2022 IEEE/ACM 26th International Symposium on Distributed Simulation and Real Time Applications (DS-RT) | 978-1-6654-9799-2/22/$31.00 ©2022 IEEE | DOI: 10.1109/DS-RT55542.2022.9932090

Efficient and Reliable Clustering by Parallel Random Swap Algorithm

Libero Nigro[1], Franco Cicirelli[2], Pasi Fränti[3]
[1]DIMES - Department of Informatics Modelling Electronics and Systems Science
University of Calabria, 87036 Rende, Italy
Email: l.nigro@unical.it
[2]CNR - National Research Council of Italy - Institute for High Performance Computing and Networking (ICAR) - 87036 Rende, Italy
Email: f.cicirelli@icar.cnr.it
[3]School of Computing, Machine Learning Group
University of Eastern Finland
P.O.Box 111, 80101 Joensuu, Finland
Email: franti@cs.uef.fi

***Abstract*—Solving large-scale clustering problems requires an efficient algorithm which can be implemented also in parallel. K-means would be suitable but it can lead to an inaccurate clustering result. To overcome this problem, we present a parallel version of random swap clustering algorithm. It combines the scalability of k-means with high clustering accuracy. The new clustering method is experimented on top of Java parallel streams and lambda expressions, which offer interesting execution time benefits. The method is applied to standard benchmark datasets, with a varying population size and distribution of managed records, dimensionality of data points and the number of clusters. The experimental results confirm that high quality clustering can be obtained by parallel random swap together with a high time efficiency.**

Keywords—Clustering problem, K-Means, Random swap, Parallelism, Streams, Lambda Expressions, Java.

I. INTRODUCTION

The clustering problem occurs in many application areas such as physics, bioinformatics, image segmentation, machine learning, medicine, and artificial intelligence. It can be stated as follows. There are N data points $\{x_i\}$ (records or data vectors), here assumed to have numerical attributes, that is $x_i \in R^D$, which are to be partitioned into $K \ll N$ clusters, in a such a way that points within a same cluster are similar and points in different clusters are dissimilar. Each cluster is represented by its central point (centroid or prototype). Finding the optimal solution to the problem is NP-hard, and only relatively small problem sizes can be solved optimally [1]. Heuristic algorithms are therefore necessary to generate sub-optimal solutions.

K-means is well-known clustering algorithm which partitions data points according to Euclidean nearest centroid by minimizing the *Sum of Squared Error* (SSE) objective function (a measure of internal variance in clusters). Although more sophisticated clustering algorithms have been defined [2-5], K-means is often used due to its simplicity and efficiency. K-means behaviour, though, strongly depends on the centroids initialization method [6-8] and will get stuck to a local sub-optimal solution. K random points of the dataset are often chosen as initial centroids.

To overcome the limitations of K-means, random swap technique was proposed in [4] together with a formal characterization of its properties. Most operations of random swap can be executed in parallel.

The original contribution of this paper is to propose and develop in Java[1] [9] Parallel Random Swap as a clustering method, and to provide experimental evidence of its efficiency from both the execution performance and clustering quality.

II. RANDOM SWAP

Random swap algorithm [4] was designed to solve the cluster structure by a sequence of centroid swaps (global search), and by fine-tuning the result by K-means (local search). The algorithm significantly improves K-means because it almost never gets stuck in a sub-optimal local solution. The results in [6] showed that it reaches the correct global allocation of the clusters with all benchmark datasets.

K-means operation

Let $\{C_1, C_2, \dots, C_K\}$ be the K clusters, and $\{c_1, c_2, \dots, c_K\}$ the corresponding representative centroid points. The SSE is defined as:

$$SSE = \sum_{i=1}^{N} \|x_i - nc(x_i)\|^2$$

where $nc(x_i)$ is the nearest centroid c_j to x_i according to Euclidean distance, that is:

$$nc(x_i) = c_j, \text{where } j \text{ is: } \underset{1 \le j \le K}{arg\,min} \left\|x_i - c_j\right\|^2$$

[1] https://github.com/uef-machine-learning

978-1-6654-9799-2/22/$31.00 ©2022 IEEE

It is often preferable to use the normalized mean SSE, indicated as $nMSE$, defined as: $nMSE = SSE/(N * D)$. Starting from an initialization of centroids, K-means iterates the two steps shown in Fig. 1 a maximum number of iterations or until convergence is sensed (the new centroids equals to the previous ones).

