Smart Swap for More Efficient Clustering

Jinhua Chen, Qinpei Zhao, and Pasi Fränti
School of Computing
University of Eastern Finland
Joensuu, Finland
{jinchen, zhao, franti}@cs.joensuu.fi

***Abstract*—Local search algorithms, such as randomized and deterministic swap-based clustering, are often used for solving clustering problem. In this paper, we propose a new swap-based local search algorithm, Smart Swap, which preserves the stability of the previous solutions but is more efficient. It performs the swap by finding the nearest pair among the centroids and sorting the clusters by their distortion values. Then it swaps one of the nearest pair centroids to any position in that cluster. K-means iteration is employed to repartition the dataset and to fine-tune the swapped solution. The algorithm is easy to implement and iterates less than the previous swap-based local search algorithms. Experiments show that the proposed algorithm keeps at least 97% stability for the synthetic datasets and 0.577 of standard deviation for the real data. It is also much faster than the other swap-based algorithms.**

I. INTRODUCTION

Clustering algorithms are widely used in machine learning, data mining, and pattern recognition. Besides the validity of the clustering result itself, computational efficiency is considered also as one of the most important criteria for identifying a good clustering algorithm.

The clustering problem can be defined as follows. Given a set of N D-dimensional data objects $X = \{x_1, x_2, \ldots, x_N\}$, partition the dataset into M clusters by optimizing a given cost function. Several swap-based local search algorithms for clustering have been proposed in literature [1-4]. In a so-called J-MEANS [1] algorithm, the jump/swap is relocating the centroids by considering all possible data objects which results in time complexity $O(N^2)$, and follows the changes of reassignments. Randomized local search (random swap) [2] is based on a simple swapping technique, which is performed by replacing a randomly selected centroid by a data object randomly selected as well. It applies first-improvement search strategy and accepts the new solution every time it improves the previous solution, as measured by the cost function.

Although the random swap is efficient search strategy on average, the total number of iterations required to find the clustering is upper bounded by quadratic $O(NM^2)$ dependency on the number of clusters [3]. A deterministic swap-based method has therefore been considered to find the good swaps faster. The method in [4] removes the centroid and replaces the centroid within the cluster that has the largest distortion. The method has the drawback that, unlike random swap, it can stuck into a local minimum, and that takes $O(MN)$ time to find the cluster to be removed.

The first problem (local minimum) was improved by considering four combinations of swap strategy in [5]. The combination of random removal and deterministic addition (RD) provided the best overall performance. The second problem (high time complexity) was attacked in [6] by proposing a faster implementation of the deterministic removal by maintaining secondary partition. Despite these improvements, the random swap remained the most favorite strategy due to its much simpler implementation and because the worst case slowness can be tolerated in most applications when M is not too high.

In this paper, we propose a simpler and more efficient alternative for the swap-based algorithm, called *Smart swap*. It can be considered as one of the deterministic swap-based algorithms. Instead of calculating the optimal choice for the centroid to be removed, it chooses the nearest pair of centroids as the target centroid to be removed. This can be calculated faster in $O(M^2)$ time. It then replaces the chosen centroid to any position in the cluster with the highest distortion. To speed up the algorithm, we employ a fast variant k-means to fine-tune the swapped result [7]. We compare the efficiency of the proposed method with random swap [2], hybrid swap (RD) [5] and fast hybrid swap (D^2R) [6] in the experiments.

II. SMART SWAP ALGORITHM

To express the algorithm more formally, we define the following notations:

- N Number of data objects;
- M Number of clusters;
- X Dataset with N data objects $X = \{x_i\}$, $i = 1,\ldots, N$;
- C Set of M cluster centroids $C = \{c_j\}$, $j = 1,\ldots, M$;
- P Set of N partitions $P = \{p_k\}$, $k = 1,\ldots, N$.

