Deterministic and Randomized Local Search Algorithms for Clustering

Pasi Fränti, Marko Tuononen, and Olli Virmajoki

Speech and Image Processing Unit
Department of Computer Science, University of Joensuu, Finland
{franti, mtuonon, ovirma}@cs.joensuu.fi

Abstract

We propose a local search algorithm for clustering based on deterministic variants of cluster swapping. Within a given time limit, the new method finds the correct clustering more efficiently than existing ones. The algorithm is simple to implement, which makes it useful for practitioners.

1. Introduction

One of the best clustering algorithm, *randomized local search* [1], is based on a simple cluster swapping technique. At every step, one cluster is tentatively re-allocated into another location and accepted if it improves the solution, see Fig. 1. This trial-and-error approach is extremely simple to implement and surprisingly effective. It has only one additional step (swapping) to the basic k-means algorithm [2]. The algorithm does not converge but it always finds the correct clustering eventually.

The main weakness of the algorithm is that it generates a large number of candidate solutions that do not provide any improvement. This is not a big problem itself but the time can become a critical factor when handling very large data sets, or used in real-time applications. On the other hand, the correct clustering in Fig. 1 could be found by a single swap, if we only knew which cluster to remove and where the new one should be added.

We consider deterministic variants of the swapping technique to produce the correct clustering more efficiently, but at the same time, avoid the problem of getting stuck into a local minimum, which can easily happen with heuristic swaps. Another challenge is to compute each swap fast.

2. Randomized local search

Given a set of N data vectors $X=\{x_1, x_2, \ldots, x_N\}$, clustering aims at solving partition $P=\{p_1, p_2, \ldots, p_N\}$ so that a given distortion function f is minimized. We consider here only the mean square error. Assuming that M is given, the clustering problem can then be defined as an optimization problem.

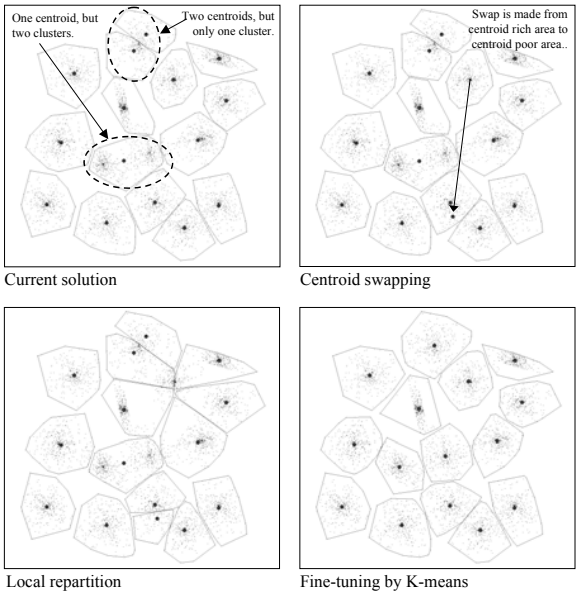

Figure 1. Demonstration of random swapping.

Randomized local search performs the clustering iteratively starting from any initial solution. At each step of the algorithm, a randomly chosen cluster is removed and another one is created elsewhere in the space. The new solution is locally fine-tuned by two iterations of k-means. The modification is accepted only if the new solution provides lower distortion than the previous solution.

The efficiency of the algorithm depends on the number of iterations needed, and the time required per iteration. We will next analyze these two factors separately.

2.1. Efficiency of the random swap

Assume that there is only one incorrect centroid allocation as in Fig. 1, and only one swap would be needed to correct the situation. The probability for selecting the incorrectly located group for removal is $1/M$, and the probability for selecting exactly the correct location for insertion is $1/M$.

As both of these must happen, the probability for a successful swap (p_{success}) is at least $(1/M)^2$.

On the other hand, the local fine-tuning performed by k-means is capable for relocating centroid gradually if the movement happens between neighbor clusters. It is therefore not necessary to find exactly the correct locations but to select the centroids for removal and insertion in the neighborhood where change is needed. For a more accurate analysis, we need to estimate the size of neighborhood.

