University of Eastern Finland

School of Computing

Master's Thesis

Swap-based Clustering for Location-based Services

Jinhua Chen

January, 2011

ABSTRACT

Clustering is an unsupervised learning method widely used in many fields, such as machine learning, pattern recognition, data mining and image analysis. The goal of this study is to investigate swap-based clustering and its application to location-based services. Swap-based clustering is a local search heuristic trying to find the optimal centroid locations by performing a sequence of centroid swaps between existing centroids and a set of candidate centroids.

Firstly, the thesis presents several swap-based clustering algorithms, such as random swap, deterministic swap and hybrid swap which is a combination of random and deterministic swap. Then we propose a simple and efficient swap-based clustering algorithm, called *smart swap.* It performs the swap by finding the nearest pair among the centroids and sorting the clusters by their distortion values, and then it swaps one of the nearest pair centroids to any position in the cluster from the clusters list sorted by distortion value. K-means iteration is employed to repartition the dataset and fine-tune the swapped solution.

Experimental results of swap-based clustering methods on both synthetic datasets and real datasets are provided and analyzed. Finally, we study location-based services and in one specific application, MOPSI project. We then apply the clustering in the MOPSI applications to reduce the clutter problem in map visualization in different scales, using a *split smart swap* clustering method to cluster the user locations and using a *grid-based* clustering with bounding box method to cluster the photo collections. Experimental results in the studied web applications show that the split smart swap method works in real-time but is slow for large dataset, and grid-based clustering method works with good clustering result and significant fast speed.

Keywords: swap-based clustering, smart swap, efficient clustering, location-based services, map visualization.

ACKNOWLEGEMENTS

I am grateful to University of Eastern Finland for giving the opportunity to participate in the IMPIT program and broaden my vision and experience.

I would like to sincerely thank my supervisor, Professor Pasi Fränti, for his significant guidance and helpful advices to my research and work. I would also like to thank every member of the Speech and Image Processing Unit, especially the members of MOPSI project team.

I would like to express my thanks to all of my friends for their support and encouragement, especially to the Chinese people in Joensuu.

At last, I would like to denote my deepest love and gratitude to my family for their moral support, and especially to my girlfriend Qinpei Zhao for her endless support and help to my studies and life. I truly appreciate all of you.

Table of contents

1 Introduction

1.1 Overview

Clustering is a method of unsupervised learning and a common technique widely used in many fields, such as machine learning, pattern recognition, data mining and image analysis. A loose definition of clustering could be "*the process of organizing objects into groups whose members are similar in some way*" [41]. A cluster is therefore a collection of objects that are similar to each other and dissimilar to the objects of other clusters [41]. We can show the clustering with a simple graphical example in Figure 1.

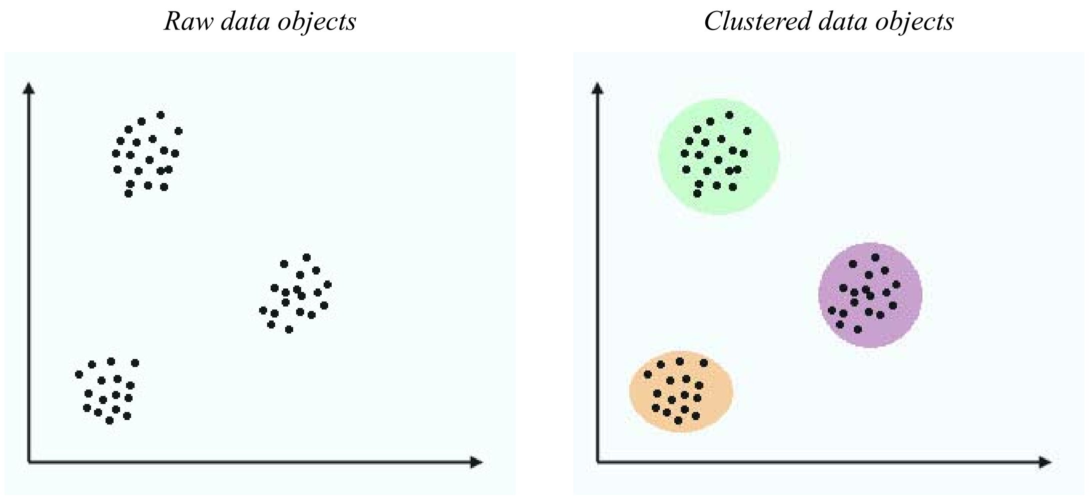


Figure 1. Example of clustering of data objects.

In this example, we can easily partition the data objects into three clusters shown as *yellow*, *green* and *purple* cluster. The similarity criterion here is distance: objects belong to the same cluster if they are "close" according to a given distance function, which is usually Euclidean distance.

The goal of clustering is to determine the intrinsic grouping in a set of unlabeled data objects [41]. There are many clustering algorithms including split-and-merge algorithms such as ISODATA [42], randomized sampling approaches such as CLARA [43], and methods based on density such as DBSCAN [11]. One of the most popular and widely studied clustering methods is called k-means clustering [3]. More information on clustering algorithms can be found in [2].

In this thesis, we study swap-based clustering including random swap [1], deterministic swap [17], and hybrid swap [23, 24]. Swap-based clustering is a local search heuristic trying to find the optimal centroid locations by performing a sequence of *centroid swaps* between existing centroids and a set of candidate centroids. We then present a more efficient clustering method called *smart swap*. Finally, we study location-based services and in one specific application, MOPSI project. We apply the clustering in the MOPSI applications to reduce the clutter problem in map visualization in different scales, using a *split smart swap* clustering method to cluster the user locations and using a grid-based clustering with bounding box method to cluster the photo collections.

1.2 Motivation

Besides the validity of the clustering result itself, computational efficiency is also considered as one of the most important criteria for selecting a good clustering algorithm. For swap-based clustering, the idea is simple and it can usually achieve good result just by random swap, whereas deterministic swap is aimed at being computationally more efficient. Therefore, researching on clustering quality and efficiency for selecting the best swap-based clustering variant is needed. On the other hand, location-based services are nowadays popular in everyday life, for example, searching the nearest service such as an ATM or hotel; turn-by-turn navigation to any address, or planning a route. Clustering technique can be used in these location-based applications to reduce the clutter of map visualization for the geographic information on the web. Thus, how to apply clustering to the practical applications is very necessary.

1.3 Purpose of the thesis

This study aims to investigate swap-based clustering algorithms including random swap, deterministic swap, and hybrid swap, and then proposes a more efficient clustering algorithm called smart swap. We compare the efficiency and clustering quality of these swap-based clustering algorithms on both synthetic datasets and real datasets. Findings are expected to shed light on the selection of proper clustering algorithm for the user.

In addition, we study location-based services and one of its applications in MOPSI project. We propose another two clustering algorithms combined with the practical web applications, split smart swap and grid-based clustering, to reduce the visual clutter of the map visualization for user locations and photo collections. We consider that clustering can be applied to real-time applications for a better representation of data.

1.4 Organization of the thesis

The rest of this thesis is structured as follows. In Section 2, we study the clustering methods and present two widely used algorithms: k-means clustering and hierarchical clustering, and then discuss one of the main problems in clustering: the number of clusters. In Section 3, we study the swap-based clustering algorithms including random swap, deterministic swap, hybrid swap and the proposed smart swap. The efficiency of these swap-based clustering methods is also analyzed. In Section 4, experimental results are reported on both synthetic datasets and real datasets to show the clustering quality and efficiency. In Section 5, we introduce location-based services and MOPSI application, and then apply the split smart swap and grid-based clustering algorithm to the MOPSI web applications to reduce the clutter problem in map visualization. Conclusions of this study are given in Section 6.

2 Clustering Methods

The clustering problem is defined to partition a set of data objects into subsets (called clusters) so that objects in the same cluster have similar features. The general clustering problem includes three sub-problems [1]: (a) selection of the evaluation function; (b) decision of the number of the clusters; (c) the choice of the clustering algorithms. Many clustering methods have been proposed in the literature [2]. These methods can be roughly classified into following five main categories: *partitional*, *hierarchical*, *density-based*, *grid-based*, and *model-based* methods. However, partitional and hierarchical algorithms are the most significant and widely used in clustering communities.

In general, partitional clustering methods attempt to decompose the dataset into various disjoint clusters. They are easy to use and work efficiently. *K-means algorithm* [3] is one of the most cited and typical partitional clustering algorithm. Hierarchical clustering methods seek to build a hierarchy of clusters, and can generally be classified into two types: *agglomerative methods* and *divisive methods*. Agglomerative method is a bottom-up approach: each observation is considered as one cluster and then two most similar clusters are merged into one cluster recursively. Divisive method, on the other hand, is a top-down approach: all observations start in one cluster, and select one cluster to be split into two clusters recursively.

In this section, we will first give the notations used for clustering and definition of the evaluation function, and then study the k-means clustering and hierarchical clustering algorithms. At last we discuss the decision of the number of clusters, which is one important sub problem in clustering.

2.1 Notations and definitions

To express the clustering problem more formally, we define the following notations used in the thesis:

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 $P = \{p_k\}, k = 1,\ldots, N$.

A cost function is used to evaluate the quality of the clustering methods. There is no universal function for all clustering, and the choice of the function usually depends on the application. We consider the clustering problem as an optimization problem, and mean squared error (MSE) is the most common selection as the cost function for the optimization problem, calculated as:

$$f(C) = \frac{1}{N}\sum_{i=1}^{N} d(x_i, c_{p_i})^2 \qquad 2.1.1$$

where c_{p_i} is the centroid of the cluster that x_i is assigned to, and *d* is a distance function. Euclidean distance and Manhattan distance are the most well-known methods for distance measurement. Euclidean distance of a *D*-dimensional data object is calculated as:

$$d(x_1, x_2) = \sqrt{\sum_{i=1}^{D}(x_1[i] - x_2[i])^2} \qquad 2.1.2$$

2.2 K-means algorithm

K-means [3] is the most famous clustering algorithm, which aims to partition N objects into *k* clusters so that each object belongs to the cluster with the minimum Euclidean distance to the cluster centroid. It starts with an initial solution, and then uses an iterative refinement technique, which is also referred to generalized Lloyd's algorithm [4], particularly in the computer science community. It contains 3 steps as follows.

1. Initialization step: Initial centroids *C* of the clusters are generated by taking *M* data objects chosen randomly from the dataset. The number of clusters *M* is given beforehand.

$$c_j = x_i \mid i = random(1, N), \quad 1 \le j \le M \qquad 2.2.1$$

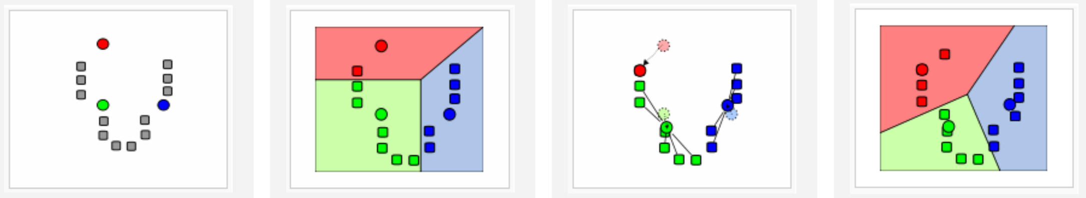
2. Assignment step: Assign each object to the cluster with the nearest centroid in respect to the distance function.

$$p_i \leftarrow \arg\min_{1 \le j \le M} d(x_i, c_j)^2 \, \forall i \in [1, N] \qquad 2.2.2$$

3. Update step: Centroid of each cluster is recalculated as the mean of the new partition.

$$c_j \leftarrow \frac{\sum_{p_i=j} x_i}{\sum_{p_i=j} 1} \forall j \in [1, M] \qquad 2.2.3$$

The assignment step and update step are performed iteratively until convergence. Often the number of iterations is set to a fixed number that depends on the data set and the desired clustering quality. A demonstration of the k-means algorithm is shown in Figure 2.



1) *k* initial "means" (*k*=3) are randomly selected from the data set (shown in color).	2) *k* clusters are created by assigning every object to the nearest mean.	3) Recalculate the centroid of each of the *k* clusters	4) Steps 2 and 3 are repeated until the centroids no longer move.

Figure 2. Demonstration of the k-means algorithm [5].

The k-means algorithm is very simple to use and reasonable effective in most cases. However, there is no guarantee that it will converge to a global optimum. The final results mainly rely on the initial steps, which results in the main drawback of k-means: getting stuck at a local minimum. It is common to run it multiple times with different initialization of centroids. This approach is called *repeated k-means*.

2.3 Hierarchical clustering algorithms

Hierarchical clustering method creates a hierarchy of clusters, which can be presented in a tree structure also known as a *dendrogram*. A dendrogram is a tree diagram frequently used to present the arrangement of the clusters. The root of the tree consists of a single cluster containing all objects, and the leaves correspond to individual objects. Basically hierarchical clustering algorithms are categorized into *agglomerative*, in which one starts at the leaves and successively merges clusters together, and *divisive*, in which one starts at the root and recursively splits the clusters. Figure 3 illustrates the process of hierarchical clustering.

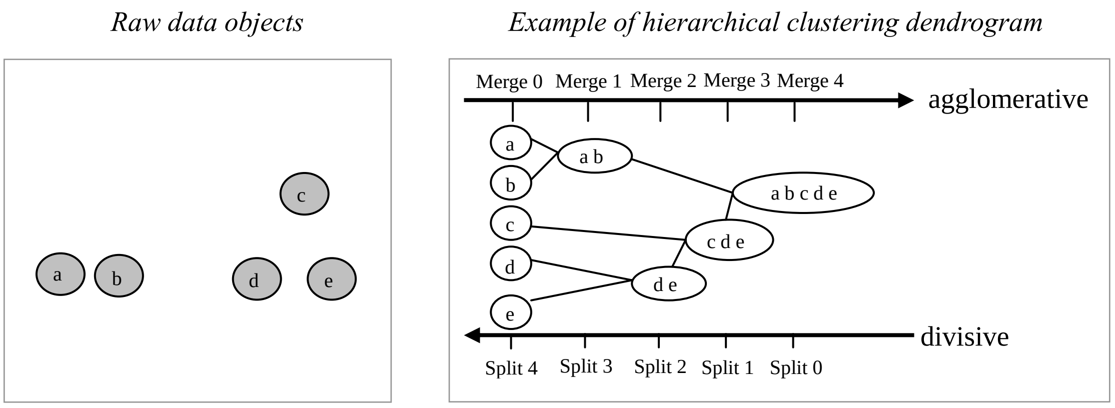


Figure 3. Dendrogram of hierarchical clustering.

The choice of which clusters to merge or split is determined by a linkage criterion, which is a function of the pairwise distances between objects. For agglomerative clustering, there are three main methods: *single-link*, *complete-link* and *average-link*, which differ in the similarity measures they employ. In single-link (or single linkage) clustering, two clusters whose two closest members have the smallest distance (two clusters with the smallest minimum pairwise distance) at each step. In complete-link (or complete linkage) clustering, two clusters whose two furthest members have the the smallest distance (two clusters with the smallest maximum pairwise distance) at each step. In average link (or average linkage) clustering, two clusters whose average of all members have the smallest distance (two clusters with the smallest average pairwise distance) at each step.

In addition, *Ward's method* [6] is distinct from the above methods because it uses variance approach to evaluate the distances between clusters. This method attempts to minimize the error sum-of-squares (ESS) of any two clusters that can be formed at each step.

Hierarchical clustering uses distance matrix as clustering criteria, and the termination condition can be specified by user, as the desired number of clusters. Hierarchical clustering outputs a hierarchy, a structure that is more informative, and it does not require user to pre-specify the number of clusters [7]. The major weaknesses are the time complexity is at least O(N^2) and they can never undo what was done previously. Furthermore, some algorithms integrate hierarchical with distance-based clustering such as BIRCH [8], CURE [9] and CHAMELEON [10].

2.4 Number of clusters

Determining the number of clusters in a data set is an important sub-problem in clustering, and it is a distinct issue from the process of actually solving the cluster problem since a priori knowledge is generally not available.

For a certain class of clustering algorithms such as k-means, the number of clusters *k* is specified already. Other algorithms, for example, DBSCAN [11] and OPTICS [12], do not require the specification of the number *k*.

The correct choice of *k* is often ambiguous and depends on the shape and scale of the distribution of the objects in a data set, and the desired clustering result user wants. If the number *k* is too small, different objects will not be separated. On the other hand, increasing *k* will reduce the amount of error in the resulting clustering but similar objects can be separated into different clusters. Both of these situations should be avoided. The *cluster validation* attempts to solve this problem and find the appropriate number of clusters. The process of cluster validation is to perform clustering over a wide range of values *k* and then find the optimal number that optimizes the given cluster validity criterion (Figure 4).

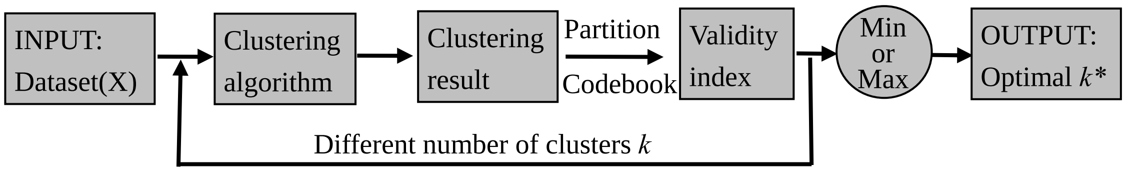


Figure 4. Flow charts view of cluster validation.

Basically there are three following approaches for measuring cluster validity.

- **External index:** Measure the similarity of clustering against known class labels (ground truth). For example, using entropy. However, ground truth is usually not available.
- **Internal index:** Measure the goodness of a clustering with intrinsic properties of data set without any external information, for example, using Sum of Squared Error (SSE), Silhouette coefficient [13].
- **Relative index:** Evaluate the quality of a clustering by comparing it to other clustering schemes, resulting by the same algorithm but with different parameter

values. Often an external or internal index is used for this function, e.g., SSE or entropy.

For more information about the clustering validity methods, you can find in [14], [15].

3 Swap-based Clustering

K-means gets stuck easily in locally optimal solutions that are far from globally optimal. In general, optimal clustering result can be achieved via finding optimal allocation of centorids. Swap-based clustering is a local search heuristic to find the optimal centroids, which works by performing a sequence of centroid swaps between existing centroids and a set of candidate centroids, and then by fine-tuning the exact location by a few iterations of k-means as demonstrated in Figure 5.