The clustering problem is an optimization problem. We use mean squared error (MSE) as the cost function for the optimization problem, calculated as:

978-1-4244-6878-2/10/$26.00 ©2010 IEEE

$$f(C)=\frac{1}{N}\sum_{i=1}^{N} d(x_i, c_{p_i})^2 \quad (1)$$

where c_{p_i} is the centroid of the cluster that x_i is assigned to, d is a distance function. In this paper, Euclidean distance is used, which is calculated as (for D-dimensional data object):

$$d(x_1, x_2)=\sqrt{\sum_{i=1}^{D}(x_1^i - x_2^i)^2} \quad (2)$$

A. Swap-based Clustering

The efficiency of a swap-based clustering algorithm depends on two questions: how many iterations (swaps) are needed, and how much time each iteration consumes. In the random swap algorithm [2], the swap step (randomly remove one centroid and randomly add at one position) is completely random so it needs a large number of iterations to provide a good quality result. In the deterministic swap method [4], the centroid to be removed is chosen by calculating removal cost, and the addition is made within the cluster of highest distortion. In this case, the number of iterations is limited because the algorithm will stop whenever there is no improvement. However the time required for each iteration is quite high, for example, it takes O(MN) for finding the minimum removal cost, or O(αN) if the secondary partition is maintained. Our algorithm attempts to find a local optimum with fewer swaps and shorter time required per iteration.

B. Smart Swap

The main challenge of efficient swap-based algorithm is to design an efficient swap pattern, i.e. which centroid to be removed and where to be replaced in as few iterations as possible.

Intuitively, combining two closest clusters is expected to work well, and more importantly, it can be calculated in a straightforward manner. Hence, we choose the centroid to be removed (c_{swap}) from the nearest pair of all centroids, which takes O(M^2) time by calculating the distance between any two centroids from C. For the replacement, one more effective choice is to choose the cluster with the largest distortion [4]. The distortion function is calculated as:

$$distortion(c_j)=\sum_{x_i \in c_j} d(x_i - c_j)^2 \quad (3)$$

High distortion value indicates a big variance inside the cluster, which implies that two clusters should appear instead of only one. Thus, the distortion values of the clusters are used to find the location for replacement and also to ensure that the algorithm will converge. In order to reduce the problem of getting stuck at a local minimum, we sort the clusters by their distortion values in descending order, marked as $S = \{s_{order}\}$ ($order$ = 1,..., M). For example, cluster s_1 has the largest distortion, and cluster s_M has the smallest distortion. At each swap, the cluster s_1 is selected as the first-priority replacement cluster ($c_{location}$). In most cases, this improves the result. However, when a local minimum is reached, no further improvement can be obtained using this greedy search strategy.

In the case of local minimum, cluster s_2 (the cluster with the second largest distortion) is selected instead of s_1. If improves, we continue by selecting the cluster s_1 again in the next iteration. Otherwise, we keep on selecting the next cluster (s_3) from the priority queue. Since the cluster with lower order in the priority queue are less likely to provide improvements, going through all of the clusters will increase the time complexity unnecessarily. Hence, we set a *Maxorder* (*Maxorder* = 1,..., M) to define the size of the search space in this algorithm. We can see that the larger the value of *Maxorder* is set, the slower the algorithm. We set *Maxorder* to $log_2 M$ as a compromise.

K-means algorithm is applied in our algorithm to repartition the swapped data objects and fine-tune the solutions. One iteration in the traditional k-means algorithm requires O(MN), which is quite high. To reduce the total time complexity of the algorithm, we employ a fast variant of k-means [7]. The fast variant classifies the clusters into active and static clusters. A cluster is labeled as active when the centroid of the cluster has been changed from previous iteration; otherwise it is labeled as static. Since most of the clusters belong to the static group during a swap, most calculation depends mainly on the number of active clusters. The time complexity of this fast variant is estimated as O(αN) in [5], where α is the number of neighbor clusters.