We denote the number of neighbor clusters by α, which depends on the number of clusters (M) and the dimensionality (d). The probability for a successful swap can now be estimated as a function of α and M as follows:

$$p_{\text{success}} = (\alpha/M)^2 \tag{1}$$

Using the probability, we can estimate the number of iterations (T) needed to correct one misplaced cluster centroid. Suppose that we need one swap, and we want to find the correct clustering with probability p_{limit} (e.g. 95%). The expected number of iterations can be calculated as:

$$\begin{aligned} &(1-p_{success})^T \le 1-p_{limit} \\ &\Leftrightarrow T \le \frac{\log(1-p_{limit})}{\log(1-p_{success})} \end{aligned} \tag{2}$$

In Fig. 1, we can visually estimate that the clusters have about 4 neighbors, on average. This would give an estimate of $p_{\text{success}} = (4/15)^2 = 0.27^2 \approx 7\%$. According to (2), we would need 41 iterations to find the correct swap with 95% probability, and 95 iterations with 99.9% probability.

2.2. Time complexity of the method

The swap itself can be performed in O(1) time, but the local repartition requires more time. As there are N data vectors divided into M clusters, a randomly selected cluster contains N/M data vectors, on average. A new group for each of these vectors is found by searching their nearest centroids. Each search takes M distance calculations and comparisons, and as there are N/M searches, they sum up to $M \cdot N/M$ = O(N). Secondly, we must check for each data vector whether it remains in its current group or relocates to the newly created group. This takes O(N) time.

The time complexity of the k-means iterations is somewhat more complicated to analyze but the same principle applies: only local changes appears because of the swap. Normally, O(NM) time would be required per iteration but as we use the reduced search variant [3], the O(M) full search is needed only for the vectors in the changed cluster. For the rest of the vectors, it is enough to compute distance only to the changed centroids.

We estimate that the number of changed clusters equals to the number of neighbors of the removed and added clusters, estimated as 2α. The number of vectors in those clusters is estimated as $2\alpha \cdot (N/M)$, and the time complexity of the k-means iteration as $O(2\alpha \cdot (N/M) \cdot M) = O(\alpha N)$, which is also the overall complexity of the RLS iteration.

3. Deterministic swap

The swap consists of two independent steps: removal and insertion. We study next these two steps separately, and then consider their different combinations.

3.1. Removal of existing cluster

The main challenge for designing the deterministic selection is to avoid getting stuck into a local minimum. If the criterion fails even once, the algorithm would immediately stop making further progress, unless there is some level of randomness in the process.

We select the cluster that increases the distortion least [4] [5]. For calculating the *removal cost*, we find the second nearest centroid (q_i) for every data vector (x_i) as follows:

$$q_i = \arg\min_{\substack{1 \le j \le m \\ j \ne p_i}} \left\| x_i - c_j \right\|^2 \tag{3}$$

The removal cost for cluster (j) can now be estimated by summing up the differences if the vectors in the cluster are repartitioned to their second closest one (q_i). Taking into account that the centroids will be modified due to the change, it can be calculated as [5]:

$$D_j = \sum_{p_i = j} \left(\frac{\left| n_{q_i} \right|}{\left| n_{q_i} + 1 \right|} d(x_i, c_{q_i}) - d(x_i, c_j) \right) \tag{4}$$

where n_{q_i} refers to the size of the secondary cluster. The drawback is that the above operations require O(N) time, and as there as M clusters, the overall complexity is O(NM).

3.2. Creation of new cluster

We divide the selection task into two sub tasks, which are considered separately:

1. select an existing cluster,
2. select a location within this cluster.

It is expected that the choice of the cluster is more important, and the exact location within the cluster is less significant as the k-means will take care of local fine-tuning. Thus, the idea is first to select the cluster, and then add the centroid somewhere in its vicinity.

We choose the cluster having the highest distortion, which takes O(M+N) = O(N) time. The first term originates from the selection, and the second from repartition. Distortion of a cluster (E_j) can be calculated as:

$$E_j = \sum_{p_i = j} d(x_i, c_j) \quad (5)$$

Furthest vector in cluster is selected for the exact location.

3.3. Demonstration of the deterministic swap

One step of the deterministic swap is demonstrated in Fig. 2. The removal costs and the overall distortions for each cluster are listed in Table 1. For the removal, clusters 1 and 2 are clearly the best choices, and for addition, cluster 12 causes the highest distortion by a large margin. We apply here the *furthest vector* heuristic. As a result of the swap, all centroids will be allocated properly in the space, and their exact locations are then fine-tuned by k-means.