1. *Partition* data points $\{x_i\}$ into clusters according to $nc(x_i)$;
2. *Update* centroids as the mean point of each cluster:

$$c_j = \frac{1}{|C_j|}\sum_{x_i \in C_j} x_i, \forall j \text{ in } [1, K]$$

Fig. 1. The two steps of a K-means iteration.

Random swap algorithm

A main point of the algorithm in Fig. 2 is the *Swap* step 3, where a new globally defined centroid configuration is established. The new centroids are then locally fine-tuned, and corresponding cluster boundaries redefined by K-means at step 4. Steps from 3 to 5 are repeated for a fixed number of iterations constituting the most processing time.

1. *Centroids initialization*. Define initial centroids as K randomly selected points in the dataset.
2. *Initial partition*. Partition data points according to the initial centroids.

Repeat T times:

3. *Swap*. A centroid is randomly selected, and replaced by a randomly chosen data point in the data space:
   $c_s \leftarrow x_i, s = rand(1, K), i = rand(1, N)$
4. *K-means*. A few iterations of K-means (e.g., 2) are executed (partition and centroids update).
5. *Test*. Check if the new solution (new centroids and associated partition) has a lower $nMSE$ cost than the solution from the previous step. If it is, it is accepted and made as the current. Otherwise, previous centroids and corresponding data partition are restored.

6. *Final tuning*. The last defined solution is improved by a final execution of K-means, which is iterated until convergence or after a maximum number of iterations were executed.

Fig. 2. Steps of the random swap algorithm.

Evaluation of clustering quality

Random swap iterations are controlled by the values of the $nMSE$ function cost, which monotonically decreases as the swap iterations proceed. This quantity guides the search. The clustering quality can also be evaluated by an internal index or by external quality indexes if ground truth (GT) centroids are known.

Centroid Index (CI). It measures how well the centroid locations match to that of the ground truth centroids [10]. It can be used with synthetic datasets constructed by some specific distribution of points around pre-defined centroids (prototypes). The CI value of a clustering solution C can be understood intuitively by counting in C the number of real clusters not having any centroid and those which have more than one centroid. The CI value is the greater of these numbers. CI can be computed by mapping C onto GT and vice versa and counting dissimilarities. In the case the bijection evaluates to 0 (each element of C maps exactly on one point of GT by minimal Euclidean distance and vice versa, that is there are "no orphan" in the two directions), the obtained solution has high probability to be correct.

Silhouette Index (SI). It is a classic measure of clustering accuracy [8,11]. First for each point x two quantities are computed: a_x and b_x. The a_x component is an intra-cluster measure, that is the average distance of x from the other points of the same cluster. The b_x expresses the minimum average distance of x from all other points in different clusters. The Silhouette value associated to the point x is calculated as:

$$S_x = \frac{b_x - a_x}{\max(b_x, a_x)}$$

Finally, the SI of the clustering solution is given by:

$$S = \frac{1}{N}\sum_{i=1}^{N} S_{x_i}.$$

The SI value ranges in [-1,1]. A value 1 indicates well separated clusters (reduced overlap). A value close to 0 mirrors high overlap of clusters. A value toward -1 indicates incorrect clustering.

III. ENABLING PARALLEL RANDOM SWAP IN JAVA

The initial partition (step 2 of Fig. 2), the partition and update phases of K-means (steps 4 and 6) and the restore partition at step 5, can purposely be carried in parallel. Further operations which benefit of parallelism include the computation of clustering indexes like the Silhouette (SI) with cost $O(N^2)$.