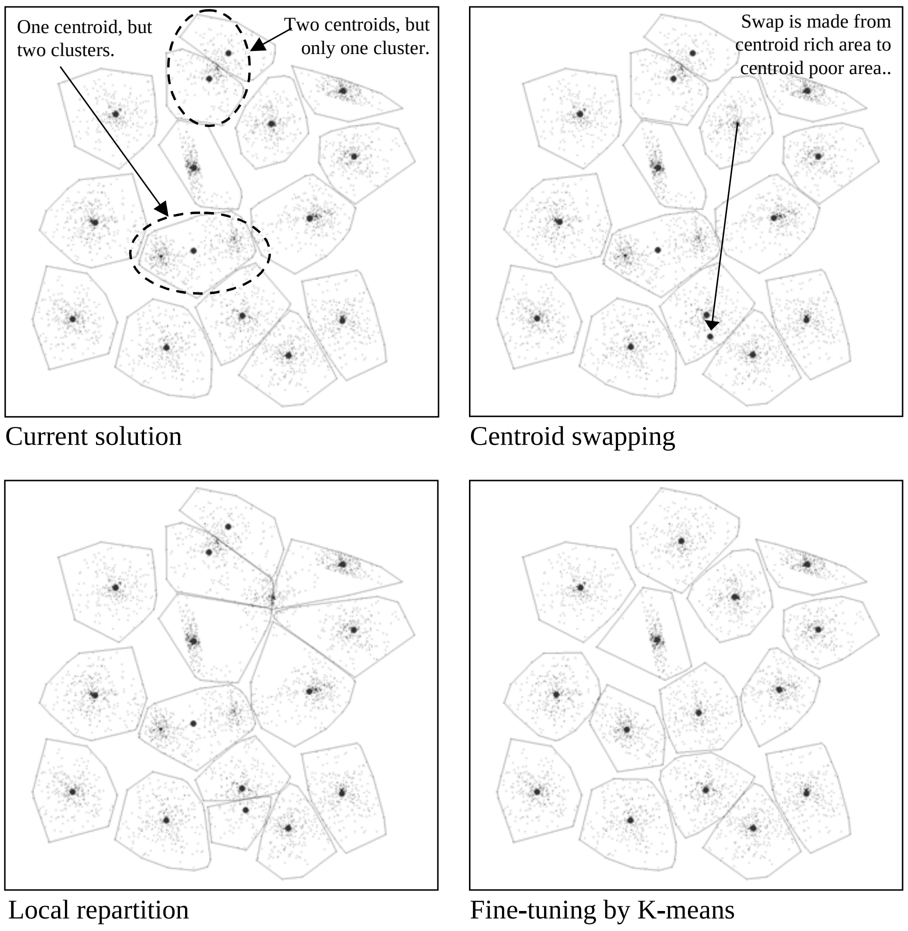

One centroid, but two clusters.
Two centroids, but only one cluster.
Swap is made from centroid rich area to centroid poor area..
Current solution
Centroid swapping
Local repartition
Fine-tuning by K-means

Figure 5. Demonstration of swap-based clustering for a data set with 5000 data points and 15 clusters [23].

Several swap-based clustering algorithms have been considered in literature [1, 16-25]. Randomized local search (random swap) [1] is based on a simple swapping technique, which is performed by replacing a randomly selected centroid by a randomly selected data object. It applies first-improvement search strategy and accepts the new solution every time it improves the previous solution, as measured by the cost function. The algorithm is easy to implement and it always finds the correct clustering eventually. In a so-called J-means algorithm [16], the swap relocates the centroids by considering all possible data objects. J-means and random swap achieve good clustering result but they do not always work efficiently since they will generate a large number of candidate solutions during swapping. Deterministic selection of the centroid to be swapped usually can speed up the algorithm. Therefore, some deterministic swap-based methods have been considered by selecting the centroid to be swapped as the one that increases the cost function value least [17, 18], or by merging two existing clusters [19, 20, 21] following the spirit of agglomerative clustering. The replacement location of the swapped centroid can be chosen either by considering locations of all possible data vectors [19], splitting an existing cluster [19], [22], or by using some heuristic such as selecting the cluster with the largest distortion [17].

The main drawback of the above deterministic swap methods is their computational complexity. Even though the correct clustering can be obtained by much fewer swaps compared to J-means and random swap, the time spent for selecting the best centroid is higher, which can make the overall efficiency lower. Another drawback is that the deterministic swap may get stuck in a local minimum if randomness is completely eliminated in the process [23].

The local minimum problem is improved by considering four combinations of swap strategy (hybrid swap) [24]: 1. random removal with random addition; 2. random removal with deterministic addition; 3. deterministic removal with random addition; 4. deterministic removal with deterministic addition. The combination of random removal and deterministic addition (RD) provided the best overall performance. The problem of high time complexity was attacked in [23] by proposing a faster implementation of the deterministic removal by maintaining secondary partition.

Despite these improvements, the time complexity is still high for real-time applications. Therefore, we propose a simpler and more efficient alternative for the swap-based algorithm, called smart swap [25]. It can be considered as one of the deterministic swap-based algorithms. Instead of calculating the optimal choice of the centroid to be removed, it finds the nearest pair of centroids as a target, and removes randomly one of the pair. This can be implemented in O(M^2) time. It then replaces the chosen centroid in the cluster with the highest distortion. For efficient implementation, we employ a fast variant k-means to fine-tune the swapped result [26].

3.1 Random swap

The random swap (randomized local search) [1] is based on a simple cluster swapping technique. It starts from any initial solution. At each step, one cluster centroid to be swapped is randomly selected, and then relocated into another randomly selected location. After that, local repartition is performed and the clustering is fine-tuned by two k-means iterations. The swap will be accepted if the clustering quality improves. This trial-and-error approach is very simple to implement and very effective in practice.

3.1.1 Principle of the algorithm

The algorithm contains the following steps.

Initialization step: Initial centroids *C* are generated by taking *M* data objects chosen randomly from the dataset.

$$c_j = x_i \mid i = random(1, N), \quad 1 \le j \le M \qquad 3.1.1$$

Partition step: Optimal partition is obtained by assigning each object in the dataset *X* to the cluster with the nearest centroid in respect to the distance function, and then gets the partition of the clustering *P*.

$$p_i \leftarrow \arg\min_{1 \le j \le M} d(x_i, c_j)^2 \quad \forall i \in [1, N] \qquad 3.1.2$$

Swap step: The centroid c_j to be swapped is chosen randomly and relocated into the location of data object x_i which is chosen randomly as well.

$$c_j \leftarrow x_i \mid j = random(1, M), \quad i = random(1, N) \qquad 3.1.3$$

Local repartition step: It contains two steps. The first step is to perform the optimal partition only for the data objects in the swapped cluster c_j, which will be removed from the clusters.

$$p_i \leftarrow \arg\min_{1 \le k \le M} d(x_i, c_k)^2 \quad \forall\, i \mid p_i = j \qquad 3.1.4$$

The second step is to create the partition for the new cluster in the region where the swapped centroid is relocated.

$$p_i \leftarrow \underset{k=j \vee k=p_i}{\arg\min}\, d(x_i, c_k)^2 \quad \forall i \in [1, N] \qquad 3.1.5$$

The motivation of the local repartition is that only one centroid is changed for each swap, which will affect the partition around the centroid. Rest of the partition remains unaffected. Furthermore, if the original partition is optimal (in respect to the previous *C*), the new partition is also optimal (in respect to the modified *C*). In this case, the local repartition corresponds to the optimal partition. However, the optimal partition takes O(*MN*) time but local repartition requires only O(*N*) time.

Fine-tuning step: For each swap iteration, one centroid is changed. However, even a single swap is a big change compared to the previous partition. The local refinement is therefore enhanced by applying k-means iterations. Usually two k-means iteration is enough providing best time-quality tradeoff [1].

Evaluation step: If the new partition provides lower distortion than the previous partition, the swap will be accepted. Usually, MSE is used as the cost function (2.1.1).

A pseudo-code of the random swap is shown in Figure 6.

```
Random swap algorithm ($X$) $\rightarrow$ $C, P$
    $C \leftarrow$ InitializeCentroids ($X$);
    $P \leftarrow$ OptimalPartition($X$, $C$);
    REPEAT $T$ times
      $C^{new} \leftarrow$ RandomSwap($C$);
      $P^{new} \leftarrow$ LocalRepartition($P$, $C^{new}$);
      KmeansIteration($P^{new}$, $C^{new}$);
      IF $f(P^{new}, C^{new}) < f(P, C)$ THEN
          $(P, C) \leftarrow P^{new}, C^{new}$;
```

Figure 6. Pseudo-code of random swap algorithm. The number of iterations *T* is fixed.

3.1.2 Number of iterations

To ensure a good clustering quality, the number of iterations T for random swap should be set to larger enough to find good (successful) swaps, which can reduce the cost function MSE and improve the clustering. At first sight, the probability for a good swap appears to be rather small and a large number of iterations would therefore be needed.

Given the number of clusters M, the probability for selecting the correct cluster centroid to be swapped is $1/M$, and the probability for selecting the right cluster to be relocated is also $1/M$ (the exact location in the cluster is not very important due to k-means fine-tuning). Thus, the probability for a good swap is at least $(1/M)^2$. However, the k-means fine-tuning is capable of relocating centorid gradually if the movement happens among neighbor clusters. It is therefore not necessary to find exactly the correct locations but to select the centroid to be swapped and the place to be relocated in the neighbor clusters.

For a more accurate analysis, we need to estimate the size of neighborhood. The number of neighbor clusters is denoted by α, which depends on the number of clusters M and the dimensionality. The probability for a good swap p can now be estimated as a function of α and M [27]:

$$p_{good} = (\alpha / M) \times (\alpha / M) = (\alpha / M)^2 \qquad 3.1.6$$

With the probability, we can estimate the number of iterations T needed to find a good swap. Suppose that we want to find a good swap with the probability p_{limit} (e.g. 95%). We use the q as the probability of failure to find a good swap, so q can be calculated as: q = 1-p_{limit}. The expected number of iterations T can be calculated as follows.

$$\begin{aligned} &(1 - p_{good})^T = q \\ &\Leftrightarrow T = \frac{\ln q}{\ln(1 - p_{good})} = \frac{\ln q}{\ln(1 - \frac{\alpha^2}{M^2})} \end{aligned} \qquad 3.1.7$$

The upper limit of T is:

$$T = \frac{\ln q}{\ln(1 - \frac{\alpha^2}{M^2})} \le \frac{-\ln q}{\frac{\alpha_2}{M^2}} = -\ln q \cdot \frac{M^2}{\alpha^2} \qquad 3.1.8$$

The lower limit is similar, so the *T* has tight bounds as:

$$T = \Theta\left(-\ln q \cdot \frac{M^2}{\alpha^2} \right) \qquad 3.1.9$$

Take the example of the dataset in Figure 5, we can estimate that the clusters have 4 neighbor clusters on average and the number of clusters *M* is 15. According to Equation (3.1.9), $T = 41$ iterations would be expected to find the *good swap* with 95% probability ($q = 0.05$), and $T = 95$ iterations with 99.9% probability. The dependency between *p* ($p = 1 - q$) in percentage and *T* is further demonstrated in Figure 7.

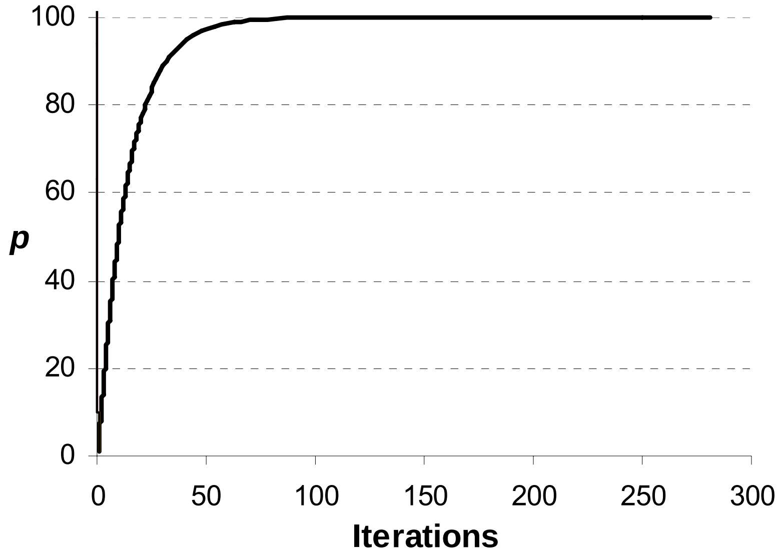


Figure 7. Probability of success *p* in percentage by iterations [27].

The number of iterations *T* we analyzed above is just for the case that only one good swap is needed for the clustering. To achieve good clustering quality, one good swap is not enough. However, it is very rare (< 2%) that three or more good swaps are needed observed from a large number of experiments.

3.2 Deterministic swap

In general, the clustering can be found only in a few swaps if the algorithm knows the centroid which should be swapped and the location where it should be relocated. Deterministic swap consists of two steps: selecting the centroid to be swapped and finding the location for the swapped centroid. The goal is to find the good swaps by systematic analysis rather than trial-and-error manner.

3.2.1 Selecting the centroid to be swapped

Several simple heuristic criteria can be considered for the selection: cluster with *smallest size* or *smallest variance* but these criteria do not work very well in practice. Some other approaches such as merging two existing clusters as in agglomerative clustering [18], and applying split-and-merge strategy as in [20, 21]. These approaches are possible to work but operating the clusters as entity can restrict the clustering too much and may result in the problem of getting stuck in a local minimum [23]. Considering the clustering problem as an optimization problem, it is sensible to select the cluster centroid to be swapped (removed) that increases the cost function value least [17, 18]. For calculating the removal cost, first it needs to find the second nearest centroid q_i for a given data object x_i in the partition p_i of the cluster to be removed, calculated as below:

$$q_i = \arg\min_{\substack{1\le j\le m \\ j\ne p_i}} \left\| x_i - c_j \right\|^2 \qquad 3.2.1$$

The removal cost for a cluster (*j*) can now be estimated by summing up the differences if the data objects in the cluster are repartitioned to their second nearest one q_i.

$$D_j = \sum_{p_i=j} \left(d(x_i, c_{q_i}) - d(x_i, c_j) \right) \quad \forall\, j \in [1, M] \qquad 3.2.2$$

where d denotes the distance function. Taking into account that the centoids will be updated after the repartition, it is calculated as [17]:

$$D_j = \sum_{p_i=j} \left(\frac{\left| n_{q_i} \right|}{\left| n_{q_i} + 1 \right|} d(x_i, c_{q_i}) - d(x_i, c_j) \right) \qquad 3.2.3$$

where n_{qi} refers to the size of the secondary cluster. The drawback of the deterministic removal is that it takes N distance calculations for each of the M clusters. Thus, the overall time complexity of the deterministic removal step becomes O(*MN*).

3.2.2 Finding the location for the swapped centroid

The replacement location of the swapped centroid can be chosen either by considering locations of all possible data points [19] but it would be very inefficient. To find the correct location, this task can be divided into two sub tasks:

1) Select an existing cluster.
2) Select the location within this cluster.

Selecting the correct cluster is more important, and the exact location within the cluster is less significant since k-means will be applied to take care of the local refinement of the centroid. Thus, the solution is first to select the correct cluster, and then add the centroid somewhere inside this cluster. One heuristic selection is to choose the cluster that has the largest distortion [17]. The distortion for the cluster (*j*) is calculated as follows:

$$E_j = \sum_{p_i = j} d(x_i, c_j) \qquad 3.2.4$$

After the correct cluster is found, the exact location with the cluster can be chosen considering the following heuristics [23]:

1) Current centroid of the cluster + ε [17].
2) Furthest data point.
3) Middle point of the current centroid and furthest data point.
4) Random.

Since the exact location is not very critical, any heuristic above can be selected.

3.2.3 Demonstration of the deterministic swap

The demonstration of the deterministic swap is shown in Figure 8. The removal costs and distortion values for each cluster are listed in Table 1. From the Table 1, cluster 1 is the best choice for removal (with minimal removal cost) and its centroid is chosen to be swapped. Cluster 12 has the largest distortion and is chosen to be the replacement for the swapped centroid. The *furthest data point* heuristic is applied in this example.

Table 1. Removal cost and distortion value for each cluster [23].

j	Removal cost (D_j)	Distortion (E_j)
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

j	Removal cost (D_j)	Distortion (E_j)
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

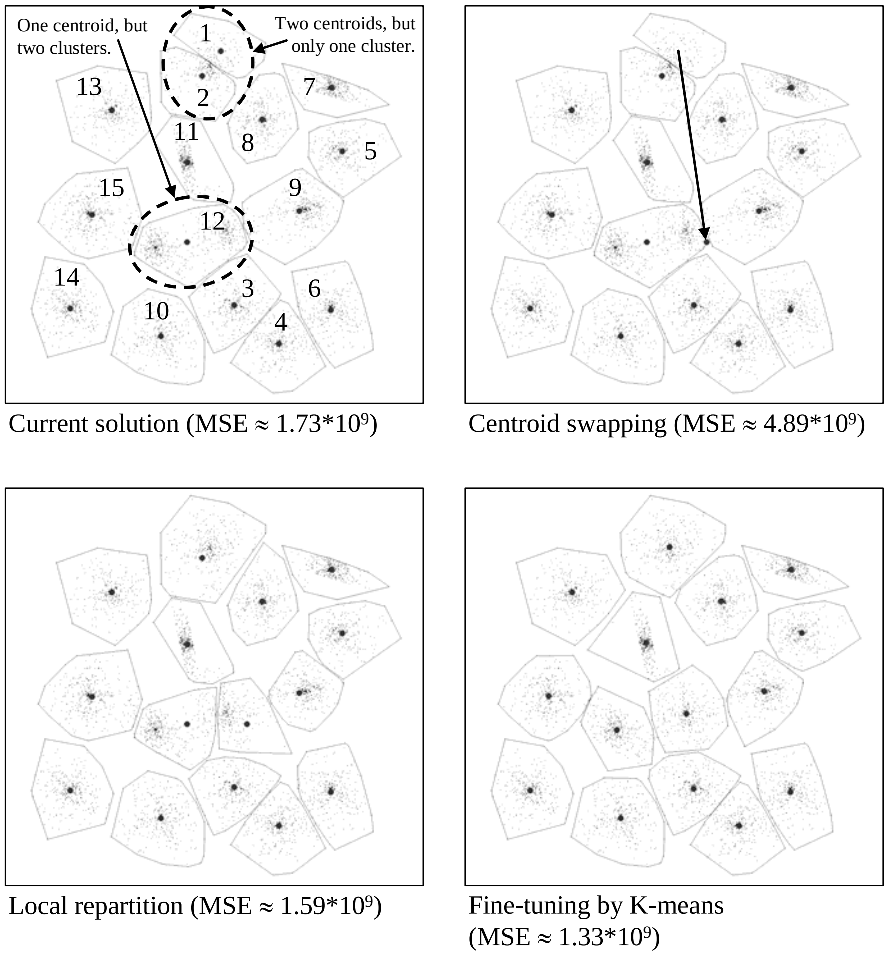


Figure 8. Demonstration of one deterministic swap for a data set with 5000 data points and 15 clusters [23].

3.3 Hybrid swap

The main drawback of the deterministic swap methods is their computational complexity. Even though deterministic swap can find the correct clustering by much fewer swaps in comparison to random swap, the time spent for selecting the best centroid to be removed is high that may make the overall efficiency lower. Another drawback is that the deterministic swap may get stuck in a local minimum, which can easily happen with heuristic swaps.

The local minimum problem is improved by considering four combinations of swap strategy (*hybrid swap*) in [24]. Another problem (high time complexity) was attacked in [23] by proposing a faster implementation of the deterministic removal by maintaining secondary partition.

The hybrid swap combines the deterministic heuristic with random swap. The following four combinations are considered.

1) RR = random removal + random addition.
2) RD = random removal + deterministic addition.
3) DR = deterministic removal + random addition.
4) DD = deterministic removal + deterministic addition.

Removal refers to the selection of centroid to be swapped, and the addition to the replacement location for the swapped centroid. We can see that RR corresponds to the random swap, and DD to the deterministic swap. Experimentally, RD was found to have the best performance [24] among these four combinations. However, the deterministic removal step takes O(MN) time and becomes the bottleneck of the hybrid swap. This problem is attacked in [23] by proposing a faster implementation of the deterministic removal by maintaining secondary partition, which only takes only O(αN) time, where α is the number of the neighbor clusters. For the k-means fine-tuning, the fast variant k-means [26] is employed for efficient implementation. The time complexities of these methods are summarized in Table 2.

Table 2. Summary of the time complexities of one iteration for hybrid swap methods.