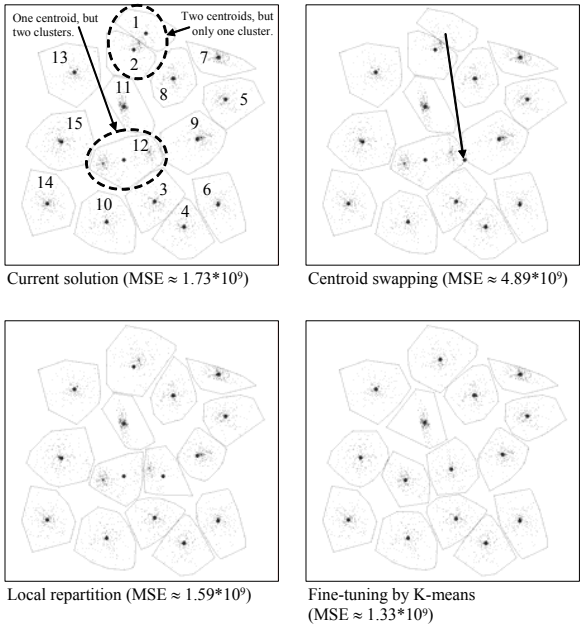

Figure 2. Demonstration of deterministic swap.

Table 1. Removal and additional costs.

	Removal cost (D_i)	Generated error (E_i)
1	0.80	0.39
2	1.04	0.64
3	5.48	1.09
4	5.66	0.92
5	6.50	0.76
6	7.67	1.01
7	8.47	0.45
8	9.10	0.75
9	9.90	1.42
10	11.09	1.26
11	11.47	0.61
12	12.17	4.70
13	14.61	0.94
14	16.41	0.93
15	16.68	1.41

3.4. Combining deterministic and random swaps

We consider the following four combinations of random and deterministic techniques:

- RR = random removal, random addition,
- RD = random removal, deterministic addition,
- DR = deterministic removal, random addition,
- DD = deterministic removal and addition.

Their time complexities are summarized in Table 2, and observed processing times in Fig. 3. The bottleneck of the deterministic swap is the removal, which dominates the processing time. Consider the analysis of section 2, the deterministic removal is more efficient than its random counter part if $M<\alpha T$, which is the case with data sets in Fig. 1 and 3 (M=15, $\alpha\approx$4, T=20).

The extra time required by the addition, on the other hand, is insignificant. It is therefore expected that the RD variant might be a good compromise between the random swap (RR) and deterministic swap (DD).

Table 2. Summary of the time complexities.

	Random removal		Deterministic removal	
	RR	RD	DR	DD
Removal	O(1)	O(1)	O(MN)	O(MN)
Addition	O(1)	O(N)	O(1)	O(N)
Local repartition	O(N)	O(N)	O(N)	O(N)
K-means	O(αN)	O(αN)	O(αN)	O(αN)
Total	O(αN)	O(αN)	O(MN)	O(MN)

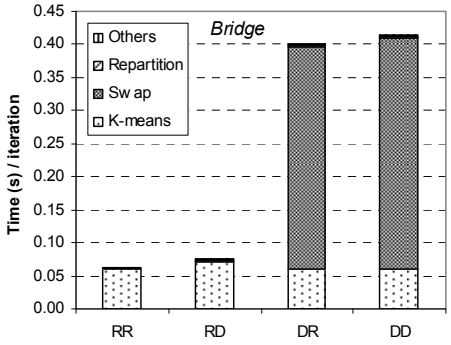

Figure 3. Profiles of the processing time.

4. Experiments

In the following, we cluster four image data sets [5] by an Intel® Xeon™ 2.80 GHz computer. Time-distortion comparison of the main variants is illustrated in Fig. 4. The best variant is RD, especially on sets that have recognizable clusters. When time is critical, it is clearly the best. In a long run, it is competitive with the other variants.

For the data sets that do not include clear clusters, the algorithms behave somewhat differently. For these sets, the deterministic swap (DD) completes the clustering fast but gets stuck into a local minimum that is slightly inferior to the variants with random removal (RR, RD).