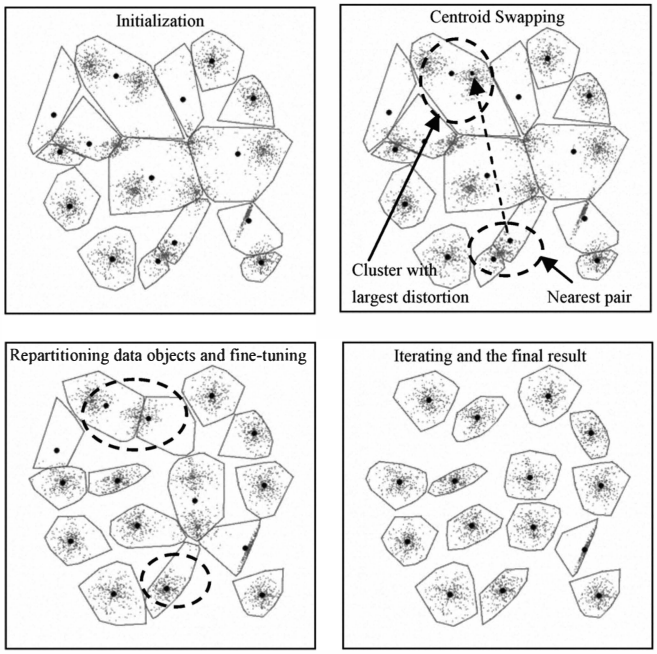

Figure 1. Demonstration of one run of Smart swap algorithm for a dataset with 5000 data objects and 15 clusters.

Summarizing the ideas above, we demonstrate one run of Smart swap algorithm visually in Figure 1, and then present the algorithm as follows.

Step 1 (Initialization): Initial centroids C are generated by taking M data objects chosen randomly from the dataset. Partition the data objects in X with corresponding C using one k-means iteration and get the partition of the clustering P. We set *order* = 1, and *Maxorder* = $log_2 M$.

Step 2 (Swap): Find the centroid to be removed and the location where the centroid should be relocated.

- Find the nearest cluster pair by calculating the distance between all of the centroid pairs in C.
- Calculate the distortions of each cluster and sort them to get a list S. Choose the cluster s_{order} as the replacing cluster $c_{location}$.
- Replace the chosen centroid with any point in the cluster $c_{location}$ since k-means can later fine-tune the solution. Here we select the first point in $c_{location}$ for simplification, and generate the C_{new}.

Step 3 (Repartitioning and fine-tuning): We use one k-means iteration for local repartitioning data objects since there is only one active centroid in the current C_{new} compared to the previous C, and then two k-means iterations for fine-tuning centroids.

Step 4 (Stopping Criterion):

- If $f(C_{new}) < f(C)$, it means that the result has been improved, we then reset the *order* = 1 and repeat the Step 2 and Step3.
- Otherwise, resume the previous C and do one more k-means iteration for refinement, then increase *order* to *order* + 1, and repeat the Step 2 and Step 3. Meanwhile, check the stopping criterion: *order* > *Maxorder*, which is to terminate the iterating (a local optimum was found).

The pseudo-code of the Smart swap is shown in Figure 2.

```
SmartSwap Local search Algorithm:

C ← InitializeCentroids(X);
P ←PartitionDataset(X, C);
Maxorder ← log2M;
order ← 1;
WHILE order < Maxorder
    ci, cj ←FindNearestPair(C);
    S ← SortClustersByDistortion(P, C);
    cswap ←RandomSelect(ci, cj);
    clocation ←sorder;
    Cnew ← Swap(cswap, clocation);
    Pnew ← LocalRepartition(P, Cnew);
    KmeansIteration(Pnew, Cnew);
    IF f(Cnew) < f(C), THEN
        order ← 1;
        C ←Cnew ;
    ELSE
        order ← order + 1;
        KmeansIteration(P, C);
```

Figure 2. Pseudo-code of Smart swap algorithm

It should be highlighted here that the Smart swap, as a local search algorithm, can converge very fast. We have also observed from a large number of experiments that the additional iterations, from increasing the search space by setting *Maxorder*, is less than 2 times of *Maxorder* ($2log_2M$ in this paper) compared to the general deterministic swap algorithms.

C. Time Complexity Analysis

Random initialization requires O(MN) time due to performing the optimal partition, which is the bottleneck of the algorithm in the beginning.

After that during each iteration, the algorithm needs O(M^2) to find the nearest pair, O(N) time to calculate the distortion of the clusters, O($MlogM$) to sort the clusters according to the distortion, and finally O(N) to evaluate the swapping result. These sum up to $M^2 + MlogM + N$ = O(N) for every iteration, with the assumption that the number of clusters is upper limited by $M < \sqrt{N}$.

The main bottleneck comes from repartitioning and fine-tuning by the k-means iterations. Even with the fast variant of k-means, it takes O(αN), on average. The fast variant has little effect at the early iterations because most of the centroids are still active. However, it reduces the processing time significantly when the algorithm approaches to the optimal solution. For higher number of clusters (M), the fast k-means algorithm works much more efficient.