The dataset and centroids were mapped onto native arrays of DataPoint objects, from which data streams are derived. DataPoint exposes methods for point arithmetic (e.g. addition and mean), distance calculation and so forth. A data point also holds a field cid, i.e. the index of the centroid it was partitioned to.

Fig. 3 shows the partition operation (see also [12]) which, depending on the value of the PARALLEL parameter, can be executed in parallel or sequentially. The map operation receives a lambda expression (Function) whose parameter p is a point, whose cid is set to the index of the nearest centroid. Each individual point only modifies itself. No shared data are modified.

```
Stream<DataPoint> p_stream=Stream.of( dataset );
if( PARALLEL ) p_stream=p_stream.parallel();
p_stream
  .map( p -> {
    double md=Double.MAX_VALUE;
    for( int k=0; k<K; ++k ) {
      double d=p.distance( centroids[k] );
      if( d<md ) { md=d; p.setCID(k); }
    }
    return p; } )
  .forEach( p->{} );
```

Fig. 3. The partition operation.

The partition operation also occurs in the kmeans(…) method which can be iterated a fixed number of times or until convergence. Fig. 4 depicts an excerpt of the parallel update centroids step of kmeans(…).

```
//update centroids
Stream<DataPoint> c_stream=Stream.of( centroids );
if( PARALLEL ) c_stream=c_stream.parallel();
c_stream
  .map( c -> {
    for( int i=0; i<N; ++i ) {
      if( dataset[i].getCID()==c.getCID() ) c.add(dataset[i]) ;
    }
    c.mean();
    return c; } )
  .forEach( c->{} );
```

Fig. 4. An excerpt of the update centroids step of K-means.

IV. EXPERIMENTAL SETUP

Correctness of parallel random swap, that is clustering quality and time efficiency, were checked by applying it to 12 basic benchmark datasets [6,13]. For all the datasets the ground truth (GT) centroids are publicly available. The datasets (see Table 1) characterize for the number and shape of clusters and point distributions used to construct the dataset.

Table 1. Parameters of benchmark datasets [13]

dataset	N	D	K
A1,A2,A3	3000,5250,7500	2	20,35,50
S1,S2,S3,S4	5000	2	15
G2-1024-100	2048	1024	2
DIM32	1024	32	16
UNBALANCE	6500	2	8
BIRCH1,BIRCH2	100000	2	100

The A sets contain spherical clusters and are organized as subsets of each other: $A_1 \subset A_2 \subset A_3$. The S sets contain Gaussian clusters with varying degree of overlap. The G2 sets contain 1024 points distributed according to 2 Gaussian clusters at fixed locations. Overlap is created by varying the standard deviation from 10 to 100. A particular dataset is named G2-dim-std accordingly. The G2-1024-100 dataset was selected for the experiments. The DIM sets contain well-separated clusters in high-dimensional space. For the experiments DIM32 was chosen: 32 dimensions for each of 1024 points. Points are randomly distributed among clusters by Gaussian distribution. The Unbalance dataset has eight clusters organized in two well-separated groups. The first three clusters are dense with 2000 points each. The remaining five clusters contain 100 points each. The two Birch sets contain spherical clusters organized on a 10x10 grid (Birch1), and to a sine curve (Birch2).

V. RESULTS

Parallel random swap was applied to the 12 benchmark datasets described in Section IV. Although only 2 iterations of K-means per swap were suggested in [4] to refine a local solution at each swap iteration, we increased this to 5 motivated by the parallel support. In addition, not originally used in [4], we also refine the final solution after the swap iterations by executing K-means exhaustively until convergence. The maximum number of swap iterations and the maximum number of steps of the final invocation of K-means were fixed, in all the executions, to $T = 5000$.

All the execution experiments were carried on a Win10 Pro, Dell XPS 8940, Intel i7-10700 (8 physical cores which with the hyperthreading give support to 16 threads), CPU@2.90 GHz, 32GB Ram, Java 17.