	Random removal		Deterministic removal		Deterministic removal with updating data	
	RR	RD	DR	DD	D^2R	D^2D
Removal	O(1)	O(1)	O(MN)	O(MN)	O(αN)	O(αN)
Addition	O(1)	O(N)	O(1)	O(N)	O(1)	O(N)
Local repartition and k-means fine-tuning	O(αN)	O(αN)	O(αN)	O(αN)	O(αN)	O(αN)
Algorithm in total	O(αN)	O(αN)	O(MN)	O(MN)	O(αN)	O(αN)

From Equation (3.1.9), we can estimate the number of iterations needed for random swap. For the hybrid swap RD, the probability for selecting the correct centroid to be swapped is α/M and the probability for selecting the replacement is 1 assuming that the heuristic works (the cluster with largest distortion). In the same way, the probability for D^2R to select the correct centroid to be swapped is 1 (the cluster with minimum removal cost) and the probability for selecting the replacement is α/M. Thus, the probability of a good swap p can be estimated as:

$$p_{good} = (\alpha / M) \times 1 = \alpha / M \qquad 3.3.1$$

Furthermore, the number of iterations T can be calculated as:

$$T = \Theta\left(-\ln q \cdot \frac{M}{\alpha} \right) \qquad 3.3.2$$

3.4 Proposed method: smart swap

Despite the random swap is simple and the deterministic swap improves its efficiency, the overall time complexities of them are still high. Therefore, we propose a simpler and more efficient alternative called smart swap [25]. 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 (or swapped). This can be calculated in O(M^2) time in comparison to O(MN) of DR. It then replaces the chosen centroid to any position in the cluster with the highest distortion.

The main challenge of efficient swap-based algorithm is to design an efficient swap heuristic 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. This takes O(M^2) time by calculating the distance between all two centroids in C. For the replacement, we choose the cluster with the largest distortion [17]. The distortion function is calculated as in Equation (3.2.4).

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 $S = \{s_{order}\}$ ($order = 1,\ldots, M$). 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 it 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 would increase the time complexity unnecessarily. Hence, we set a *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* = *logM* as a compromise.

K-means algorithm is applied in our algorithm to repartition the swapped data objects and fine-tune the solutions. One iteration of 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 [26]. 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 after the swap, most calculation depends mainly on the number of active clusters. The time complexity of this fast variant is estimated as O(αN) in [24], where α is the number of neighbor clusters.

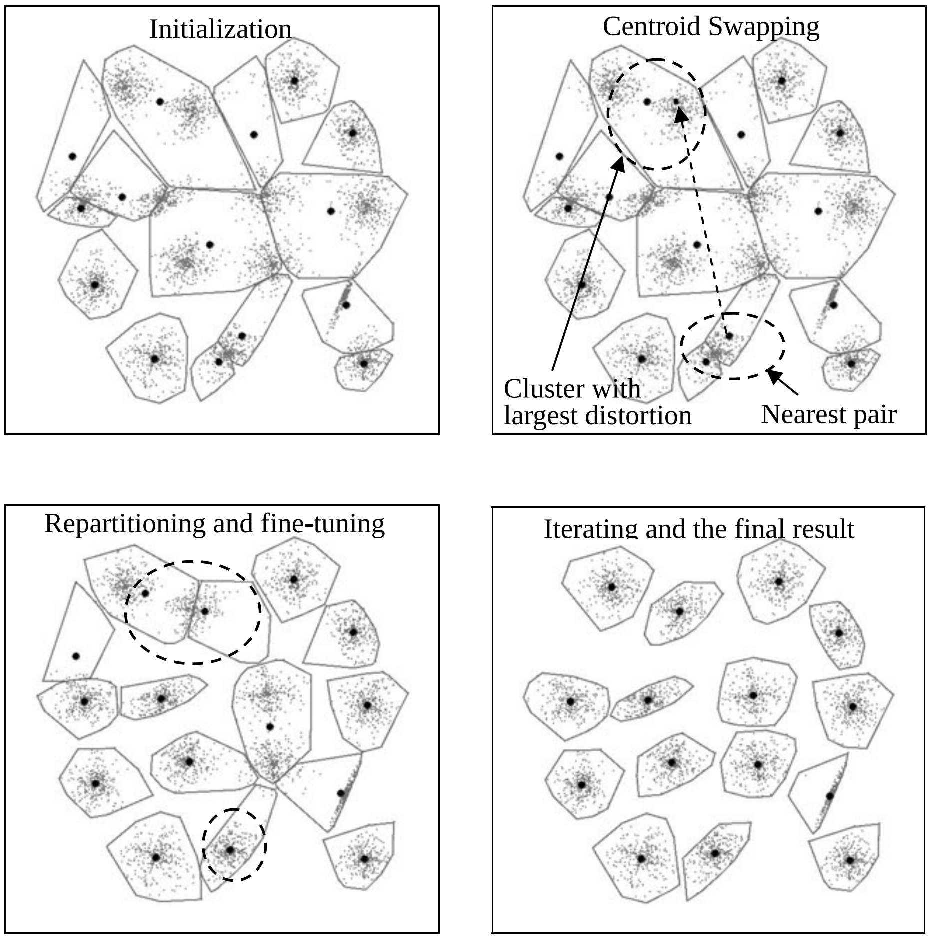

Initialization
Centroid Swapping
Cluster with largest distortion
Nearest pair
Repartitioning and fine-tuning
Iterating and the final result

Figure 9. 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 9, and then present the algorithm as following steps.

***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* = *logM*.

***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 centroids.
- Calculate the distortion of each cluster and sort them to 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 of the 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 the centroids.

***Step* 4 (Evaluation):**

- If $f(C_{new}) < f(C)$, replace C by the new solution and 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 10. 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* (2*logM* in this paper) compared to the general deterministic swap algorithms.

```
SmartSwap Local search Algorithm:
$C \leftarrow$ InitializeCentroids($X$);
$P \leftarrow$PartitionDataset($X$, $C$);
$Maxorder \leftarrow logM$;
$order \leftarrow 1$;
WHILE $order < Maxorder$
    $c_i$, $c_j \leftarrow$FindNearestPair($C$);
    $S \leftarrow$ SortClustersByDistortion($P$, $C$);
    $c_{swap} \leftarrow$RandomSelect($c_i$, $c_j$ );
    $c_{location} \leftarrow s_{order}$;
    $C_{new} \leftarrow$ Swap($c_{swap}$, $c_{location}$);
    $P_{new} \leftarrow$ LocalRepartition($P$, $C_{new}$);
    KmeansIteration($P_{new}$, $C_{new}$);
    IF f($C_{new}$) < f($C$), THEN
        $order \leftarrow 1$;
        $C \leftarrow C_{new}$ ;
    ELSE
        $order \leftarrow order + 1$;
        KmeansIteration($P$, $C$);
```

Figure 10. Pseudo-code of the smart swap algorithm.

3.5 Efficiency analysis

The efficiency of a swap-based clustering algorithm depends on two issues: how many iterations (swaps) are needed, and how much time each iteration consumes. We will next analyze the efficiency of the above swap-based clustering algorithms.

In the random swap algorithm, 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. It takes O(αN) [24] (α is the number of neighbor clusters on average, for example, 4 neighbor clusters in Figure 9) at least for each iteration with a fast variant of k-means [26] for fine-tuning. The main bottleneck of random swap is that the number of iterations T has the quadratic dependency on the number of clusters M from Equation (3.1.9), which increases the overall time complexity.

In the deterministic swap method, 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 high. For example, it takes O(MN) for finding the minimum removal cost, and O(N) for the addition cost, and O(αN) for the local partition and fine-tuning, so the total time complexity is O(MN) + O(αN) = O(MN).

In the hybrid swap, considering the RD (random removal with deterministic addition) with the best performance, the random removal takes O(1) time and the deterministic addition takes O(N) time, and O(αN) for the local partition and fine-tuning summing up to O(αN) in total for each iteration. However, the number of iterations needed is higher than that of the deterministic swap to ensure as good clustering quality as random swap. In the other hybrid swap D^{2}R [23], it takes O(αN) for the deterministic removal with the fast implementation, and O(1) for the random addition, and O(αN) for the local partition and fine-tuning. It takes O(αN) time in total for each iteration.

In the smart swap, the algorithm needs O(M^2) in each iteration 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 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, which is the bottleneck of all the other swap-based algorithms. 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, for the initialization step, it takes O(MN) time for all these swap-based clustering algorithms, and for each iteration step, it takes O(MN) time for the normal deterministic swap [17] and O(αN) time for the other algorithms: random swap, hybrid swap RD [24] and D^{2}R [23], and smart swap. However, the random swap needs a large number of iterations to provide good clustering quality. The deterministic swap has less number of iterations, but it takes much more time O(MN) for each iteration and may get stuck in a local minimum.

The hybrid swap methods RD and D^{2}R and can achieve better performance but still need many iterations. The smart swap takes O(αN) time for each iteration, which outperforms the deterministic swap, and equals the hybrid swap in [23, 24] and the random swap. Meanwhile, the algorithm requires much fewer iterations to reach the same clustering quality than the algorithm D^2R, and significantly less iterations than the random swap. The time complexities of these swap-based clustering methods are summarized in Table 3. The number of iterations is estimated from Equation (3.1.9) for random swap and from Equation (3.3.2) for hybrid swap. For smart swap, the number of iterations is $O(logM)$ and $O(\alpha N)$ for each iteration, so the total time complexity for iterations step is $O(\alpha NlogM)$. With the initialization step, the total time complexity for smart swap is $O(MN)$ assuming that $\alpha logM < M$.

Table 3. Summary of the time complexities for swap-based clustering methods.

		Random swap	**Deterministic swap**	**Hybrid swap**		**Smart swap**
				RD	D^2R	
Initialization		$O(MN)$	$O(MN)$	$O(MN)$	$O(MN)$	$O(MN)$
One iteration	Removal	$O(1)$	$O(MN)$	$O(1)$	$O(\alpha N)$	$O(M^2)$
	Addition	$O(1)$	$O(N)$	$O(N)$	$O(1)$	$O(N)$
	Local repartition and fine-tuning	$O(\alpha N)$	$O(\alpha N)$	$O(\alpha N)$	$O(\alpha N)$	$O(\alpha N)$
	Total	$O(\alpha N)$	$O(MN)$	$O(\alpha N)$	$O(\alpha N)$	$O(\alpha N)$
Number of iterations		$O(M^2/\alpha^2)$	$O(1)$	$O(M/\alpha)$	$O(M/\alpha)$	$O(logM)$
Total		$O(NM^2/\alpha)$	$O(MN)$	$O(MN)$	$O(MN)$	$O(MN)$

4 Experiments and Results

In this section, we present experimental results on the swap-based clustering algorithms and k-means as follows.

- Random swap
- Deterministic swap
- Hybrid swap RD
- Fast hybrid swap D^{2}R
- Smart swap
- K-means

All experiments are coded in Java and run on the Hewlett-Packard Presario Notebook PC with 1.6 GHz Intel Pentium CPU and 3062 RAM Window Vista.

The datasets include the synthetic datasets *S1*, *S2*, *S3*, *A1*, *A2* and *A3* (http://cs.joensuu.fi/sipu/datasets/), and large datasets *Birch1*, *Birch2*, *Birch3* [28], and real datasets *Iris* [29], *Bridge* and *House* (http://cs.joensuu.fi/sipu/datasets/). The datasets are described in Table 4.

Table 4. Datasets used in experiments

Name	**Number of objects**	**Description**
S1	5000	Synthetic 2-d (two dimensional) dataset with 15 clusters
S2	5000	Synthetic 2-d dataset with 15 clusters
S3	5000	Synthetic 2-d dataset with 15 clusters
A1	3000	Synthetic 2-d dataset with 20 clusters
A2	5250	Synthetic 2-d dataset with 35 clusters
A3	7500	Synthetic 2-d dataset with 50 clusters
Iris	150	Famous 4-d Iris data set of Fisher
Bridge	4096	Image 16-d dataset (64 clusters)
House	34112	Image 3-d dataset (64 clusters)
Birch1	100000	Synthetic 2-d dataset with 100 clusters in regular gird structure
Birch2	100000	Synthetic 2-d dataset with 100 clusters at a sin curve
Birch3	100000	Synthetic 2-d dataset with 100 clusters randomly

4.1 Synthetic datasets

The synthetic datasets *S1*, *S2*, *S3*, *A1*, *A2* and *A3* are two dimensional datasets plotted in Figure 11.

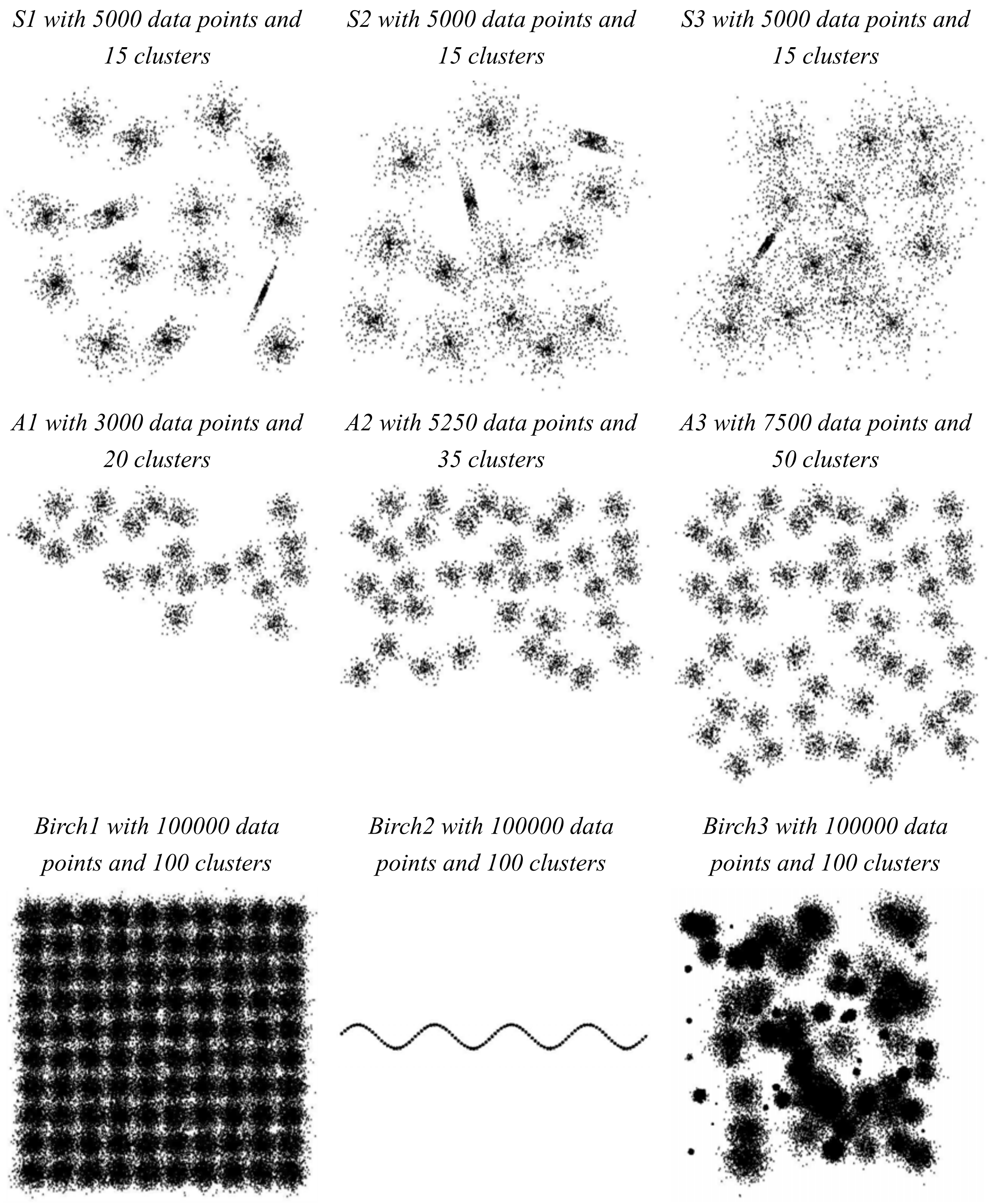


Figure 11. Synthetic datasets and their plots.

We first test the performance with mean square error (MSE) on these synthetic datasets *S1*, *S2*, *S3*, *A1*, *A2*, and *A3*, and large datasets *Birch1*, *Birch2* and *Birch3*. For the deterministic swap, the iterations will stop when no improvement found, and for smart swap, the iterations will terminate when the *Maxorder* is reached. However, for random swap, hybrid swap RD and fast hybrid swap D^2R, the stopping criterions are not specified and the number of iterations must be set by user. The bigger the number, the better clustering is obtained. Usually it is set to a number that is of the same order than the size of the dataset. In our experiments, the number of iterations T is calculated from Equation (3.1.9) for random swap and from Equation (3.3.2) for hybrid swap, assuming that the probability of failure q is 0.01 and two good swaps are needed. Table 5 shows the size of neighborhood on average estimated visually from Figure 11, and the iterations needed.

Table 5. Iterations needed for synthetic datasets.

Dataset	Number of clusters	Number of neighbors	Iterations needed	
			Random swap	Hybrid swap
S1	15	4	128	34
S2	15	4	128	34
S3	15	4	128	34
A1	20	4	230	46
A2	35	5	450	64
A3	50	6	638	76
Birch1	100	8	1438	114
Birch2	100	3	10232	306
Birch3	100	5	3648	184

The performances of these swap-based clustering methods on the synthetic datasets are shown in Table (6-14). Optimal solutions (MSE) are estimated by the random swap algorithm with a large number of iterations (size of the data points in the dataset).

Table 6. Performance for *S1* dataset (estimated optimal MSE: 1.7835×10^9).

Algorithms	MSE (10^9)	Time consumed (ms)	Iterations run
Random swap	1.7835	1182	128
Deterministic swap	1.7835	126	6
Hybrid swap RD	1.7835	392	34
Fast hybrid swap D^2R	1.7835	444	34
Smart swap	1.7835	75	6
K-means	3.2470	162	25

Table 7. Performance for *S2* dataset (estimated optimal MSE: 2.6558×10^9).

Algorithms	MSE (10^9)	Time consumed (ms)	Iterations run
Random swap	2.6558	1321	128
Deterministic swap	2.6567	69	4
Hybrid swap RD	2.6559	388	34
Fast hybrid swap D^2R	2.6558	470	34
Smart swap	2.6559	92	7
K-means	5.5286	159	25

Table 8. Performance for *S3* dataset (estimated optimal MSE: 3.3780×10^9).

Algorithms	MSE (10^9)	Time consumed (ms)	Iterations run
Random swap	3.3780	1492	128
Deterministic swap	3.4610	108	6
Hybrid swap RD	3.3781	431	34
Fast hybrid swap D^2R	3.3781	519	34
Smart swap	3.3783	113	8
K-means	3.8531	137	22

Table 9. Performance for *A1* dataset (estimated optimal MSE: 4.0488×10^6).

Algorithms	MSE (10^6)	Time consumed (ms)	Iterations run
Random swap	4.0488	1397	230
Deterministic swap	4.0771	60	5
Hybrid swap RD	4.0488	298	46
Fast hybrid swap D^2R	4.0488	332	46
Smart swap	4.0488	70	11
K-means	5.5108	82	15

Table 10. Performance for *A2* dataset (estimated optimal MSE: 3.8641×10^6).

Algorithms	MSE (10^6)	Time consumed (ms)	Iterations run
Random swap	3.8642	5004	450
Deterministic swap	3.8643	326	11
Hybrid swap RD	3.8641	814	64
Fast hybrid swap D^2R	3.8642	890	64
Smart swap	3.8642	157	11
K-means	7.4004	311	24

Table 11. Performance for *A3* dataset (estimated optimal MSE: 3.8583×10^6).

Algorithms	**MSE (10^6)**	**Time consumed (ms)**	**Iterations run**
Random swap	3.8585	9969	638
Deterministic swap	3.8638	734	13
Hybrid swap RD	3.8585	1570	76
Fast hybrid swap D^2R	3.8584	1494	76
Smart swap	3.8585	278	15
K-means	8.7950	760	27

Table 12. Performance for *Birch1* dataset (estimated optimal MSE: 9.2773×10^8).

