57.88 (the difference is small comparing to the other cases in which ours are better than that of the k-means), and S.A. gets 60.8 (in fact, the SSEs of our method are all better than that of the S.A. method in Tables 1 and 2). Table 3 shows that the results of our method, the k-means method and the S.A. method are all of the same when the data used is the 8ox.

In this paper, we successfully apply the tabu search to clustering problem. The method is found to perform well and is observed to be competitive, usually superior, to the k-means and S.A. methods.

Table 1: The SSE of our method, the k-means method and the S.A. method for the GFB data.

	number of classes		
	2	3	4
k-means	3.598 E10	3.051 E10	2.951 E10
S.A.	3.093 E10	0.825 E10	0.642 E10
our method	1.740 E10	0.756 E10	0.534 E10

Table 2: The SSE of our method, the k-means method and the S.A. method for the Iris data.

	number of classes		
	2	3	4
k-means	152.4	142.8	57.2
S.A.	152.4	93.6	60.8
our method	150.69	78.65	57.88

Table 3: The SSE of our method, the k-means method and the S.A. method for the 8ox data.

	number of classes		
	2	3	4
k-means	1507.5	1201.2	1026.2
S.A.	1507.5	1201.2	1026.2
our method	1507.5	1201.2	1026.2

References

[1] L. Kaufman and P.J. Rousseeuw, *Finding Groups in Data.* New York: John Wiley & Sons, 1990.

[2] A.K. Jain and R.C. Dubes, *Algorithms for clustering data.* New Jersey: Englewood Cliffs, 1988.

[3] F. Glover, J.P. Kelly and M. Laguna, "Genetic algorithms and tabu search: hybrids for optimization," Computer operation researches, Vol. **22**, No. 1, pp. 111-134, 1995.

[4] G.P. Babu and M.N. Murty, "A near-optimal initial seed value selection in k-means algorithm using a genetic algorithm," *Pattern Recognition Letters*, Vol. **14**, pp. 763-769, 1993.

[5] H. Spath, *Cluster analysis algorithms for data reduction and classification of objects*, Wiley, New York, pp. 103-104, 1980.

[6] R.A. Fisher, *The use of multiple measurements in taxonomic problems, Contribution to Mathematical Statistics.* Wiley, New York, 1950.

[7] R.W. Klein and R.C. Dubes, "Experiments in projection and clustering by simulated annealing," *Pattern Recognition,* Vol. 22, No. 2, pp. 213-220, 1989.

[8] F. Glover, Tabu search–part I, *ORSA Journal on Computing*, Vol. 1, No. 3, pp. 190-206, 1989.

[9] F. Glover, Tabu search–part II, *ORSA Journal on Computing*, Vol. 2, No. 1, pp. 4-32, 1990..

[10] T.E. Morton and D.W. Pentico, *Heuristic scheduling systems: with applications to production systems and project management,* John Wiley & Sons, New York, pp. 82-120, 1993.