To sum up, the proposed algorithm has O(MN) time complexity of the initialization, and then O(αN) for each iteration. This outperforms the previous algorithms presented in [1, 4, 5], and equals the hybrid swap in [6], and he random swap in [2]. Meanwhile, the algorithm requires much fewer iterations to reach the same clustering quality as that of the algorithm in [6], and significantly less iterations than required by the random swap [2].

III. EXPERIMENTS

The algorithm is tested on both synthetic and real datasets. To demonstrate the efficiency of the proposed algorithm, it is compared with three other swap-based local search algorithms: random swap [2], hybrid swap (RD) in [5] and fast hybrid swap (D^2R) in [6]. The datasets are described in Table 1 (http://cs.joensuu.fi/sipu/datasets).

TABLE 1. DATASETS USED IN EXPERIMENTS

Name	Number of objects	Description
S1	5000	Synthetic 2-d dataset with 15 clusters
A2	5250	Synthetic 2-d dataset with 35 clusters
A3	7500	Synthetic 2-d dataset with 50 clusters
House	34112	Image 3-d dataset (64 clusters)

The swap-based algorithms can improve the local optimal problem of local search algorithms [1-4]. The proposed algorithm shares the stability (low variation in the MSE across different runs) with random swap and hybrid swap. We run the Smart swap on all of the datasets 100 times. As shown in Figure 3, at least 97% of the runs for synthetic datasets (S1, A2, and A3) we obtain the same result (presumably the optimal clustering), which indicates the stability of the proposed algorithm on synthetic data. For real dataset (House), the standard deviation is 0.577. As shown in the figure, the result from each run keeps close to the

(assumed) optimal result. In our experiments, unstable result for real datasets is quite rare in practice.

As shown by the analysis in the previous sections, the proposed algorithm is equally efficient than the random swap and hybrid swap in one iteration. In the following experiments, we will demonstrate that the proposed smart swap requires fewer iterations.

The experiment is performed so that we compare the time consumed of each algorithm when they achieve the same MSE value (value that is observed to be the optimal or very close to optimal) with high stability (more than 97%). As shown in Figure 4, the Smart swap is the most efficient. It becomes even more efficient when increasing the number of clusters M in comparison to random swap. The main reason to affect the efficiency is the number of iterations. The Smart swap has dramatical improvements on the efficiency compared to the other two algorithms. It is at least 90% faster than the random swap, 75% faster than the hybrid swap, and 60% faster than the fast hybrid swap. Moreover, with the higher M value, the Smart swap becomes more efficient.

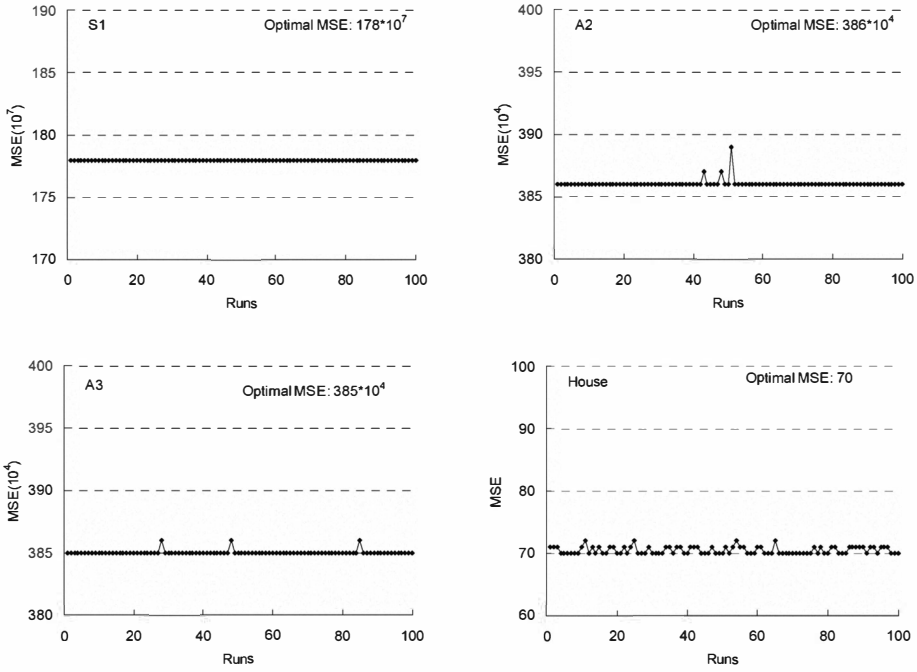

Figure 3. Stability of Smart swap algorithm. The experiments are performed 100 times on datasets S1, A2, A3 and House.