Algorithms	**MSE (10^8)**	**Time consumed (ms)**	**Iterations run**
Random swap	9.2773	546468	1438
Deterministic swap	9.2789	16474	10
Hybrid swap RD	9.2773	76222	114
Fast hybrid swap D^2R	9.2773	81541	114
Smart swap	9.2773	14820	18
K-means	11.227	62930	100

Table 13. Performance for *Birch2* dataset (estimated optimal MSE: 4.5672×10^6).

Algorithms	**MSE (10^6)**	**Time consumed (ms)**	**Iterations run**
Random swap	4.5672	1956723	10232
Deterministic swap	4.5672	23837	22
Hybrid swap RD	4.5672	83320	306
Fast hybrid swap D^2R	4.5672	87111	306
Smart swap	4.5672	9969	29
K-means	14.663	26520	43

Table 14. Performance for *Birch3* dataset (estimated optimal MSE: 3.7362×10^8).

Algorithms	**MSE (10^8)**	**Time consumed (ms)**	**Iterations run**
Random swap	3.7362	1254958	3648
Deterministic swap	3.8497	28314	18
Hybrid swap RD	3.8197	131274	184
Fast hybrid swap D^2R	3.8128	139869	184
Smart swap	3.8731	31886	26
K-means	4.0712	59389	96

From Tables 6-14, we can see that the five swap-based clustering algorithms have almost the optimal clustering quality. However, k-means is very far from the optimal solutions. The clustering results of random swap, hybrid swap RD and fast hybrid swap D^2R are very stable and almost optimal, which indicates that analysis of estimating the number of iterations is reasonable. Random swap is the most inefficient due to the large number of iterations to guarantee good clustering quality.

Hybrid swap RD and fast hybrid swap D^2R have the same number of iterations. However, RD is a litter bit more efficient than D^2R because the removal and addition cost are $O(1) + O(N)$ for RD, but $O(\alpha N) + O(1)$ for D^2R for each iteration as shown in Table 3. The deterministic swap is very efficient in most datasets and the most efficient in case of *S2*, *S3*, *A1* and *Birch3*. On the other hand, it has the main weakness that it can get stuck in a local minimum. For example, for dataset *S3* in Table 8, the MSE value is 3.4610 but the optimal is 3.3781; and for dataset *A1* in Table 9, the MSE value is 4.0771 but the optimal is 4.0488. The proposed smart swap achieves the best performance in most cases with almost optimal clustering, and is the most efficient.

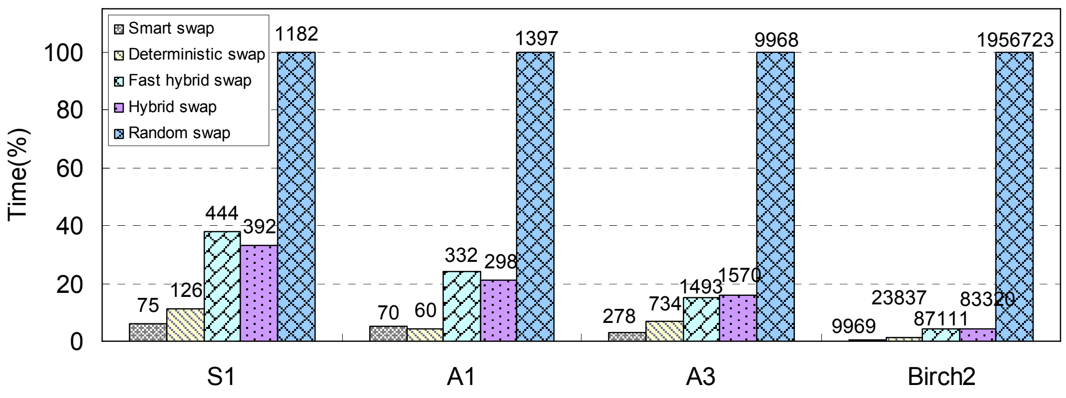


Figure 12. Efficiency comparison of Smart swap, Deterministic swap, Fast hybrid swap D^2R, Hybrid swap RD and Random swap on datasets S1, A1, A3 and Brich2 relative to Random swap. The numbers on the bars are the exact processing time in millisecond to obtain the approximate optimal MSE with high stability.

Observed from Figure 12, the smart swap is almost the fastest with dramatical improvements on the efficiency compared to the other three algorithms. It is at least 90% faster than the random swap, 75% faster than the hybrid swap RD, which is as fast as D^2R. Deterministic swap is the second fastest in most cases and especially the fastest for dataset *A1*. Moreover, with a higher value of *M*, smart swap becomes even faster. Especially for the large dataset *Birch2*, random swap takes 1957 seconds but smart swap only 10 seconds, which is 0.5% of the time random swap consumed. The hybrid swap RD and D^{2}R are also much more efficient compared to random swap.

We next compare the time-distortion efficiency of these clustering methods on all the synthetic datasets based on the above experimental data. The purpose of the comparison is to check the clustering quality that these algorithms achieve with increasing the processing time. The time-distortion efficiency is very useful in practice for users to select the favorable method in case of time limitation. The results are shown in Figures 13-21.

The proposed smart swap outperforms other swap-based clustering methods in most cases. For example, for datasets *S1*, *A1*, *A2*, *A3*, *Birch1* and *Birch2*, it achieves the best efficiency especially for *A2*, *A3*, *Birch1* and *Birch2*. From Figure 11 with the plotted image, we can find that the distribution of clusters in the datasets *S1*, *A1*, *A2*, *A3*, *Birch1* and *Birch2* are more obvious than the other datasets *S2*, *S3* and *Birch3*, It indicates that the smart swap works more efficiently for the dataset with clear cluster structure.

The deterministic swap is the second most efficient method. However the problem of getting stuck in a local minimum is its main bottleneck. From the comparison of hybrid swap RD and fast hybrid swap D^{2}R, they achieve similar efficiency. RD works a bit more efficient than D^{2}R on the datasets *S1, S2*, *A1*, *A2*, *Birch2* and *Birch3* as shown in Figures 13, 14, 16, 17, 20 and 21 respectively, but D^{2}R is better for *S3*, *A3* and *Birch1*.

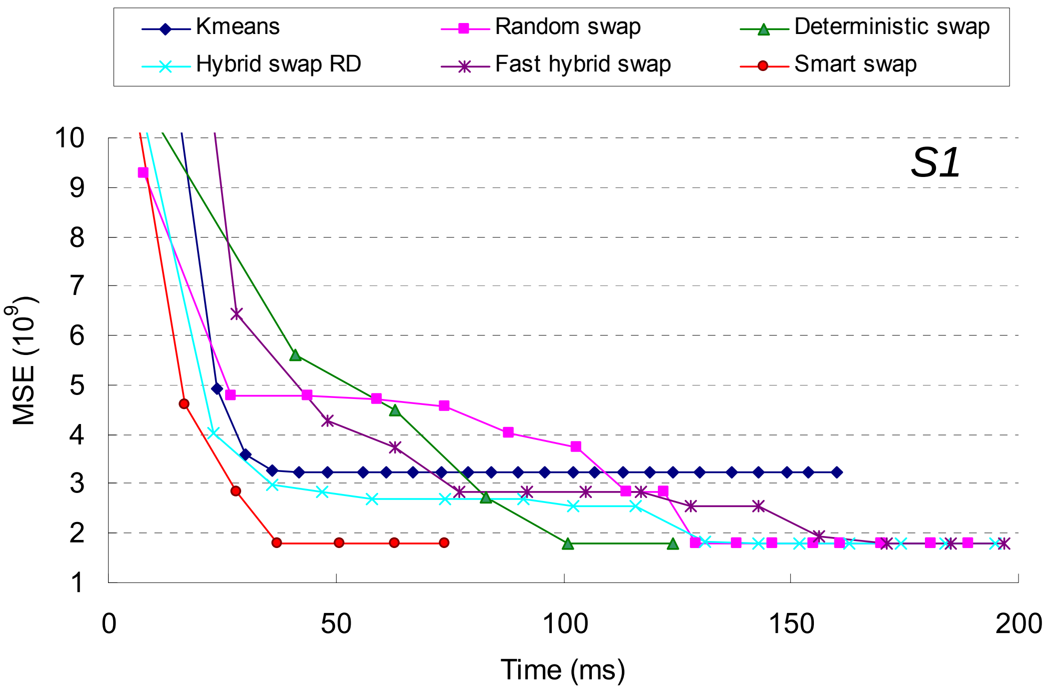


Figure 13. Time-distortion efficiency of clustering methods on dataset *S1*.

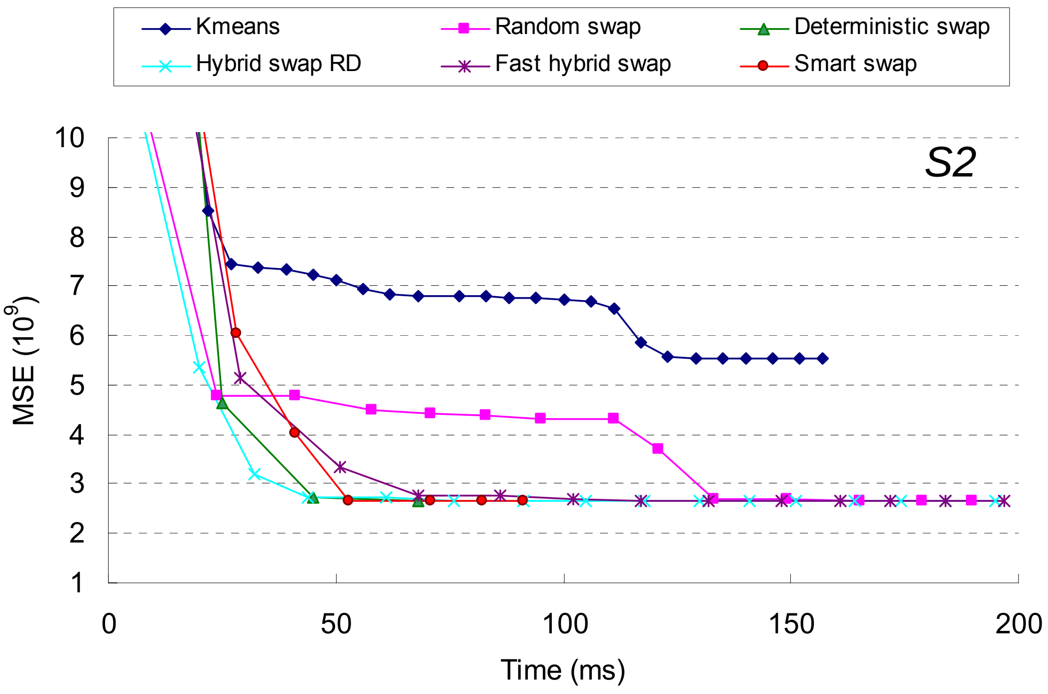


Figure 14. Time-distortion efficiency of clustering methods on dataset *S2*.

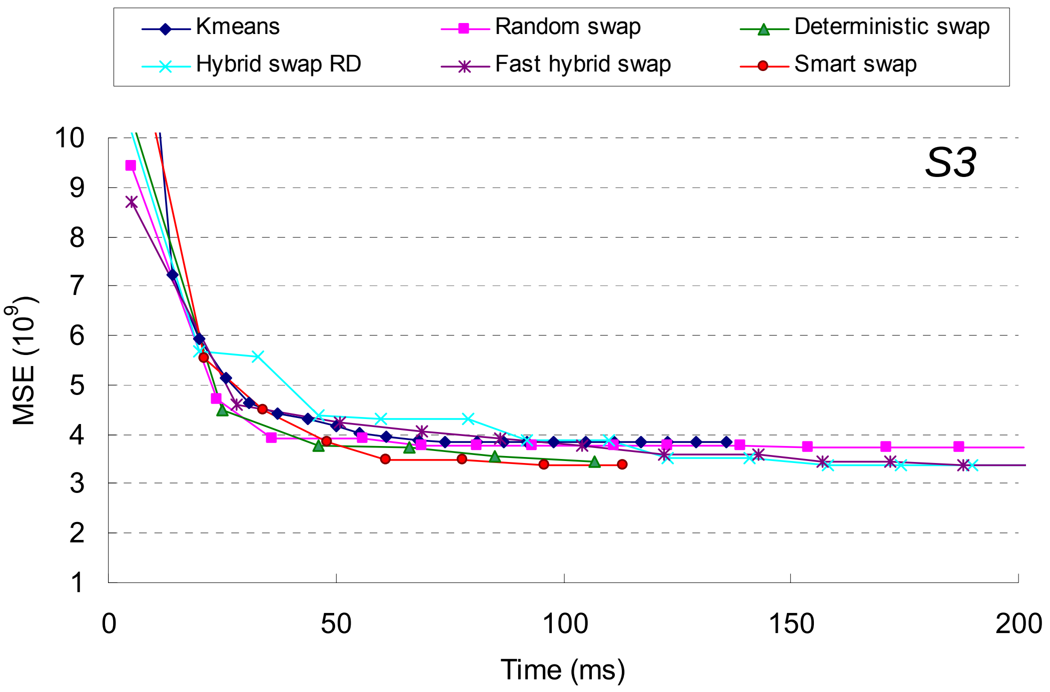


Figure 15. Time-distortion efficiency of clustering methods on dataset *S3*.

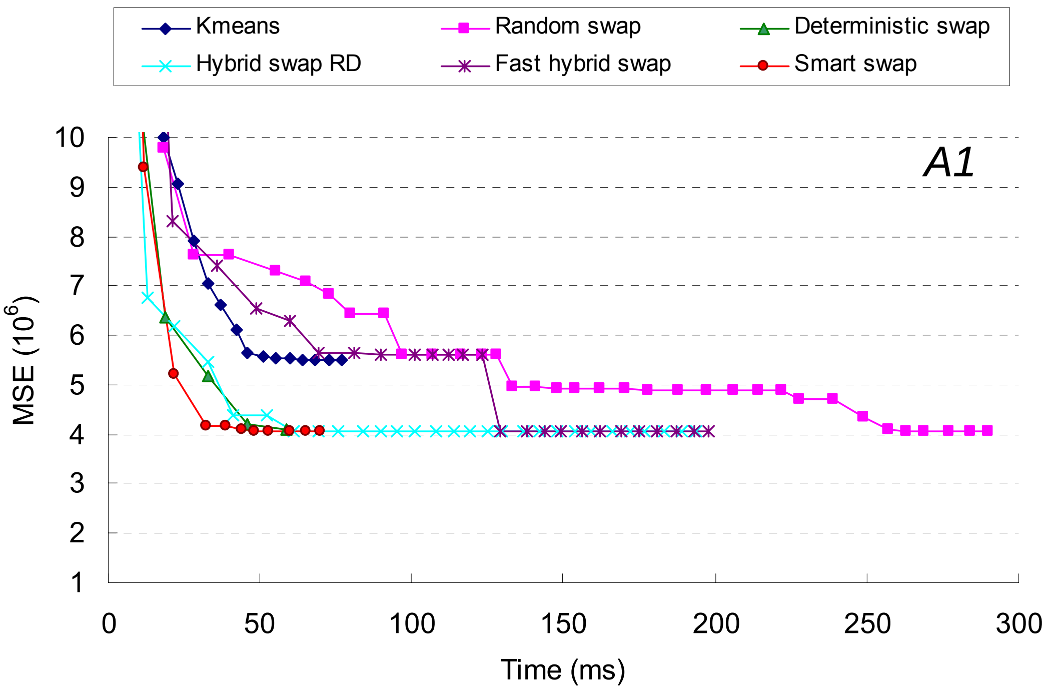


Figure 16. Time-distortion efficiency of clustering methods on dataset *A1*.

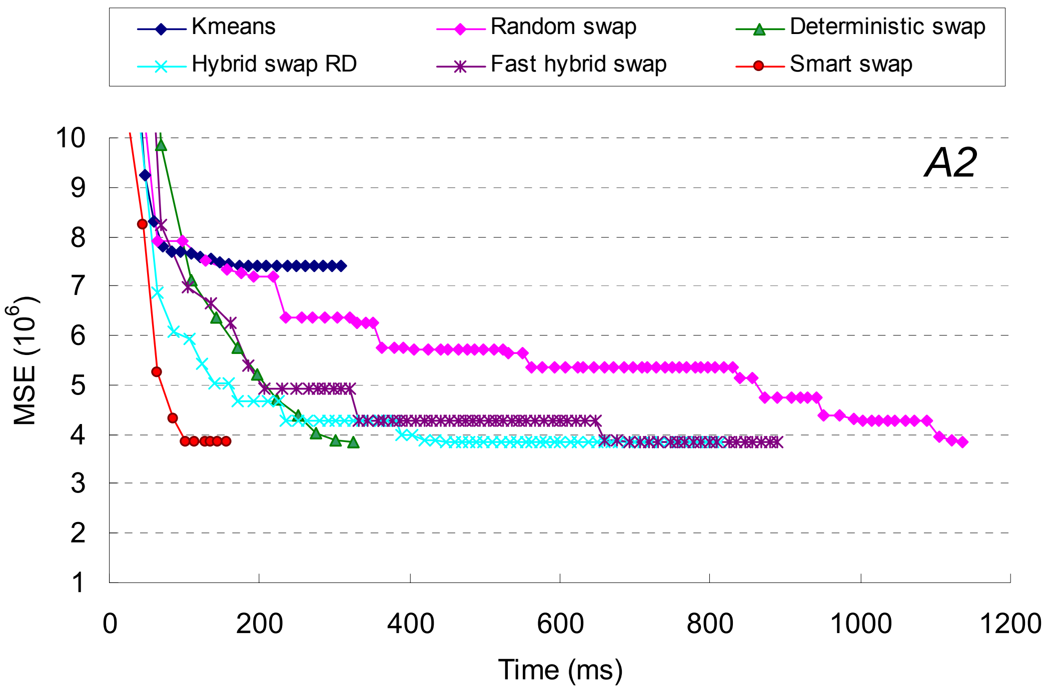


Figure 17. Time-distortion efficiency of clustering methods on dataset *A2*.

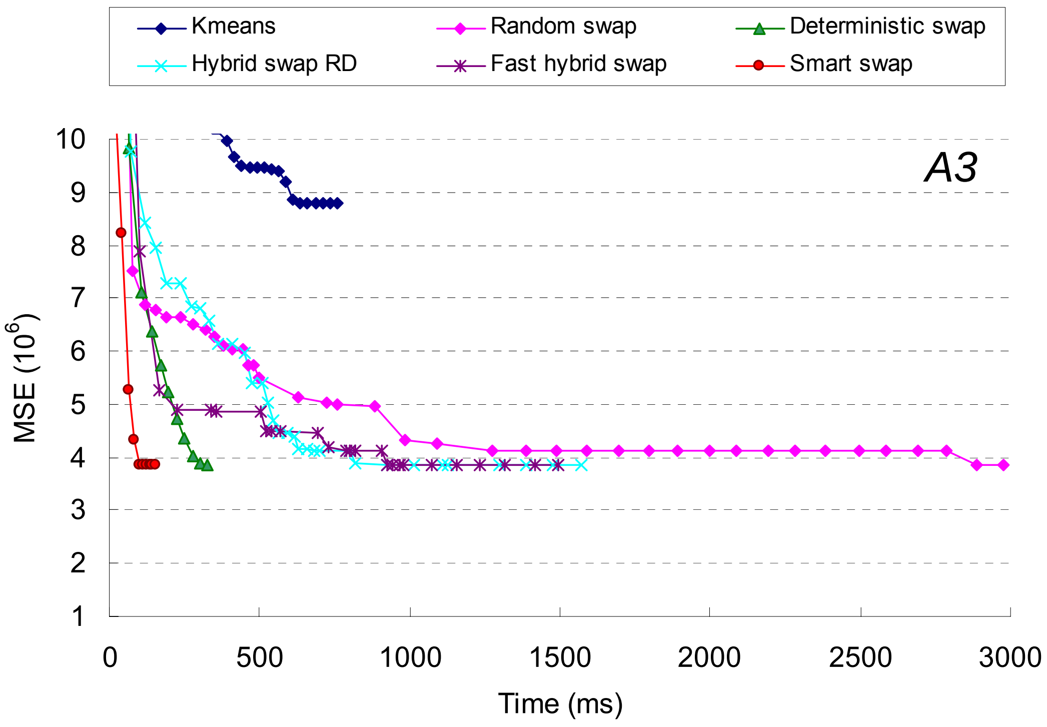


Figure 18. Time-distortion efficiency of clustering methods on dataset *A3*.