Testing

Preliminary runs were devoted to assessing the correctness of the developed random swap program, in particular that the program delivers exactly the same result in sequential and in parallel mode on all the benchmark datasets.

Fig. 5 shows an extract of the output generated by 1 run of Birch1. Steps 2 and 3 are examples of successful steps. Unsuccessful steps (nMSE did not improve) are omitted in the listing. Sometimes nMSE improved but CI did not. Such changes are also accepted although not considered successful steps like those that failed to improve nMSE. The results confirm the monotonic decrease of nMSE. Starting from step 237, CI stabilizes to value 0. From 4347 to 5000 all swaps are unproductive as nMSE did not improve further.

Step	nMSE	CI
1	6.622308508119364E8	13
2	6.05899752096259E8	9
3	5.74568446939268E8	8
5	5.73673847218781E8	8
…		
14	5.629913152219414E8	7
…		
19	5.385070562725195E8	6
20	5.3324325081301624E8	5
27	5.213528313807281E8	4
…		
32	5.052130404197854E8	3
…		
100	4.815435483122903E8	1
…		
237	4.7379832580636716E8	0
255	4.664072159088196E8	0
265	4.6386566539892936E8	0
330	4.6386462201360106E8	0
706	4.638641013410574E8	0
1968	4.638640964116055E8	0
2071	4.638640668751212E8	0
2559	4.638640333827571E8	0
4346	4.638640290995247E8	0
R 5001	4.638640290995247E8	0

Fig. 5. Debugging example of Birch1.

Computational efficiency

To check the computational efficiency, five runs of the Java-based random swap program were executed on the Birch1 dataset (see Table 1), both in sequential and parallel modes.

The overall wall-clock time required for completing the T=5000 swap iterations (see Fig. 2) was measured. The two times are referred to as RS Sequential Elapsed Time (RS-SET) and RS Parallel Elapsed Time (RS-PET). After that, the Sequential Average Swap Time (SAST) and the Parallel

Average Swap Time (PAST) were estimated. The speedup was then evaluated as the ratio of SAST/PAST.

In a similar way, the SI-SET and SI-PET were measured by computing the Silhouette index (SI) both in sequential and parallel mode. The average execution times, namely aSI-SET and aSI-PET, were then calculated, and the corresponding speedup evaluated as the ratio aSI-SET/aSI-PET. The recorded execution times are summarized in Table 2.

Table 2. Run times of Random Swap and Silhouette Index (P=16 threads)

#run	RS-SET (ms)	RS-PET (ms)	SI-SET (ms)	SI-PET (ms)
1	1224352	132090	42788	4600
2	992717	142649	43239	4733
3	996234	144315	43238	4942
4	1000580	151051	42916	5186
5	986371	157491	42824	5382

From Table 2, we derive an average value of SAST=208.1 ms per swap iteration, and PAST=29.10 ms. This results in a speedup of 7.15.

The average times for SI emerged to be: aSI-SET=43001 ms and aSI-PET=4968 ms, with a speedup of 8.65.

Clustering accuracy

The quality of clustering solutions generated by parallel random swap was estimated by 10 runs, each of $T = 5000$ swap iterations, executed in parallel mode, for each dataset in Table 1. A clustering solution consists of final centroids and the index (label) of the clusters in which the data points were assigned to. The quality of the solution is captured by the values of nMSE cost, Silhouette index (SI) and Centroid index (CI) calculated between the proposed solution and the ground truth centroids.

Table 3 contains the recorded results for all the 12 selected benchmark datasets. As a notable property, parallel random swap was capable, almost deterministically, to find the correct solution of the various datasets but with low runtime. For the Birch1 $(N = 100000, K = 100, T = 5000)$ the result is generated in about 2.5 min instead of the 17 min required by the sequential approach. The always obtained CI=0 confirms the same result predicted in [4].