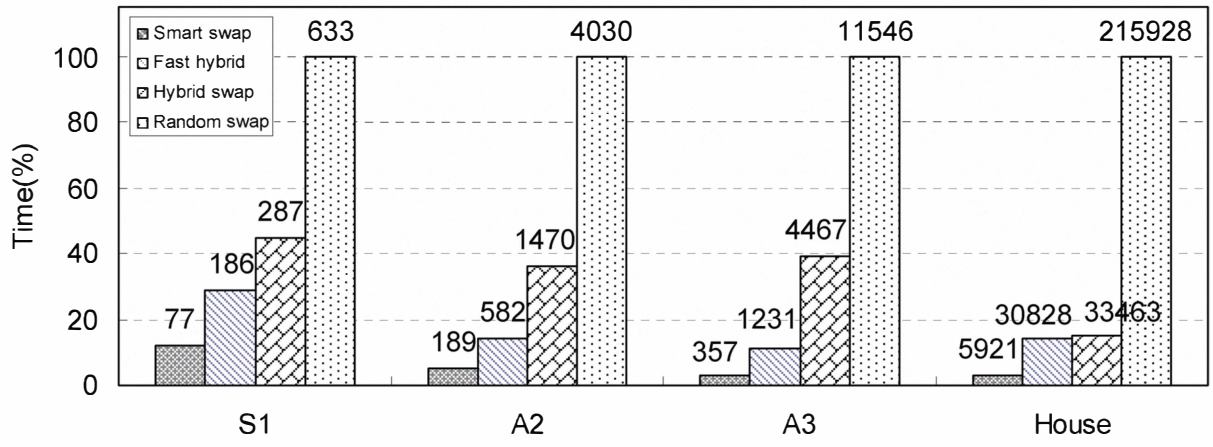

Figure 4. Efficiency comparison of Smart, Fast hybrid, Hybrid and Random swap-based local search algorithms on datasets S1, A2, A3 and House. The time is a percentage based on 100% for Random swap. The numbers on the bars are the exact processing time in millisecond to obtain the approximate optimal MSE with high stability.

IV. CONCLUSIONS

We have proposed a more efficient swap-based algorithm called Smart swap. It always selects the nearest cluster pair to be removed, which takes only $O(M^2)$ time, and chooses the cluster with largest distortion for relocating its position. In order to avoid getting stuck into a local minimum, we extend the search space by considering the next best cluster pairs, which causes only little time burden in the overall processing time. To reduce the total time complexity, a fast variant of k-means is applied for repartitioning and fine-tuning is employed. According to the experimental results, significant efficiency and competitive clustering result are obtained.

The proposed algorithm is easy to implement and gets competitive performance efficiently. However, as a local search algorithm, it still has the problem of getting stuck into a local optimal sometimes. How to improve the local optimal is a key point in the future studies.

V. REFERENCES

[1] P. Hansen and N. Mladenovic, "J-means: A new local search heuristic for minimum sum-of-squares clustering," Pattern Recognition, vol. 34, pp. 405-413, Jan. 2001.

[2] P. Fränti and J. Kivijärvi, "Randomised local search algorithm for the clustering problem," Pattern Analysis and Applications, vol. 3, pp. 358-369, 2000.

[3] P. Fränti, O. Virmajoki and V. Hautamäki, "Probabilistic clustering by random swap algorithm," IAPR Int. Conf. on Pattern Recognition (ICPR'08), Tampa, Florida, USA, Dec. 2008.

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

[5] P. Fränti, M. Tuononen and O. Virmajoki, "Deterministic and randomized local search algorithms for clustering," IEEE Int. Conf. on Multimedia and Expo, (ICME'08), Hannover, Germany, pp. 837-840, Jun. 2008.

[6] P. Fränti and O. Virmajoki, "On the efficiency of swap-based clustering," Int. Conf. on Adaptive and Natural Computing Algorithms (ICANNGA'09), Kuopio, Finland, pp. 303-312, Apr. 2009.

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