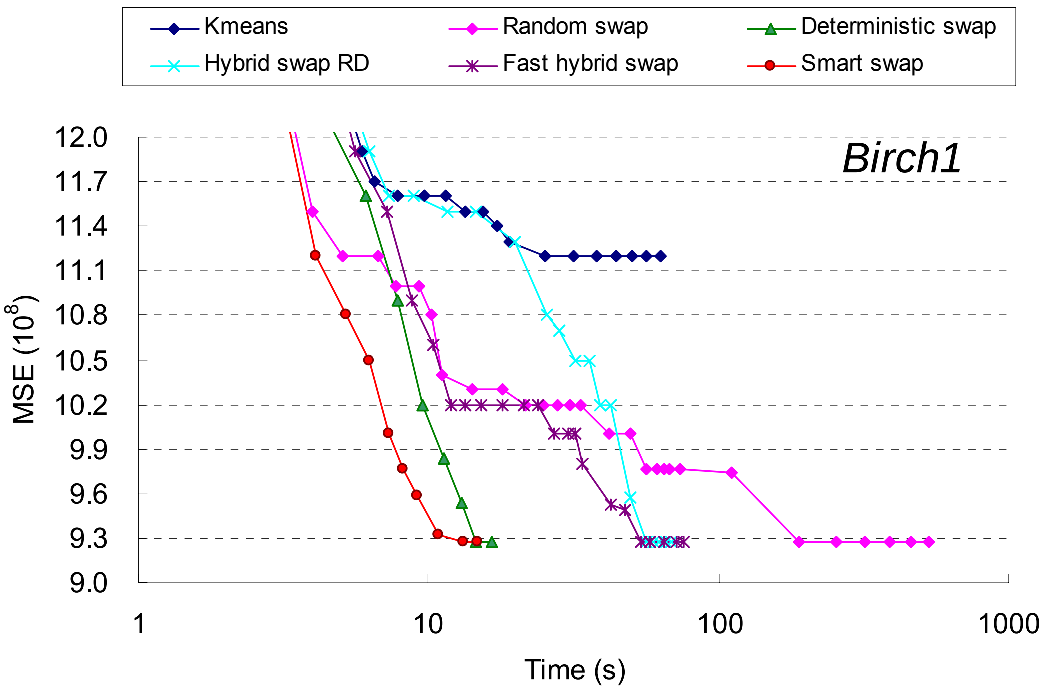


Figure 19. Time-distortion efficiency of clustering methods on dataset *Birch1*.

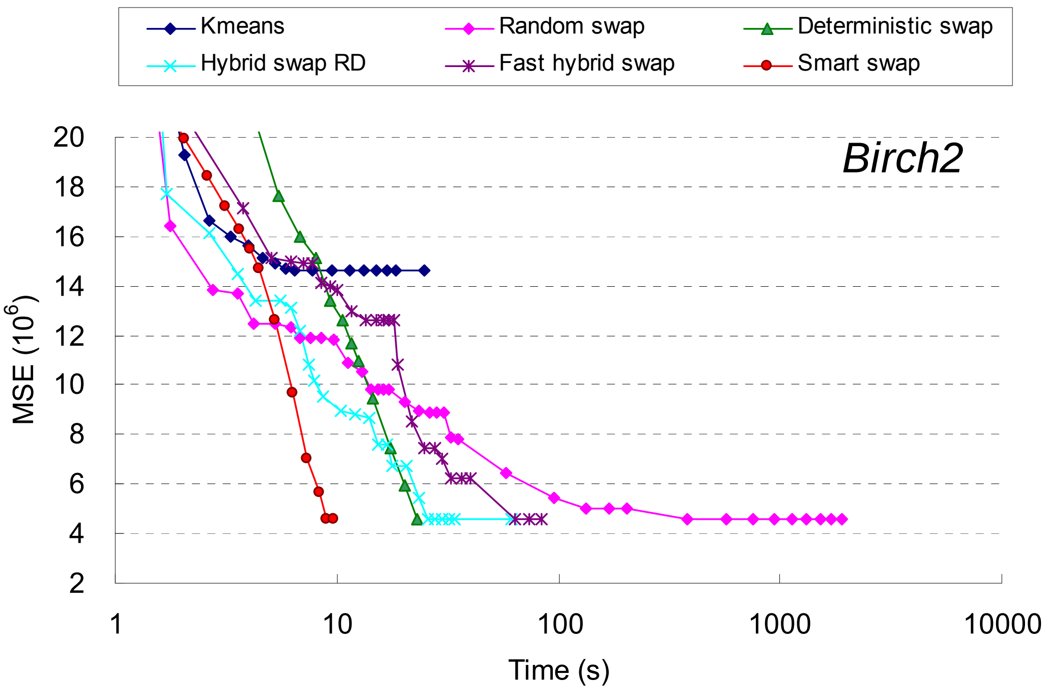


Figure 20. Time-distortion efficiency of clustering methods on dataset *Birch2*.

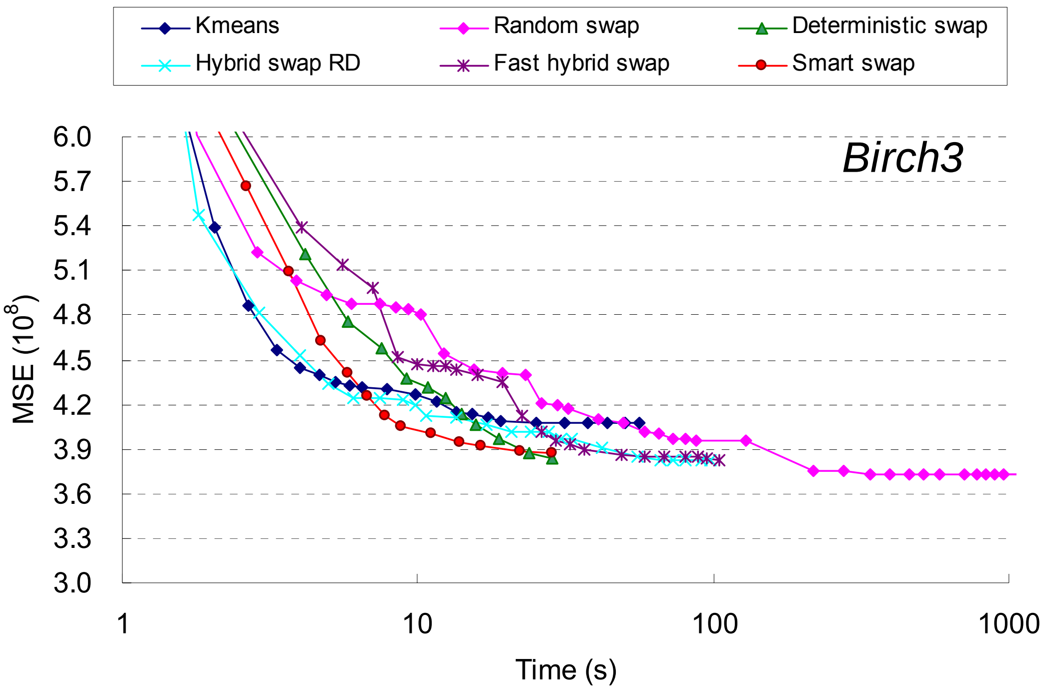


Figure 21. Time-distortion efficiency of clustering methods on dataset *Birch3*.

4.2 Real datasets

We next compare the clustering methods on the real datasets described in Table 4. *Iris* is a famous 4-dimentional iris data set [29] that contains 150 instances with three classes; each class refers to one type of iris plant and has four attributes: sepal length, sepal width, petal length and petal width. *Bridge* is a 16-dimensional image dataset that consists of 4 × 4 spatial pixel blocks sampled from the image (8 bits per pixel); the dataset is very sparse and does not have clear cluster boundaries. *House* is a 3-dimensional image dataset that consists of the RGB color vectors with three attributes (red, green and blue color value). In comparison to *Bridge*, *House* has more compact clusters observed from Figure 22.

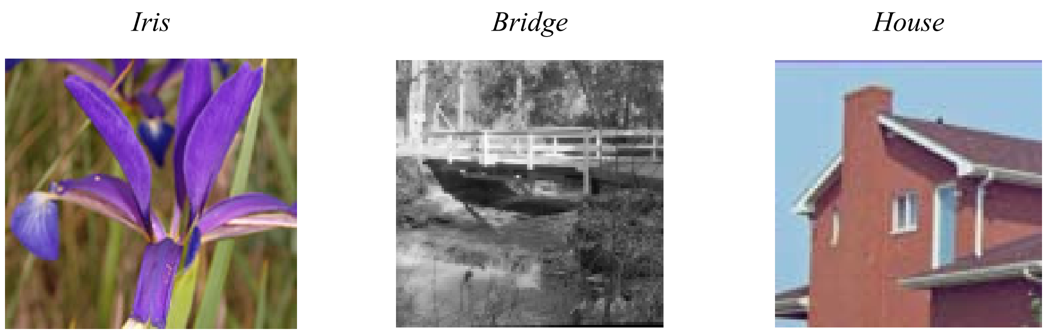


Figure 22. Sources for the real datasets.

The first experiment is to compare the swap-based clustering algorithms and k-means on dataset *Iris* with different number of clusters *M* from 2 to 10. Since the neighborhood of the dataset is not easy to be estimated, we set the number of iterations for the random swap, hybrid swap RD and fast hybrid swap D^2R to a fixed number 150, which is the size of the dataset *Iris*. Optimal solution is estimated by the random swap by setting the number of iterations to 1000, which is much bigger than the size of the dataset 150. The clustering quality of each algorithm, measured by MSE, is shown in Table 15. Furthermore, to verify the stability, the above algorithms are run 5 times on the dataset *Iris* with 10 clusters. The MSE value of each run and the standard deviation of each algorithm are shown in Table 16.

Observed from Tables 15-16, there are following observations:

- When the number of clusters *M* is small (from 2 to 5) all methods achieve almost the optimal clustering. While increasing the *M* (from 6 to 10), random swap achieves the best clustering, and next is the hybrid swap RD. Deterministic swap, smart swap and fast hybrid swap D^2R are near the optimal clustering. K-means is still far from the optimal solutions.
- Regarding the clustering quality, the random swap can be clearly ranked first in all the clustering methods even with large number of iterations. The other swap-based methods have the problem of getting in a local minimum, but in these cases not very far from the optimal.
- Regarding the stability of algorithm, random swap is the most stable (0.0004) and deterministic swap is the worst (0.0053) among those swap-based clustering methods excluding K-means.

Table 15. Clustering quality (MSE) of swap-based clustering algorithms for *Iris* dataset with different number of clusters *M* from 2 to 10.

Algorithms	2	3	4	5	6	7	8	9	10
Random swap	1.0158	0.5263	0.3821	0.3102	0.2595	0.2279	0.1992	0.1851	0.1732
Deterministic swap	1.0158	0.5263	0.3835	0.3106	0.2616	0.2290	0.2013	0.1876	0.1796
Hybrid swap	1.0158	0.5263	0.3821	0.3119	0.2613	0.2279	0.1992	0.1886	0.1735
Fast hybrid swap	1.0158	0.5263	0.3823	0.3103	0.2597	0.2309	0.2018	0.1856	0.1762
Smart swap	1.0158	0.5263	0.3821	0.3104	0.2616	0.2314	0.2034	0.1902	0.1740
K-means	1.0158	0.5263	0.3823	0.3104	0.2618	0.2900	0.2603	0.2347	0.2036
Optimal	1.0158	0.5263	0.3821	0.3102	0.2595	0.2279	0.1992	0.1851	0.1721

Table 16. Clustering quality (MSE) of swap-based clustering algorithms for *Iris* dataset with 10 clusters run 5 times.

Run	Random swap	Deterministic swap	Hybrid swap	Fast hybrid swap	Smart swap	K-means
1	0.1730	0.1793	0.1769	0.1740	0.1756	0.2023
2	0.1730	0.1738	0.1766	0.1758	0.1763	0.2322
3	0.1730	0.1774	0.1759	0.1738	0.1777	0.1814
4	0.1730	0.1882	0.1789	0.1721	0.1745	0.1786
5	0.1721	0.1798	0.1733	0.1773	0.1757	0.1965
Standard deviation	0.0004	0.0053	0.0020	0.0020	0.0012	0.0215

We will next focus on the efficiency of those swap-based algorithms on the real datasets. The image datasets *Bridge* and *House*, which have bigger size and higher dimensional attributes, are used in the experiment. Since the number of clusters (*M*) of the image datasets is not a priori value, we set *M* to a fixed number 64. The number of iterations needed for the random swap, hybrid swap RD and fast hybrid swap D^2R is set to 4000 for datasets *Bridge* and *House*. The time-distortion efficiency results of clustering methods are illustrated in Figures 23-24.

For the dataset *Bridge* (see Figure 23), the comparison of efficiency of all mentioned algorithms is very similar, even k-means is competitive. They can achieve the clustering quality MSE near 4.2 (10^3) within 1000 milliseconds. Random swap and fast hybrid swap D^2R achieve the best clustering quality with MSE 4.1 (10^3); the others could not avoid the problem to getting stuck in a local minimum.

For the dataset *House* (see Figure 24), the comparison of efficiency is more clear. The proposed smart swap is the most efficient. It achieves the estimated optimal clustering quality with MSE 70 in 6 seconds. Deterministic swap also achieves the competitive efficiency with smart swap. Moreover, the random swap, hybrid swap RD and fast hybrid swap D^2R achieve better clustering quality with MSE about 69.

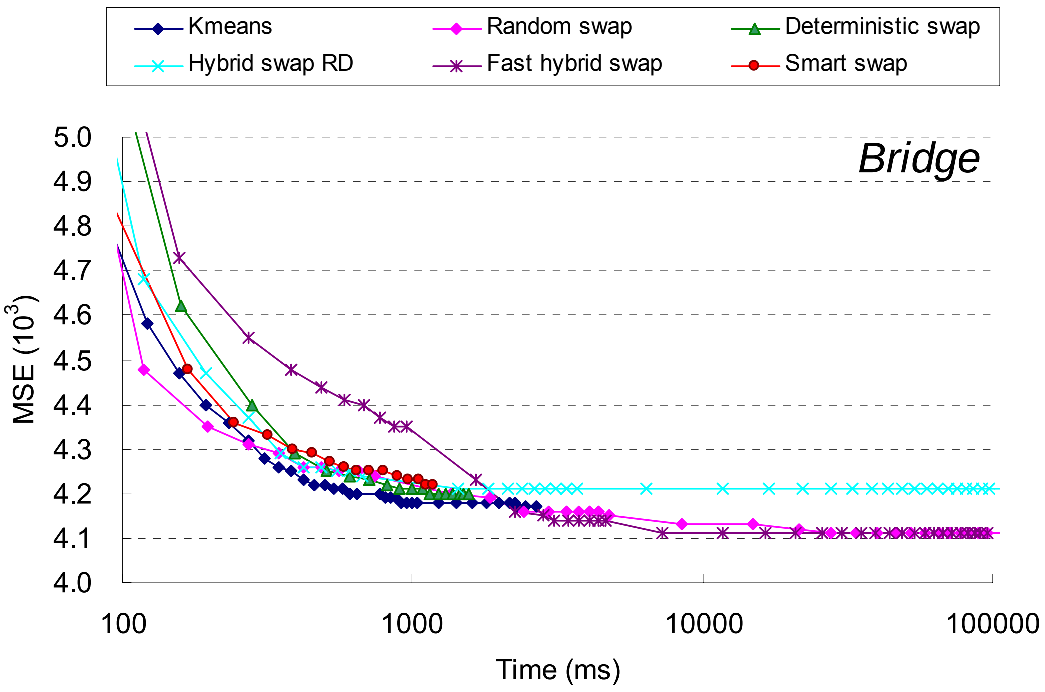


Figure 23. Time-distortion efficiency of clustering methods on dataset *Bridge*.

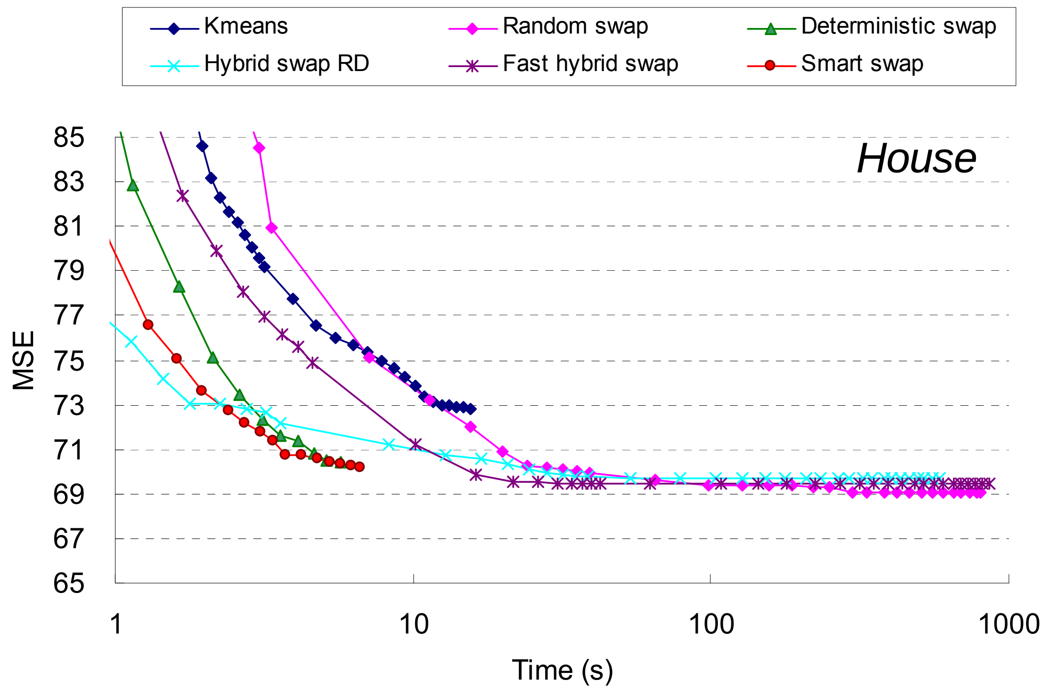


Figure 24. Time-distortion efficiency of clustering methods on dataset *House*.

4.3 Result comparisons

We have compared the clustering quality and time-distortion of the five swap-based clustering methods, random swap, deterministic swap, hybrid swap RD, fast hybrid swap D^2R and proposed smart swap, on both synthetic datasets and real datasets in the above experiments.

From the clustering quality point of view, random swap always achieves the best clustering quality or even the optimal result in case the number of iterations is set high enough. Hybrid swap RD and fast hybrid swap D^2R can also produce approximately as good results as random swap for the synthetic datasets but are little bit weaker for image datasets.

The proposed smart swap is capable of achieving almost optimal clustering for the synthetic datasets with clear clusters, e.g. *S1*, *A1*, *A2*, *A3*, *Birch1* and *Brich2*, and competitive clustering for the other synthetic datasets where clustering structure is not very clear (*S2*, *S3* and *Birch3*). However, its performance on the image datasets is not the best especially when the datasets do not have clear cluster boundaries such as *Bridge*, this is a common issue for the hybrid swap RD and fast hybrid swap D^2R as well. Deterministic swap achieves good clustering in most cases, but its main problem is getting stuck in a local minimum easily.