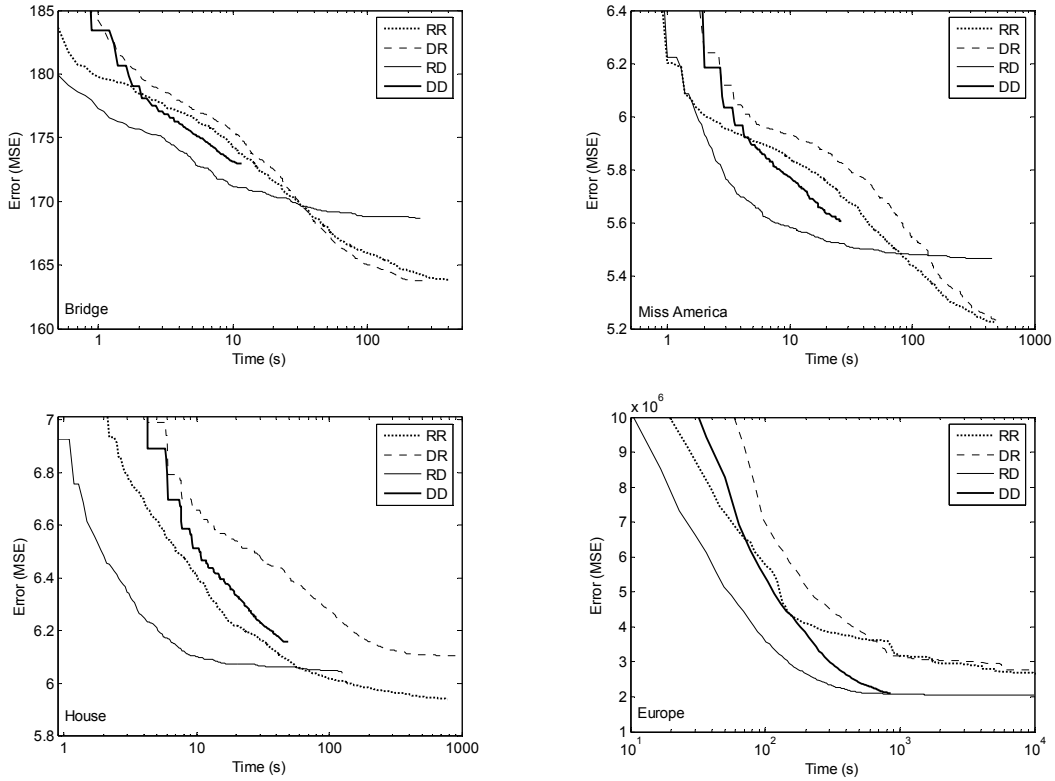

Figure 4. Time-distortion comparison of the main variants: RR, RD, DR and DD.

The results are briefly compared to some of the existing methods in Fig. 5; k-means, two hierarchical algorithms (PNN and Split), and the best known clustering algorithm (GAIS) [5]. The results compare favorable as RD outperforms most of the existing methods by a simple and time-efficient algorithm.

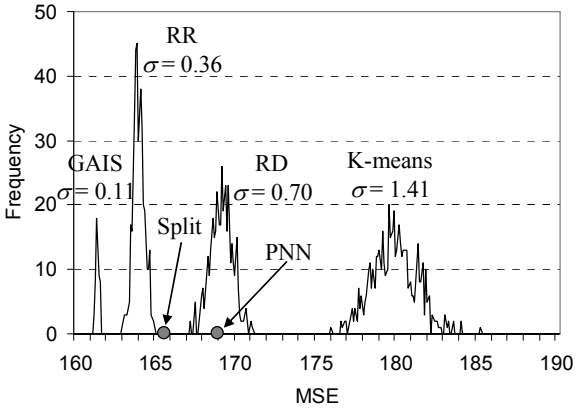

Figure 5. Histograms of the MSE-values of 50 runs of the *GAIS* method, 500 runs of the *k-means*, and 300 runs of the local search variants.

5. Conclusions

Deterministic and semi-deterministic variants of the swapping-based clustering were considered. The best combination was constructed from random cluster removal and a deterministic insertion.

6. References

[1] P. Fränti, J. Kivijärvi, "Randomized local search algorithm for the clustering problem", *Pattern Analysis and Applications,* 3 (4), pp. 358-369, 2000.

[2] J.B. McQueen, "Some methods for classification and analysis of multivariate observations", *5th Berkeley Symp. on Mathematical Statistics and Probability*, Berkeley, 1, pp. 281–297, 1967.

[3] T. Kaukoranta, P. Fränti, O. Nevalainen, "A fast exact GLA based on code vector activity detection", *IEEE Trans. on Image Processing,* 9 (8), pp. 1337-1342, 2000.

[4] B. Fritzke, "The LBG-U method for vector quantization – an improvement over LBG inspired from neural networks", *Neural Processing Letters,* 5 (1), pp. 35-45, 1997.

[5] P. Fränti and O. Virmajoki, "Iterative shrinking method for clustering problems", *Pattern Recognition*, 39 (5), 761-775, 2006.