Table 3. Clustering accuracy results.

dataset	nMSE	SI	CI
A1	2.02E6	0.595	0
A2	1.93E6	0.598	0
A3	1.93E6	0.601	0
S1	8.92E8	0.711	0
S2	1.33E9	0.626	0
S3	1.69E9	0.493	0
S4	1.57E9	0.480	0
G2-1024-100	9.98E3	0.183	0
DIM32	7.096	0.946	0
UNBALANCE	1.65E7	0.858	0
BIRCH1	4.64E8	0.460	0
BIRCH2	2.28E6	0.736	0

VI.CONCLUSIONS AND FUTURE WORK

This paper proposes Parallel Random Swap [4], a novel method which provides efficient and reliable clustering on nowadays multi-core machines. The new method leverages on the lock-free concurrency support which Java supplies when dealing with parallel streams of non-trivial datasets. The tool was applied to a set of benchmark datasets thus confirming careful clustering solutions can be achieved with significant time efficiency.

Prosecution of the work will address the following points.

First, to identify possible heuristics for early termination of method. Preliminary experiments seem to suggest that when a certain number of consecutive accepted iterations occur with a 0 value of the *local* Centroid index (that is, evaluated between the new centroids configuration and the one existing at the beginning of the swap iteration), followed by a few hundred unproductive iterations, the last found solution, with great success frequency, is the correct one. More investigation, though, is deemed necessary using both synthetic and real-world datasets.

Second, to port Parallel Random Swap on the efficient Theatre actor-system with message-passing [14-15], which enables a better exploitation of the computing resources through a custom split of the dataset into blocks to be managed in parallel by a specific number of theatres/threads.

REFERENCES

[1] P. Fränti, O. Virmajoki. Optimal clustering by merge-based branch-and-bound. Applied Computing and Intelligence, 2(1):63–82, 2022.

[2] P. Fränti, Genetic algorithm with deterministic crossover for vector quantization. Pattern Recognit Lett., 21(1):61–8, 2000.

[3] A. Likas, N. Vlassis, JJ. Verbeek. The global k-means clustering algorithm. Pattern Recognit., 36:451–61, 2000.

[4] P. Fränti. Efficiency of random swap algorithm. J. Big Data, 5(1), 1-29, 2018.

[5] A. Rodriguez, A. Laio. Clustering by fast search and find of density peaks. Science, 344(6191), 14.92–14.96, 2014.

[6] P. Fränti, S. Sieranoja. K-means properties on six clustering benchmark datasets. Applied Intelligence, 48(12), 4743-4759, 2018.

[7] P. Fränti, S. Sieranoja. How much can k-means be improved by using better initialization and repeats? Pattern Recognition, 93, 95-112, 2019.

[8] A. Vouros, S. Langdell, M. Croucher, E. Vasilaki. An empirical comparison between stochastic and deterministic centroid initialization for K-means variations. Machine Learning, 110, 1975–2003, 2021.

[9] R.G. Urma, M. Fusco, A. Mycroft. Modern Java in Action. Manning, Shelter Island, 2019.

[10] P. Fränti, M. Rezaei, Q. Zhao. Centroid index: cluster level similarity measure. Pattern Recognition, 47(9), 3034-3045, 2014.

[11] P.J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. Journal of Computational and Applied Mathematics, 20, 53-65, 1987.

[12] L. Nigro. Performance of parallel K-means algorithms in Java. Algorithms, 15(4), 117, 2022.

[13] Benchmark datasets, http://cs.uef.fi/sipu/datasets/.

[14] L. Nigro. Parallel Theatre: An actor framework in Java for high performance computing. Simulation Modelling Practice and Theory, 106, https://doi.org/10.1016/j.simpat.2020.102189, 2021.

[15] F. Cicirelli, L. Nigro. Parallel simulation of Stochastic Reward Nets using Theatre. 25th ACM/IEEE Int. Symp. on Distributed Simulation and Real-Time Applications (DS-RT 2021), IEEE Xplore, 2021.