From the clustering efficiency point of view, the smart swap is the most efficient for most datasets (*S1*, *A1*, *A2*, *A3*, *Birch1*, *Birch2*, *Birch3* and *House*). Moreover, it becomes more efficient with the higher *M* value (the number of clusters). It is no doubt that the random swap is the most inefficient among these five swap-based clustering methods due to a large number of iterations required for better clustering quality. The other swap-based clustering methods, deterministic swap, hybrid swap RD and fast hybrid swap D^2R, perform similarly to each other without any surprises.

In summary, the swap-based clustering methods, as heuristics clustering, can achieve good performance both in terms of clustering quality and efficiency. It is a good choice to select random swap to achieve the best clustering. On the other hand, the proposed smart swap is recommended to be chosen from the efficiency viewpoint.

5 Application to Location-based Services

5.1 Location-based services

Location-based services (LBS) are information services accessible via mobile devices through mobile network and utilizing the ability to make use of location of the mobile device [30, 31].

In general, the LBS consist of the following components [30].

- Mobile device
- Communication network
- Positioning component
- Service and application provider

Mobile device is a tool for the user to request needed information in terms of text, pictures and speech. Possible devices are mobile phones (most used), PDA's, laptops, and even navigators used in the car. The communication network can be mobile network, wireless network, or internet, which transfer the user data and services request from the mobile devices to the services provider and then the requested information back to the user. For the positioning component, user location should be determined to process the services either by using *Global Positioning System* (GPS), or by using the mobile communication networks. Further possibilities to obtain the position are WLAN stations, active badges or radio beacons, which can be used especially for indoor navigation in a museum for example. Moreover, the position can also be specified by the user if it cannot be determined automatically. The service provider (application) offers a number of different services such as finding a route, and searching specific information on objects of user interest so-called point of interest (POI).

Considering an example of searching a Chinese restaurant nearby, user sends the request from his mobile, with his location information obtained by GPS, to the service provider via mobile network, and then receives the requested information displayed on his mobile. The example is illustrated in Figure 25.

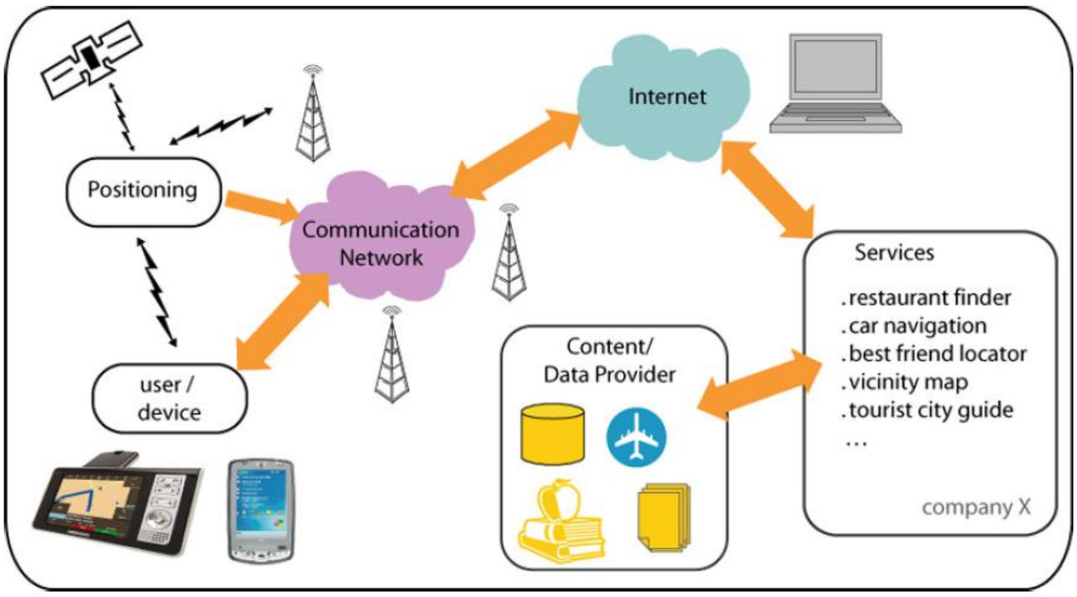


Figure 25. LBS components and information flow [30].

Location-based services can be used in a variety of contexts, such as work and personal life [32]. In a simplest form it is a positioning service, informing the user about her/his current location (Where am I?) [31]. Examples of location-based services are listed as below [32, 33].

- Searching the nearest business or service, such as an ATM or hotel
- Turn by turn navigation to a given address or planning a route
- Locating people on a map displayed on the mobile phone
- Location-based mobile advertising
- Tracking for the workforce, fleet management
- Receiving alerts, such as notification of a sale on gas or warning of a traffic jam
- Social networking, such as friend-finder or instant messaging

More examples can be found in [30].

Nowadays, location-based services power Mobile Local Search to enable the search and discovery of persons, places, and things within an identifiable space defined by distinct parameters, such as social networks, individuals, cities, neighborhoods, landmarks, and actions that are relevant to the searcher’s past, current, and future location. These parameters provide structure to vertically deep and horizontally broad data categories that can stand-alone or are combined to comprise searchable directories [34].

5.2 MOPSI project

MOPSI project[1] develops LBS applications in Speech and Image Processing Unit, School of Computing of University of Eastern Finland. MOPSI is a Finnish abbreviation for **Mo**biilit **P**aikkatieto**S**ovellukset ja **I**nternet which can be translated in English as “Mobile Location-based Applications and Internet”. The MOPSI project implements different location-based services and applications such as mobile search engine, data collection, user tracking and route recording. The project has its applications integrated into both the web and mobile phones with the aim to integrate user location as a search option. In the following sub sections the main features of MOPSI will be described.

5.2.1 Search engine

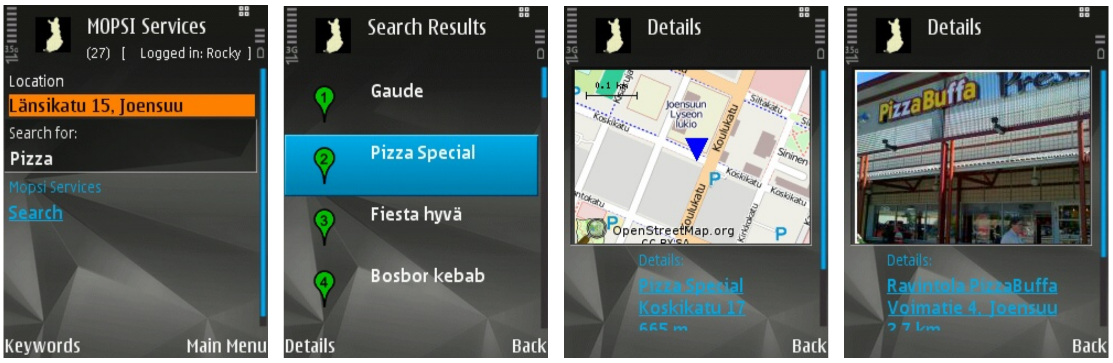
Current search engines, such as Google, Bing, Yahoo, are very powerful but fail to utilize one important aspect of relevance: the location of the user because of two reasons. Firstly the user location was not as widely available as nowadays due to GPS phones being less frequent. Secondly, the location information is rarely attached in the web pages [35].

Existing LBS search engines such as Google maps[2] and Yellow pages[3], use databases where all entries, such as hotel directories, restaurant lists or common similar services, have been explicitly geo-referenced beforehand. The main drawback is that the data must be collected beforehand, which makes the systems depend on the providers to put efforts to keep the information up-to-date [35].

MOPSI search engine is a combination of traditional location-based service and search engine. It first retrieves data from the local database similarly as Google maps, then queries relevant data from the user collections attached with photos, and finally performs location-based search from web as originally proposed in [36], and later implemented in practice as summarized in [37]. The key idea is to use ad-hoc geo-referencing of the web pages based on address detection within the body text [38], rather than relying on geo-tags or address tags which rarely exist.

[1] http://cs.joensuu.fi/mopsi/
[2] http://maps.google.com
[3] http://en.fonecta.fi/yellow-pages.html

User can access MOPSI search from a GPS-supported mobile with Java application (Figure 26) or via web (Figure 27).


MOPSI Services

(27) [Logged in: Rocky]

Location: Länsikatu 15, Joensuu

Search for: Pizza

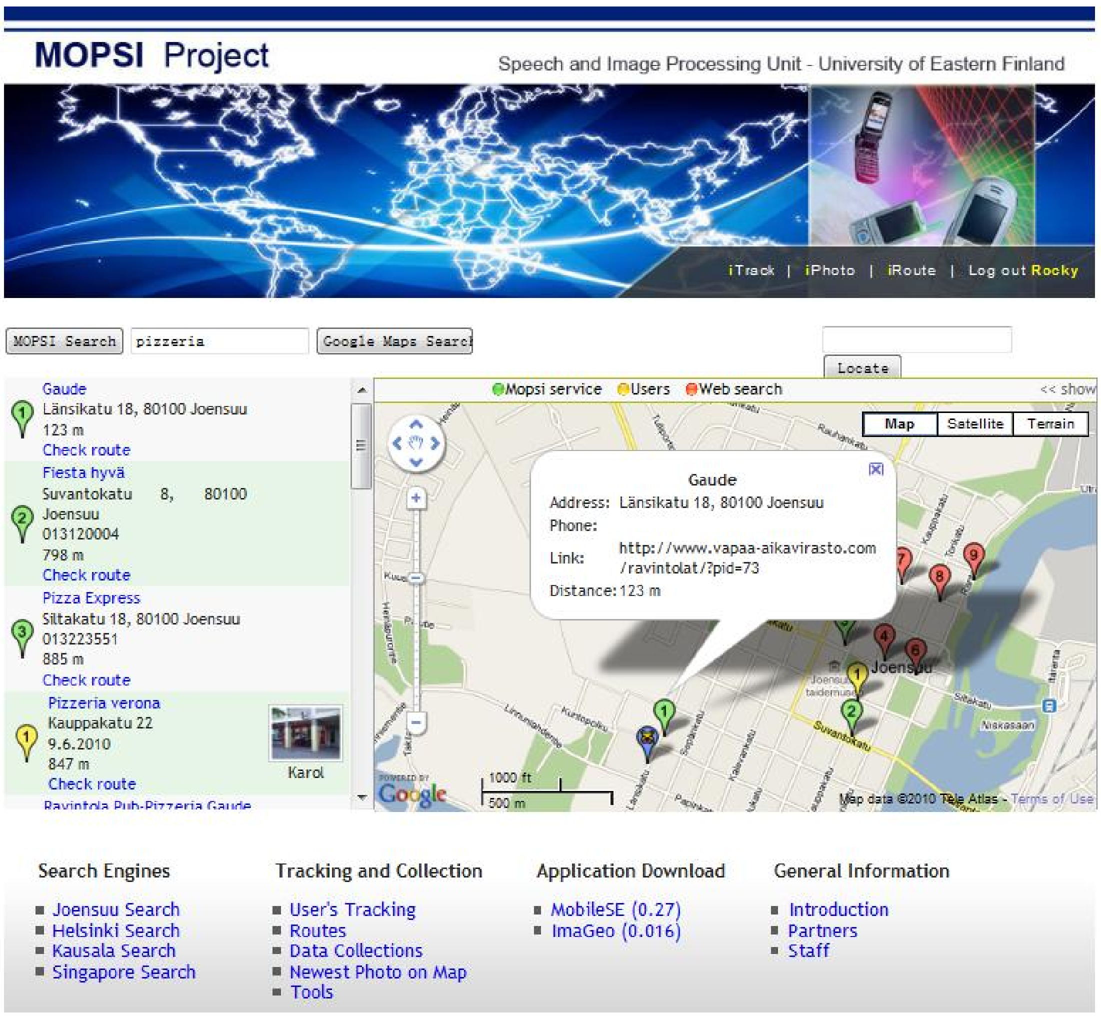
Search Results

1 Gaude
2 Pizza Special
3 Fiesta hyvä
4 Bosbor kebab

Details: Pizza Special, Koskikatu 17, 665 m

Details: Ravintola PizzaBuffa, Voimatie 4, Joensuu, 2.7 km

Figure 26. Mobile Java application interfaces for MOPSI search engine.


MOPSI Project

Speech and Image Processing Unit - University of Eastern Finland

iTrack | iPhoto | iRoute | Log out Rocky

MOPSI Search | pizzeria | Google Maps Search | Locate

1 Gaude
Länsikatu 18, 80100 Joensuu
123 m
Check route

2 Fiesta hyvä
Suvantokatu 8, 80100 Joensuu
013120004
798 m
Check route

3 Pizza Express
Siltakatu 18, 80100 Joensuu
013223551
885 m
Check route

1 Pizzeria verona
Kauppakatu 22
9.6.2010
847 m
Check route
Karol

Ravintola Pub-Pizzeria Gaude

Gaude
Address: Länsikatu 18, 80100 Joensuu
Phone:
Link: http://www.vapaa-aikavirasto.com/ravintolat/?pid=73
Distance: 123 m

Search Engines
- Joensuu Search
- Helsinki Search
- Kausala Search
- Singapore Search

Tracking and Collection
- User's Tracking
- Routes
- Data Collections
- Newest Photo on Map
- Tools

Application Download
- MobileSE (0.27)
- ImaGeo (0.016)

General Information
- Introduction
- Partners
- Staff

Figure 27. Web page interface for MOPSI search engine.

Given location, user can make search queries similarly as done with normal search engine, but instead of providing relevant search results by the content alone, the location of the user is used to restrict the results only near-by. Results will be provided if they exist in the local database (green bubbles), in user collection (yellow bubbles), and additional results are searched from internet (red bubbles).

5.2.2 Photo collection

MOPSI also provides a service that user can collect photos via mobile application, and then view them with the locations on a map visually. The Google maps API is used in the implementation. Figure 28 demonstrates the map view of the photo collection. User can find relevant information of the photo, such as time stamp, address and description. The time line view of the photos is also available. The MOPSI photo collection could be used as a travel documentation of the user.

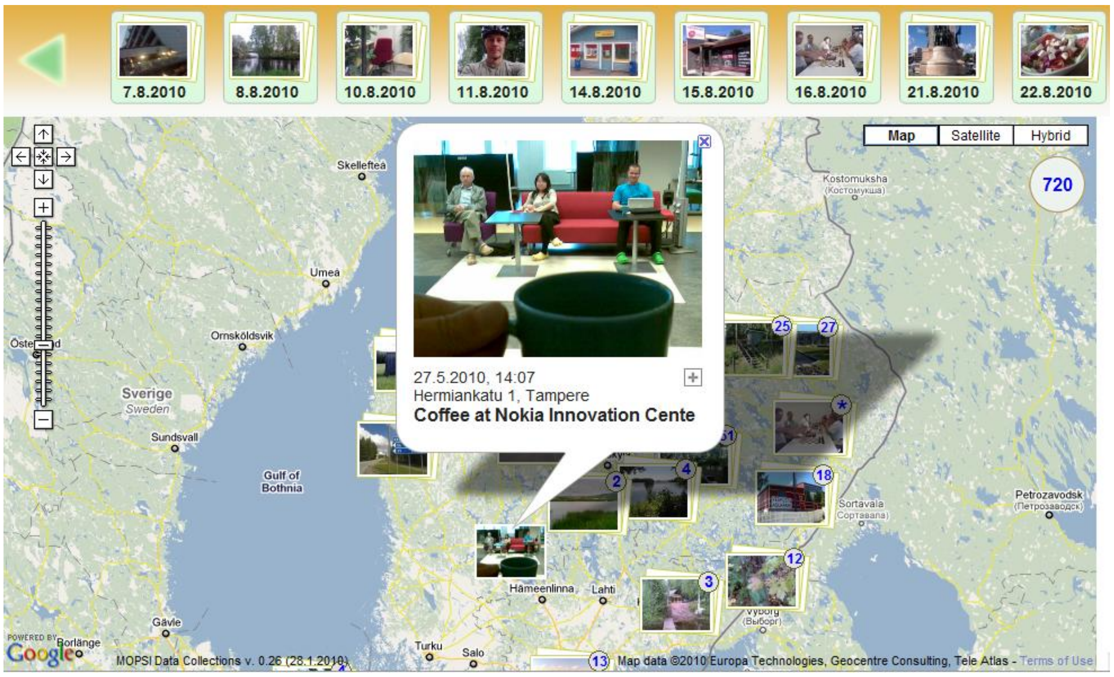
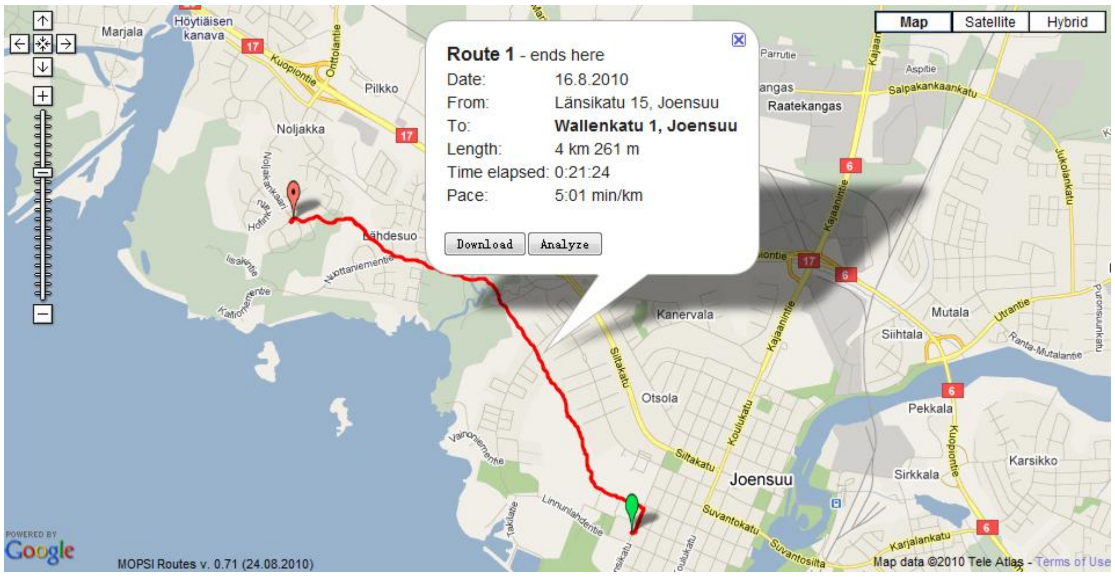


Figure 28. Map view of user photo collection in MOPSI.

5.2.3 Route tracking

In MOPSI collection, tracking user’s routes is the other main feature. User records her/his route anywhere on Earth via MOPSI mobile application. Tracks are traced on Google maps in the MOPSI web page. Moreover, detailed information such as addresses, duration, and speed can also be analyzed, as shown in Figure 29.


Route 1 - ends here

Date: 16.8.2010

From: Länsikatu 15, Joensuu

To: **Wallenkatu 1, Joensuu**

Length: 4 km 261 m

Time elapsed: 0:21:24

Pace: 5:01 min/km

Download Analyze

MOPSI Routes v. 0.71 (24.08.2010)

Figure 29. Map view of a user route in MOPSI.

5.2.4 User tracking

MOPSI user tracking views all of its users latest recorded location as stored either in the route or photo collection. This is visualized in Figure 30. With MOPSI user tracking, user can share her/his location with friends similarly done as in Google latitude[4].

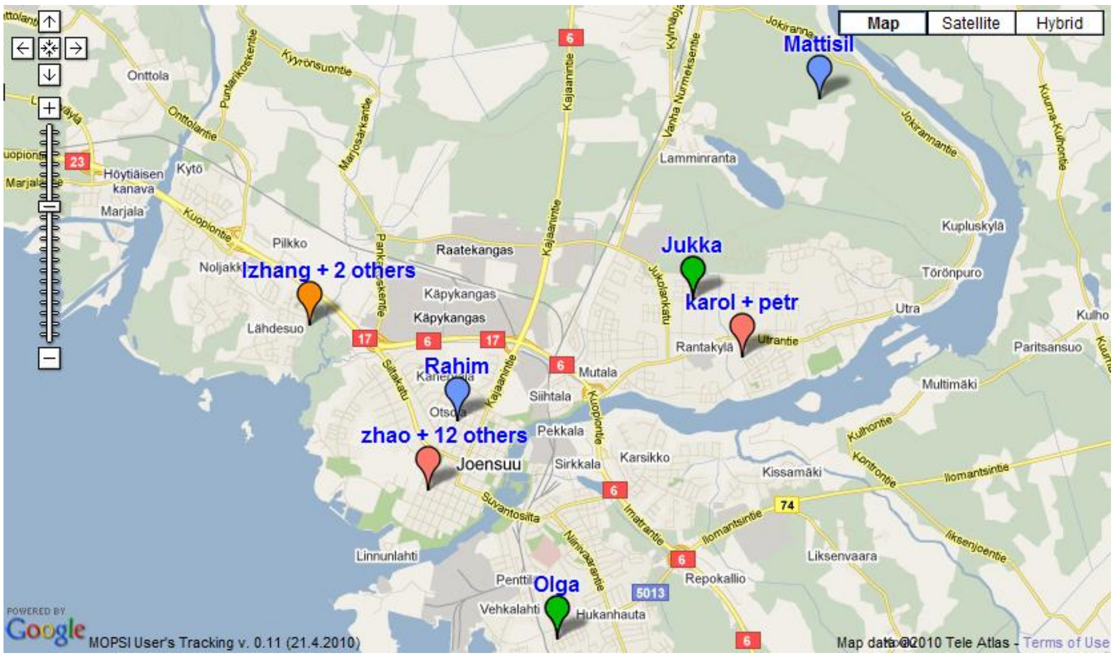

MOPSI User's Tracking v. 0.11 (21.4.2010)

Figure 30. Map view of user locations in MOPSI.

[4] http://www.google.com/intl/en_us/latitude/intro.html

5.3 Clustering in MOPSI

It is a very natural way to share geographic information on the web using Web Mapping System (WMS) such as Google maps. MOPSI uses Google maps to share user photo collections, routes and locations (see Figures 28, 29, 30). User can zoom and pan the maps to concern the content on the map.

However, clutter becomes the main problem of visualizing a large number of overlays such as user location bubbles on the map in a larger map scale (see Figure 31). Clutter not only reduces the map background visibility but also hinders the users understanding of the structure and content of the data on the map [39].

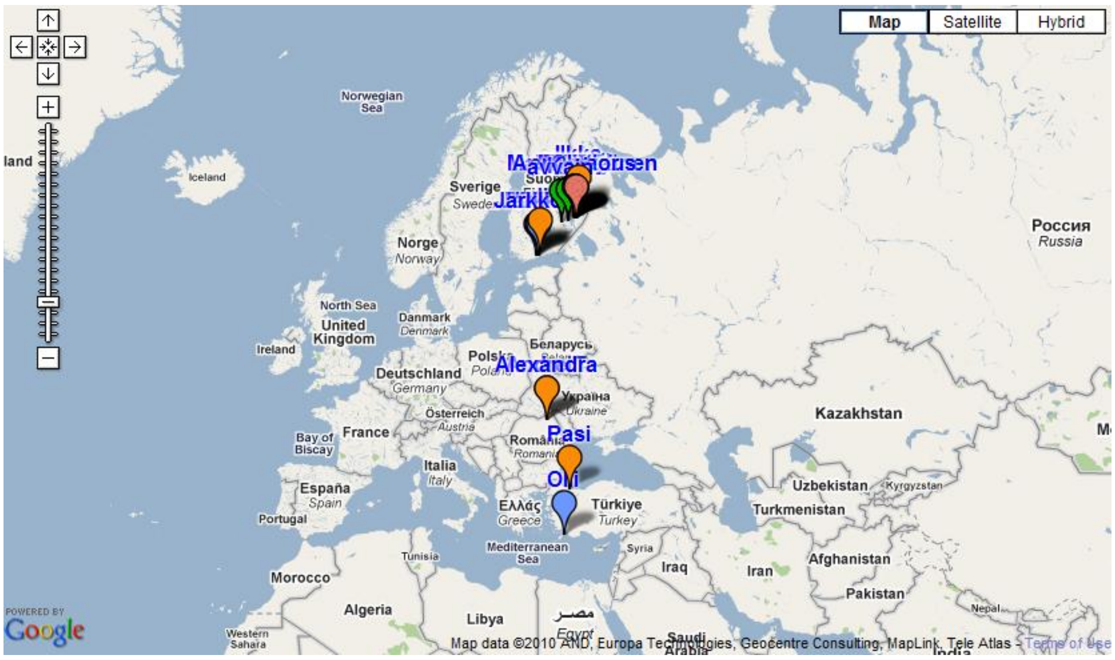

Map
Satellite
Hybrid
Alexandra
Pasi
Oli
Jarkko
Map data ©2010 AND, Europa Technologies, Geocentre Consulting, MapLink, Tele Atlas - Terms of Use

Figure 31. Clutter of map visualization in MOPSI user tracking.

The clutter problem is actually that there are too many data overlays that can be fit on the map screen. Clustering is therefore needed to reduce the amount of data overlays to be represented on the map. In [39], it presents a hierarchical aggregation, which is a common visualization technique to make visual representations more visually scalable and less cluttered [40]. The method creates a hierarchical clustering tree, which can be subsequently used to extract clusters for a given map scale without cluttering the map. The main drawback of this technique is the high time computation, which makes it impractical for a real-time web application.

5.3.1 Split smart swap clustering

We apply a clustering technique using smart swap, one efficient swap-based clustering method, to MOPSI user tracking web application for solving this clutter problem.

In MOPSI user tracking, the user locations are represented on the map visually with bubbles (see Figure 31). For a given map scale, we calculate the minimum distance d_{min} of the two bubbles that can be represented separately from each other on the map. Observed from the Google maps, it has 22 scales (zoom levels) from 0 to 21, and in scale 0 the minimum distance $D_0 = 2500$ km can separate two bubbles converted to pixel coordinates on the map. The minimum distance d_{min} for each scale s is calculated as follows:

$$d_{\min} = \frac{D_0}{2^s} \quad \forall\, s \in [0, 21] \qquad 5.3.1$$

In general, with increasing the number of clusters M, the distance of the two nearest clusters d_{near} becomes smaller and smaller. The solution to solve the clutter problem is therefore to find a clustering with a proper M for each scale, where d_{min} is close to the d_{near} (see Figure 32).

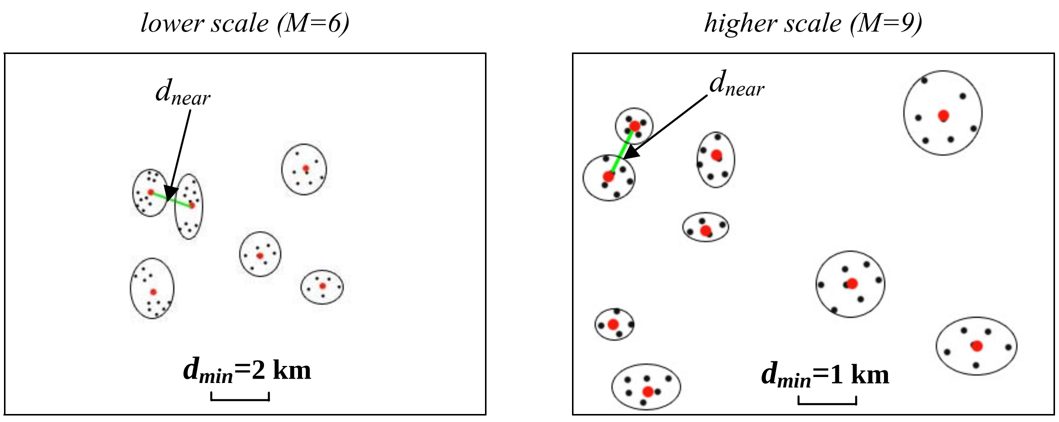
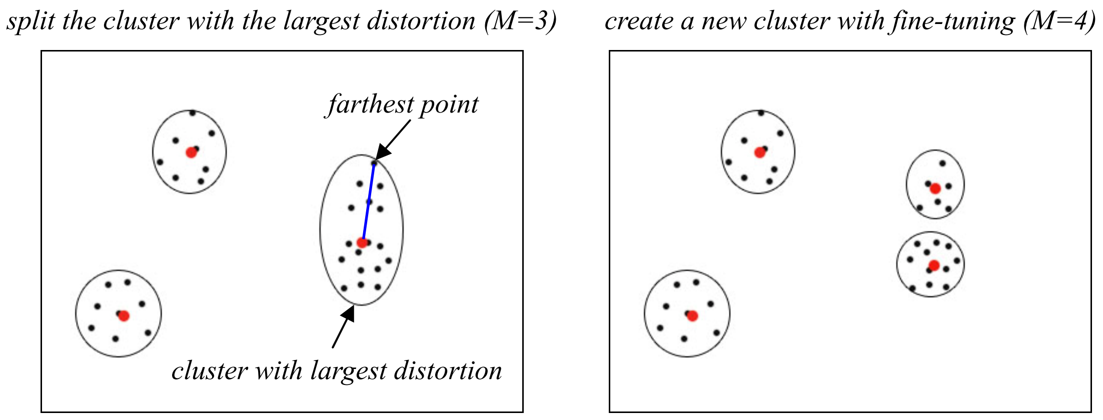


Figure 32. The relation of M, d_{near} and d_{min} in different map scales.

We propose a so-called *split smart swap* clustering algorithm to do the clustering for the locations dataset with M from 1 to N (maximum is the size of the locations dataset). The main idea of the split smart swap is to split the cluster that has the largest distortion, into two clusters, and then apply the smart swap to optimize (fine-tune) the clustering results at each iteration step (see Figure 33). After the split smart swap clustering, it outputs the *centroids array*, which stores the centroids of clusters with M from 1 to N, and the *distance array* which stores the d_{near} corresponding to the number of clusters M.



split the cluster with the largest distortion (M=3)
create a new cluster with fine-tuning (M=4)
farthest point
cluster with largest distortion

Figure 33. Adding a new cluster by selecting the farthest point in the cluster with the largest distortion.

```
SplitSmartSwap Clustering Algorithm

N;    // size of the dataset X
CentroidsArray[N]; // centroids for the clusters with number from 1 to N
DistanceArray[N]; // nearest distance for the clusters with number from 1 to N
M ← 1; // start from 1 cluster at the beginning
C, P ← assignData(X, M); // optimal partition

WHILE M < N
    M ← M + 1; // add a new cluster
    maxIndex ← GetLargestDistortion(C, P); // find the cluster with largest distortion
    // set the farthest point as the centroid of the new cluster
    $C_m$ ←GetFarthestPoint(C, maxIndex);
    SmartSwap(X, M); // clustering optimization
    CentroidsArray[M] ← C; // add the centroids to the array
    $d_{near}$ ← FindNearestDistance(C); // the distance of nearest two centroids
    DistanceArray[M] ←$d_{near}$; // add the nearest distance

RETURN CentroidsArray, DistanceArray
```

Figure 34. Pseudo-code of split smart swap clustering algorithm.

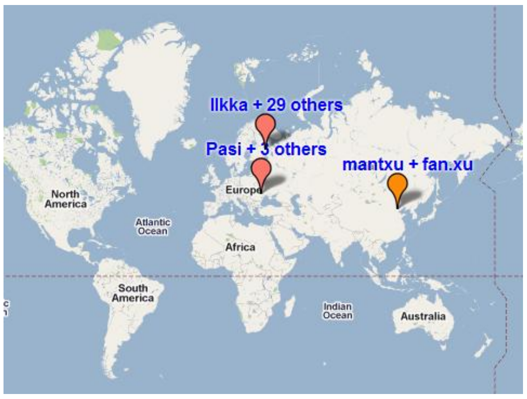
For a given map scale, the clustering solution can be found as selecting the clustering with such number of clusters M, for which $d_{near} \leq d_{min}$ (see Figure 35). After that, we select the centroids from the *centroids array* with the corresponding M. The clustering result can therefore be obtained via optimal partition with the chosen centroids according to Equation (3.1.2).

distance:	64	42	36	…	4	2	1
M:	1	2	3	…	30	31	32

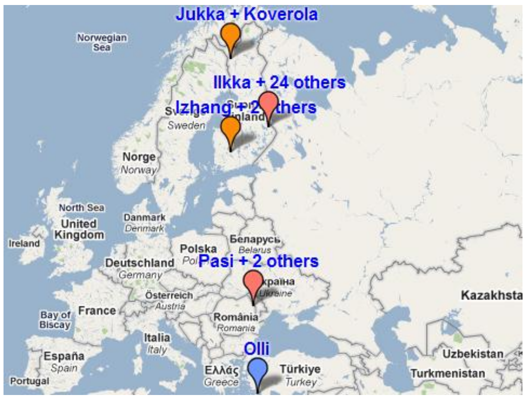
Figure 35. An example of selecting M. Assuming that d_{min}=39, the close d_{near} in the distance array is 36, and the corresponding M=3 is selected.

The clustering results using split smart swap for user tracking with different map scales are show in Figure 36.

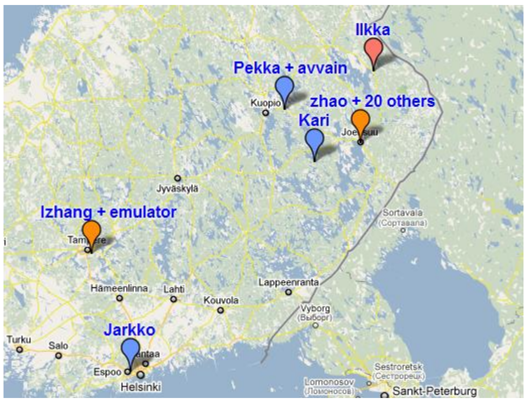
Google maps scale 1 (Global)



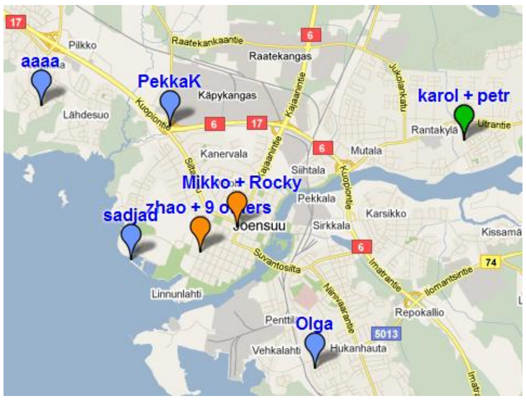
Google maps scale 3 (Europe)



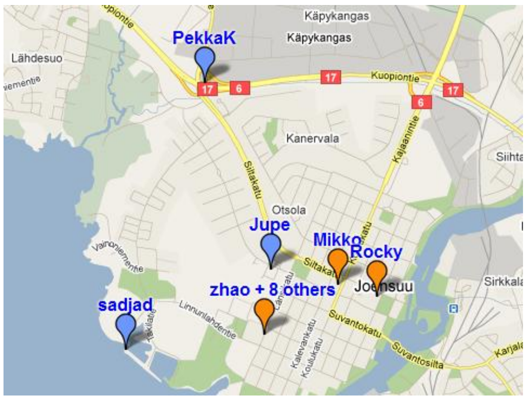
Google maps scale 6 (Finland)



Google maps scale 12 (Joensuu)



Google maps scale 13 (City center)



Google maps scale 19 (Länsikatu road)

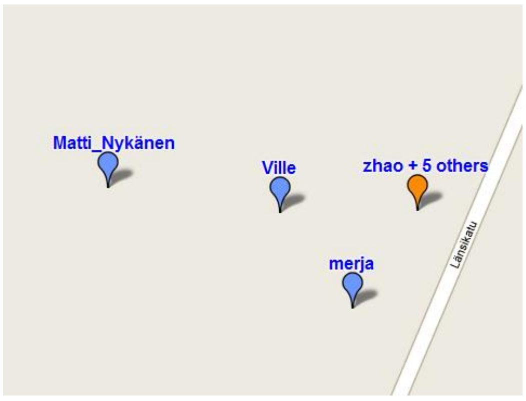


Figure 36. Split smart swap for user tracking with different map scales.

Observed from Figure 36, the visualization of user locations on the map is scalable and very less visually cluttered. With increasing the scale levels, the number of clusters becomes bigger and bigger so as to reduce the clutter on the map.

The above clustering method was coded in JavaScript and run in the web application in Firefox browser using 1.6 GHz Intel CPU computer (same as below). The processing time is about 115 milliseconds for the clustering at the beginning. Afterwards it takes only few milliseconds when zooming map. The number of clusters *M* needed and d_{min} in each scale are illustrated in Table 17.

Table 17. The number of clusters *M* and minimum distance d_{min} (km) in each scale.

Scale	0	1	2	3	4	5	6	7	8	9	10
M	2	3	4	7	7	7	12	13	14	14	19
d_{min}	2500	1250	625	313	156	78.1	39.1	19.5	9.77	4.88	2.44
Scale	11	12	13	14	15	16	17	18	19	20	21
M	21	25	27	27	27	27	28	31	31	31	31
d_{min}	1.22	0.61	0.31	0.15	0.08	0.04	0.02	0.01	0.005	0.002	0.001

To evaluate the efficiency of the split smart swap for more data, we use the method to cluster the photos locations in photo collection web page. The processing time for different number of photos is listed in Table 18.

Table 18. The processing time for different number of photos using split smart swap.

Number of photos	31	47	86	145	177	219
Processing time (ms)	86	141	1145	5644	10800	21123

Observed from Table 18, the processing time remains real-time (<1s) only up to 80 photos and becomes huger with increasing the number of photos (Note: in practice, the processing time is also influenced by different browsers). The main advantage of split smart swap is that the clustering method is applied only once for all scales. However, the drawback is that it is too slow for big dataset, which is a bottleneck for real-time applications.

5.3.2 Grid-based clustering with bounding box

In the split smart swap, we consider the clustering solution for all map scales. Once the clustering is done at the beginning, the proper clusters for each scale can be selected according to the distance array and centroids array. It works for real-time applications if the size of dataset is small. However, the processing time becomes a bottleneck of the method if the dataset is larger.

Instead of solving all map scales at once, we consider another approach by solving only the current map scale. It uses a bounding box on the map to limit the amount of data objects. Only those data objects that are inside the bounding box are clustered.

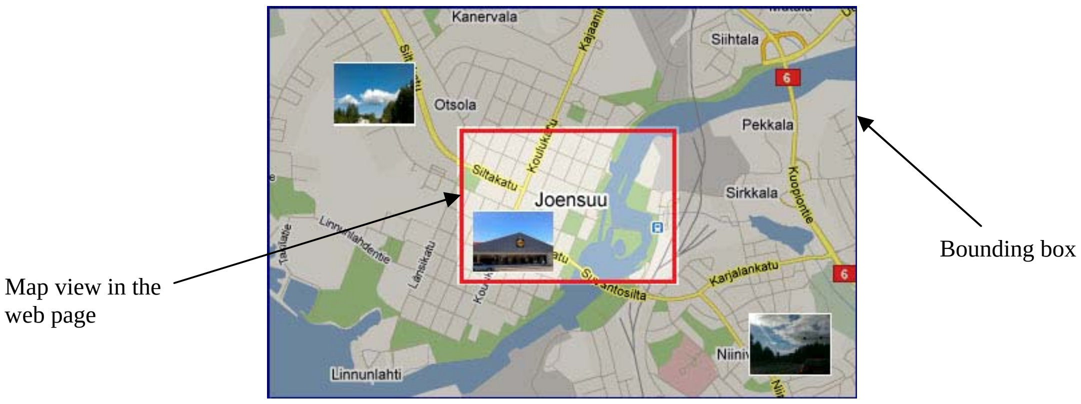


Figure 37. Current map view and bounding box in the photo collection page.

Since user needs visualization only on the current map area, the bounding box is set to big enough to cover the map area that user is viewing (see Figure 37). The bounding box will be updated when the user map view moves out of the current bounding box due to map panning or zooming. As a consequence, the clustering will be applied again corresponding to the new bounding box.

For clustering, we use a grid-based clustering method that works much more efficiently with time complexity O(*N*). We present the method as follows.

Grid cells construction.
A set of grid cell is defined according to the size (in pixels) of the data object such as photo in the given map scale. The bounding box is set to 9 times of the size of current map view in MOPSI applications (see Figure 38). However, when the map view is in a very lower scale (scale 0, scale 1 and scale 2), and the bounding box (9 times) will

be too big since the current map view can view the whole world. So the bounding box can be set as big as the current map view if the map scale is less than 3; otherwise it is set to 9 times of the size of the current map view.

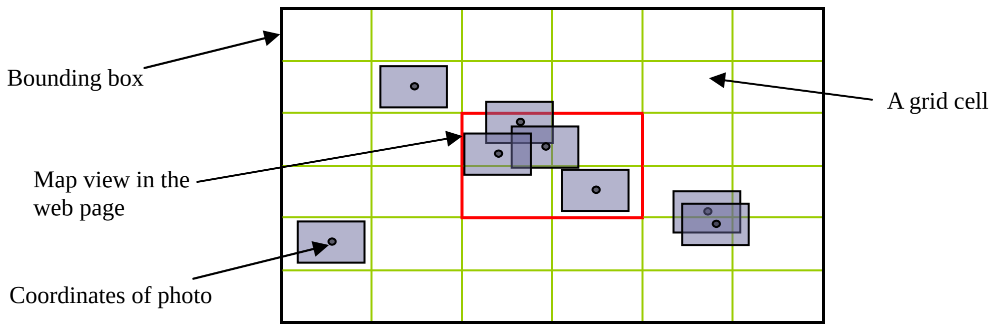


Figure 38. Grid cells and photo data objects assignment.

The number of rows n_{row} and columns n_{column} of the grid cells in scale s are calculated as:

$$
\begin{aligned}
w_{cell} &= \frac{W_0}{2^s} \quad \forall s \in [0, 21] \\
h_{cell} &= \frac{H_0}{2^s} \quad \forall s \in [0, 21] \\
n_{row} &= \left\lceil \frac{H_b}{h_{cell}} \right\rceil \\
n_{column} &= \left\lceil \frac{W_b}{w_{cell}} \right\rceil
\end{aligned}
\qquad 5.3.2
$$

where w_{cell} is the width of the grid cell and h_{cell} is height in scale s. W_0 and H_0 are the width and height of a data object in distance converted from the pixels coordinates presented on the map in scale 0. In MOPSI photo collection web application, the photo thumbnails (64*px* × 49*px* in pixels) are visualized on the map. Observed from the Google maps in scale 0, we set the W_0=6000km and H_0=4000km, respectively. W_b and H_b are the width and height of the bounding box in distance (km).

Data objects assignment.

Those data objects, which are inside of the bounding box, are assigned to the appreciate grid cell according to their coordinates and the coordinates of bounding box (see Figure 38). The row and column number of the grid cell for a given data object with coordinates (*lat* and *lon*) are calculated as:

$$row = \left\lfloor \frac{lat_{\max} - lat}{lat_{\max} - lat_{\min}} \cdot n_{row} \right\rfloor$$
$$column = \left\lfloor \frac{lon_{\max} - lon}{lon_{\max} - lon_{\min}} \cdot n_{column} \right\rfloor \qquad 5.3.3$$

where lat_{max} and lat_{min} are the maximum and minimum latitude of the bounding box, and lon_{max}, lon_{min} for longitude respectively. Each grid cell with valid data objects presents one initial cluster.

Result fine tuning.

Fine-tuning of the clustering result is done by merging neighbor clusters if their distance $d_{neighbor}$ is closer than a given threshold d_{merge}:

$$d_{neighbor} < d_{merge} \qquad 5.3.4$$

In MOPSI photo collection, we set $d_{merge} = 0.5 \times w_{cell}$. The merging is done by searching all clusters and their neighbors. Demonstrated in Figure 39, each cluster is presented by one photo (centroid photo) and has 8 neighbors at most. The current cluster (blue cell) and its *neighbor cluster A* can be merged.

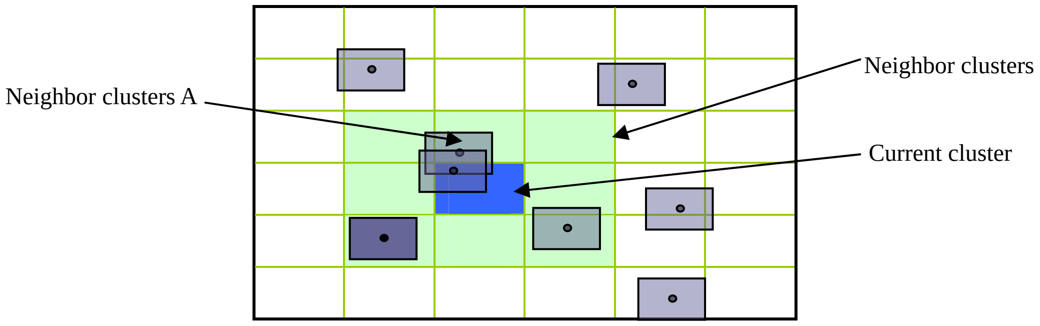


Figure 39. Cluster merging by checking the distance of cluster neighbors

In the algorithm, the grid cells constructions takes O(1) time, and O(N) for the data objects assignment. For the fine-tuning, it takes O(8·M) = O(M) time. So the total time complexity is O(N).

We next apply the above grid-based clustering to the photo collections web page in MOPSI. There are 3672 photo locations currently. The clustering results in different map scales are shown in Figure 40.

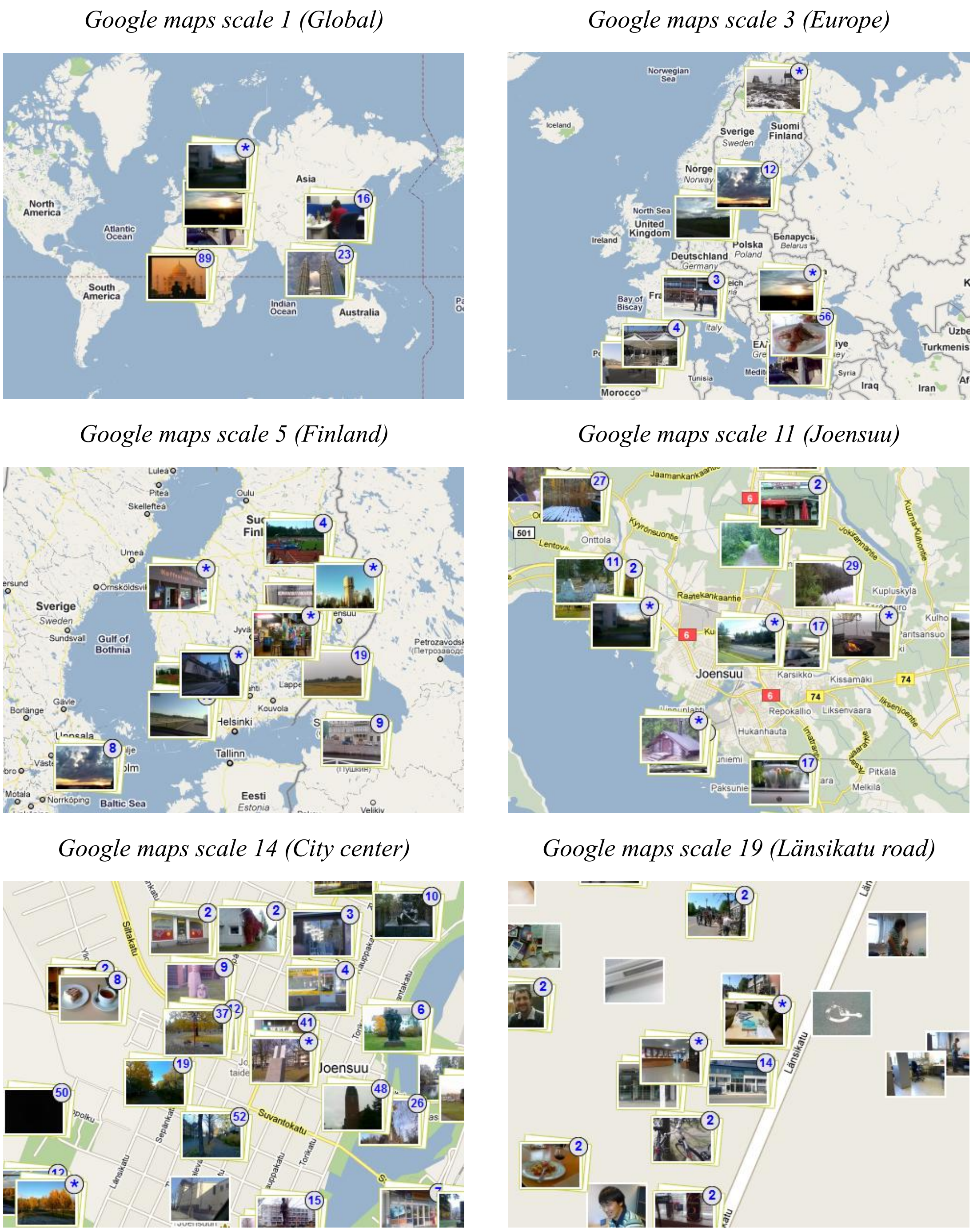

Google maps scale 1 (Global)
Google maps scale 3 (Europe)
Google maps scale 5 (Finland)
Google maps scale 11 (Joensuu)
Google maps scale 14 (City center)
Google maps scale 19 (Länsikatu road)

Figure 40. Clustering for photo collections in different map scales.

The processing time for clustering in different map scales is reported in Table 19. We can see that the clustering takes about 30 milliseconds for each scale, which is fast enough for real-time applications. Especially in higher scales such as 19, 20 and 21, only 10 milliseconds are used because the amount of photos in the bounding box is smaller in high scales.

Table 19. Processing time T (ms) of grid-based clustering in each scale

Scale	0	1	2	3	4	5	6	7	8	9	10
T	26	25	25	27	26	31	25	27	27	26	21
Scale	11	12	13	14	15	16	17	18	19	20	21
T	22	23	26	28	26	24	23	18	13	12	10

The clustering method will be run again when the map view moves out of the bounding box due to panning of the map. Nevertheless, it works smoothly in the web application since the processing time of the method is in milliseconds for each run when zooming or panning the map.

6 Conclusions

In this thesis, we have investigated swap-based clustering methods such as random swap, deterministic swap and hybrid swap. 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 burden in the overall processing time. To reduce the total time complexity, a fast variant of k-means is employed to repartition the dataset and fine-tune the swapped solution.

We have compared the clustering quality and efficiency of the swap-based clustering methods on both synthetic datasets and real datasets. According to the experimental results, from the clustering quality point of view, the random swap always achieves the best clustering quality or even the optimal result in case the number of iterations is set to high enough. However, from the clustering efficiency point of view, smart swap is more efficient for most datasets. It is therefore a good choice to select random swap when best clustering is needed but to use smart swap when efficiency is more important.

In addition, we have applied the clustering in a location-based application called MOPSI project to reduce the clutter problem in map visualization in different scales, using a split smart swap clustering method to cluster the user locations and using a grid-based clustering with bounding box method to cluster the photo collections. The evaluation results show that the split smart swap works in the MOPSI user tracking application in real-time. However, it is slow for the larger photo collection database, which is a challenge for real-time applications. It can be improved further considering running the clustering method on the web server side to reduce the processing time. Meanwhile, the grid-based clustering with bounding box method was implemented. It works with good clustering result and significant fast speed.

The main contribution of this thesis is that we have proposed a more efficient swap - smart swap and discussed the swap-based clustering methods from both clustering quality and clustering efficiency points of view, which can be considered as the important reference to the selection of clustering methods for the user. Moreover, we have applied the clustering to map visualization of user tracking and photo collections in a location-based application MOPSI.

References

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

[2] R. Xu, II Donald Wunsch, “Survey of clustering algorithms,” IEEE Trans on Neural Networks, vol. 16 (3), pp. 645-678, 2005.

[3] McQueen JB, “Some methods of classification and analysis of multivariate observations,” Proc 5th Berkeley Symposium Matchmatical Statistical Probability, vol. 1, pp. 281-297, 1967.

[4] Stuart P. Lloyd, “Least squares quantization in PCM”, IEEE Transactions on Information Theory, vol. 28 (2), pp. 129-137, 1982.

[5] Wikipedia, “k-means clustering”, http://en.wikipedia.org/wiki/K-means_clustering, accessed 15.7.2010.

[6] J.H. Ward, “Hierarchical grouping to optimize an objective function,” Journal of Amer. Statist.Assoc. 58, pp. 236-244, 1963.

[7] Hierarchical clusteirng, http://nlp.stanford.edu/IR-book/html/htmledition/hierarchical-clustering-1.html, accessed 28.9.2010.

[8] T. Zhang, R. Ramakrishnan, and M. Livny, “BIRCH: an efficient data clustering method for very large databases,” ACM SIGMOD Record, vol 25 (2), pp. 103-114, 1996.

[9] S. Guha, R. Rastogi, and K. Shim, “Cure: An efficient clustering algorithm for large databases,” ACM SIGMOD, pp. 73-84, 1998.

[10] G. Karypis, E.H. Han, and V. Kumar, “Chameleon: A hierarchical clustering algorithm using dynamic modeling,” IEEE Computer, vol. 32 (8), pp. 68–75, 1999.

[11] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, “A density-based algorithm for discovering clusters in large spatial databases with noise,” Int. Conf. on Knowledge Discovery and Data Mining, pp. 226-231, 1996.

[12] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, Jörg Sander, “OPTICS: Ordering Points To Identify the Clustering Structure,” ACM SIGMOD international conference on Management of data, pp. 49-60, 1999.

[13] Peter J. Rousseeuw, “Silhouettes: a Graphical Aid to the Interpretation and Validation of Cluster Analysis,” Computational and Applied Mathematics vol. 20, pp. 53-65, 1987.

[14] M. Halkidi, Y. Batistakis and M. Vazirgiannis, “Clustering Validity Methods: Part I,” ACM SIGMOD Record, vol. 31 (2), pp. 40-45, 2002.

[15] M. Halkidi, Y. Batistakis and M. Vazirgiannis, “Clustering Validity Methods: Part II,” ACM SIGMOD Record, vol. 31 (3), pp. 19-27, 2002.

[16] 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, 2001.

[17] 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.

[18] P. Fränti and O. Virmajoki, “Iterative shrinking method for clustering problems,” Pattern Recognition, vol. 39 (5), pp. 761-765, 2006.

[19] A. Likas, N. Vlassis and J.J. Verbeek, “The global k-means clustering algorithm,” Pattern Recognition, vol. 36, pp. 451-461, 2003

[20] T. Kaukoranta, P. Fränti and O. Nevalainen, “Iterative split-and-merge algorithm for VQ codebook generation,” Optical Engineering, vol. 37 (10), pp. 2726-2732, 1998.

[21] H. Frigui and R. Krishnapuram, “Clustering by competitive agglomeration,” Pattern Recognition, vol. 30 (7), pp. 1109-1119, 1997.

[22] P. Fränti, T. Kaukoranta and O. Nevalainen, “On the splitting method for vector quantization codebook generation,” Optical Engineering, vol. 36 (11), pp. 3043-3051, 1997.

[23] 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.

[24] 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.

[25] J. Chen, Q. Zhao, and P. Fränti, “Smart swap for more efficient clustering”, Int. Conf. on Green Circuits and Systems (ICGCS’10), Shanghai, China, pp. 446-450, Jun. 2010.

[26] 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.

[27] P. Fränti, O. Virmajoki and V. Hautamäki, “Probabilistic clustering by random swap algorithm,” IAPR Int. Conf. on Pattern Recognition (ICPR’08). Tampa, FL, USA, 2008.

[28] Zhang et al., “BIRCH: A new data clustering algorithm and its applications,” Data Mining and Knowledge Discovery, vol. 1 (2), pp. 141-182, 1997.

[29] C.L. Blake, C.J. Merz, UCI repository of machine learning databases, University of California, Irvine, Department of Information and Computer Sciences, 1998, http://archive.ics.uci.edu/ml/datasets/Iris.

[30] Stefan Steiniger, Moritz Neun and Alistair Edwardes, “Foundations of Location Based Services”, University of Zurich.

[31] K. Virrantaus, J. Markkula, A. Garmash, Y. V. Terziyan, “Developing GIS-Supported Location-Based Services,” First International Workshop on Web Geographical Information Systems (WGIS’2001), Kyoto, Japan, pp. 423-432, 2001.

[32] Wikipedia, “Location-based services”, http://en.wikipedia.org/wiki/Location-based_service, accessed 11.8.2010.

[33] Shu Wang, Jungwon Min and Byung K. Yi, “Location Based Services for Mobiles: Technologies and Standards,” IEEE International Conference on Communication (ICC), Beijing, China, 2008.

[34] “Mobile Local Search Saturates Profit over LBS Vendors, Advertisers, and Search Application Developers”,

http://www.directionsmag.com/pressreleases/mobile-local-search-saturates-profit-over-lbs-vendors-advertisers-and-searc/120497, accessed 12.8.2010.

[35] P. Fränti, J. Kuittinen, A. Tabarcea, L. Sakala, “MOPSI location-based search engine: concept, architecture and prototype,” ACM Symposium on Applied Computing (SAC'10), Sierre, Switzerland, pp.872-873, Mar. 2010.

[36] G. Hariharan, P. Fränti, and S. Mehta, “Data Mining for Personal Navigation,” SPIE Conf. on Data Mining and Knowledge Discovery: Theory, Tools, and Technology IV, Orlando, Florida, Vol. 4730, pp.355-365, Apr. 2002.

[37] P. Fränti, A. Tabarcea, J. Kuittinen, V. Hautamäki, “Location-based search engine for multimedia phones,” IEEE Int. Conf. on Multimedia & Expo (ICME’10), Singapore, pp.558-563, Jul. 2010.

[38] A. Tabarcea, V. Hautamäki, P. Fränti, “Ad-hoc georeferencing of web-pages using street-name prefix trees,” Int. Conf. on Web Information Systems & Technologies (WEBIST’10), Valencia, Spain, vol. 1, pp. 237-244, Apr. 2010.

[39] Jean-Yves Delort, “Vizualizing Large Spatial Datasets in Interactive Maps,” Second International Conference on Advanced Geographic Information Systems, Applications, and Services, pp.33-38, 2010.

[40] N. Elmqvist and J.-D. Fekete, “Hierarchical aggregation for information visualization: Overview, techniques and design guidelines,” IEEE Transactions on Visualization and Computer Graphics, vol. 16 (1), pp. 439-454, 2010.

[41] A Tutorial on Clustering Algorithms, http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/, accessed 1.8.2010.

[42] G. Ball and D. Hall, “A clustering technique for summarizing multivariatedata,” Behav. Sci., vol. 12, pp. 153-155, 1967.

[43] L. Kaufman and P. Rousseeuw, “Finding Groups in Data: An Introduction to Cluster Analysis,” John Wiley & Sons, Newyork, 1990.