Coevolutive clustering algorithm for large datasets

Fábio Fabris
Computer Science Department
Federal University of Espirito Santo
Vitoria, Brazil
fabiofabris@gmail.com

Diego Luchi
Computer Science Department
Federal University of Espirito Santo
Vitoria, Brazil
diego.lucchi@gmail.com

Flávio Miguel Varejão
Computer Science Department
Federal University of Espirito Santo
Vitoria, Brazil
fvarejao@inf.ufes.br

***Abstract*—Clustering is a recurrent task in machine learning. The application of traditional heuristics techniques in large sets of data is not easy. They tend to have at least quadratic complexity with respect to the number of points, yielding prohibitive run times or low quality solutions. The most common approach to tackle this problem is to use weaker, more randomized algorithms with lower complexities to solve the clustering problem. This work proposes a novel approach for performing this task, allowing traditional, stronger algorithms to work on a sample of the data, chosen in such a way that the overall clustering is considered good. Preliminary experimental results indicate that the proposed approach is competitive to classical algorithms in large datasets with the advantage of automatically adapting to many different datasets.**

***Index Terms*—clustering, co-evolution, large datasets**

I. Introduction

Data Clustering is the task of assigning data to groups (called clusters) so that similar objects belong to the same group given a similarity metric. Grouping similar objects is a recurrent problem when dealing with analysis of large datasets.

This work focuses on tackling the hard clustering problem in metric spaces using the *Sum of Squared Errors* (*SSE*) as the error function, mostly known as the k-means problem [1]. There are many algorithms proposed in the literature to tackle this problem. They are divided in *exact* [2] and are *NP-Hard*; *approximate* guarantee an $(1-\epsilon)$ approximation where ϵ is coupled to the complexity of the underlying algorithm [3]; and *heuristic* algorithms [4].

Given these facts, the most common way to solve the k-means problem on medium-sized datasets is by using heuristic algorithms. This kind of algorithm guarantees neither optimal nor approximate solutions, but usually has polynomial complexity.

The use of co-evolutionary algorithms for solving optimization problems is ubiquitous [5]–[8]. The co-evolutionary approach is based on the natural behaviour of species in nature and employs the use of two or more populations competing or collaborating towards a common goal. This work focuses on the application of a novel co-evolutionary algorithm to the clustering problem and performs a comparison with classical algorithms.

The remainder of this paper is organized as follows: Section II contains the formal definition of the clustering problem and a revision of related work regarding clustering of large datasets. Section III describes the novel approach. Section IV exposes the set up of the experiments, special considerations and results and finally, in Section V the conclusions are drawn.

II. Related Work

In this work, clustering is the restricted discret optimization problem of dividing n spatial points $\bar{x_i} \in \Re^d, i \in \{1,\ldots,n\}$ of a set X, in k complete and disjoint sets (clusters) $C_i, i \in \{1,\ldots,k\}$. This division must minimize the *Sum of the Squared Errors (SSE)* of all points with respect to the *cluster centroid*. The centroid of a cluster i, $\bar{c_i}$ is the mean of all points of cluster i.

More formally, if $X = \{\bar{x}_1, \bar{x}_2, \bar{x}_3, \ldots, \bar{x}_n\}$ is a set of points, $X \subset \Re^d$, then the clustering result must adhere to the following restrictions:

$$\bigcup_{i=1}^{i\leq k} C_i \equiv X \tag{1}$$

$$\bigcap_{i=1}^{i\leq k} C_i \equiv \emptyset \tag{2}$$

And minimize the following expression:

$$SSE(c) = \sum_{i=1}^{i\leq k} \sum_{j=1}^{j\leq S_i} (||C_{i,j} - \bar{c_i}||_2) \tag{3}$$

Where $C_{i,j}$ is the j-th point of the i-th cluster and S_i is the size of cluster j. In addition, $||\bar{t}||_2$ represents the euclidean norm of the vector $\bar{t} \in \Re^d$.

This work also focuses on algorithms for large datasets. This kind of algorithm must be efficient, i.e., avoid the need of scanning the whole data set multiple times. Complexities greater than quadratic time on the size of the input must be avoided. Exponential complexities are almost always prohibitive.

A. Classical algorithms for solving the k-means problem in large datasets

The most famous algorithm for dealing with the k-means problem is called the *Loyd Algorithm* [1] (or simply the *k-means algorithm*, due to its ubiquity). The k-means algorithm is widely used in the literature with very good results, however, has particularities that restrict its use for large datasets. Although fast in small datasets, the algorithm does not scale well when the number of points increase. With large datasets the use of this algorithm is unfeasible due to its complexity

978-1-7281-6929-3/20/$31.00 ©2020 IEEE

$O(nkdi)$ in most cases – where n is the number of d-dimensional points, k the number of clusters and i the number of iterations needed until convergence – and super-polynomial in the worst case [9], [10].

The first attempts to elucidate the problem of clustering large datasets was solving the k-medoids problem. The k-medoids problem sets the centroids of the clusters to actual points of the data set, not arbitrary ones, like the k-means algorithm. One of the first k-medoids clustering algorithms proposed for large data sets was the *CLARANS* (Clustering Large Applications Based on Randomized Search) algorithm [11]. This algorithm was inspired by the *PAM* (Partitioning Around Medoids) and *CLARA* (Clustering Large Applications) algorithms [12]. It performs a very simple random search in a graph, randomly selecting neighbors of a given solution and checking if the new solution is better than the old one, with the possibility of accepting few worst solutions before falling back to the previous best solution found. This simple idea turned out to be both fast and effective, and it is one of the major algorithms for solving the problem of clustering large datasets until today, being widely used as a validation algorithm in benchmarks of the literature [13].

BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) [14] is an algorithm used to perform hierarchical clustering over large datasets. The advantage of *BIRCH* is its ability to incrementally and dynamically cluster incoming, multi-dimensional metric data points in an attempt to produce the best quality clustering for a given set of resources (memory and time constraints). In most cases, *BIRCH* only requires a single scan of the database.

CURE [15] (Clustering Using REpresentatives) is an efficient data clustering algorithm for large datasets. *CURE* represents each cluster by a fixed number of points generated by selecting well scattered points from the cluster and then shrinking them toward the center of the cluster by a specified fraction. To handle large datasets, *CURE* employs a combination of random sampling and partitioning.

DBSCAN [16] is also an algorithm for clustering large spatial datasets, even though it is not intended for solving the k-means problem. *DBSCAN* does not try to optimize the SSE, but is included here for being a traditional clustering algorithm for large datasets. The idea of *DBSCAN* is that isolated, high density regions, must belong to the same cluster. This implies that it is not possible to choose the number of clusters directly. This may be useful in applications where the number of clusters is unknown but it is not advantageous if the number k of cluster is known *a priori*.

B. Co-evolution algorithms and clustering

This work uses a co-evolutive algorithm for solving the clustering problem. This kind of algorithm is inspired in the field of game theory, where two agents compete until they find the *Nash Equilibrium* of the game.

The use of co-evolutive algorithms is common when one needs to optimize two distinct sets of parameters with a coupled fitness function. The sets of parameters may be competing, i.e., while one set tries to minimize the objective function, the other tries to maximize it (competitive co-evolution); or cooperating, i.e., both populations try to optimize the same objective function. Much work has been published with this approach, in particular in function optimization with restrictions, e.g. [17], [18]. The most common approach is to encode the decision variables of the problem as individuals of one population, that tries to optimize the objective function, and the other population as the lagrangian multipliers of the restrictions, encoded in the objective function. Many works use this approach to solve complex design problems with restrictions [17], [18].

This idea coupled with *Genetic Algorithms* for solving many restricted optimization benchmark functions with great success [8]. Also, another work developed a competitive co-evolution algorithm that iteratively selects the most difficult instances to classify using supervised algorithms and the most efficient algorithms, each taken from specific populations [7].

Since co-evolution is designed for solving the generic optimization problem with dual objective functions, this work applies an idea based on the work of [7], creating a population for choosing "hard" points for the clustering process and another population of classical algorithms for solving the clustering process. The goal is to find the set of points that are most relevant for the final clustering process and the best possible solution for this set of points.

There are some works in the literature that couple co-evolution and clustering, for instance, [19] uses a co-evolution clustering algorithm for minimizing both the clustering error and the result quality of the algorithm. In [20] the author uses co-evolution for selecting a good set of attributes for the clustering algorithm using co-evolution. In [21] the author uses cooperative clustering for grouping documents and the words that form them in a cooperative and simultaneous manner. [22] proposes a co-evolutionary algorithm for the dynamic adjustment of feature weights during data clustering. Two populations are simultaneously evolved for the optimization of both the clusters and their associated feature weights. A novel hierarchical co-evolutionary clustering tree-based rough feature game equilibrium selection algorithm is presented in [23]. It aims to select out the high-quality feature subsets, which can enrich the research of feature selection and classification in the heterogeneous big data. These approaches differ greatly from the one exposed in this work.

III. Co-evolutionary Clustering

Co-evolutionary algorithms differ from traditional evolutionary algorithms by maintaining two or more populations at the same time for optimizing different, but coupled, objective functions. The mechanics of co-evolutionary algorithms is simple. Both populations try to maximize their profit in respect to the decision of the other population until neither one can improve their solution independently. The configuration that does not allow for any individual in both population to improve their fitness without changing individuals of the

other population is called a *Nash Equilibrium*. In this state, no population can improve their quality individually.

This section presents the contribution of this work: the development of a *co-evolutionary* algorithm for clustering large datasets called *COCLU*. Co-evolution is employed in this work for choosing the most difficult points to cluster in one population and the best algorithms to cluster those points, in another population. The subset of points that are the hardest to cluster contributes the most to the *SSE* of a given clustering solution. Similarly, a good set of algorithms for hard points consists in algorithms capable of clustering well those points, reducing the overall SSE.

A. Co-evolutionary Algorithms

Co-evolutionary algorithms try to mimic the idea of *predatory* or *symbiotic* behavior that regulates populations in nature. Symbiotic approaches enforce cooperation of populations for achieving a common goal, while predatory behavior introduces competition: if the prey population wishes to survive, it has to adapt itself to its predators. The opposite is also true, i.e., the predator must react to changes in prey configuration if it wishes to maintain its existence. In nature, this arms race constitutes an important evolutionary force that leads to very complex evolution patterns. This work uses the predatory interactions between two populations as an inspiration for the proposed algorithm.

1) Using co-evolution for clustering large data sets: This approach relies on the fact that not all points are equally important for the clustering process. It is possible to select a subset of the original dataset without changing the final clustering result. Figure 1 presents this idea: if the clustering algorithm is run in a subset of good, selected points, it is expected that the final clustering result will be the same. However, not every subset is a good one. Also, random selection of points tends to leave important points out of the clustering procedure, resulting in poor results.

From this point forward, "selected points" will denote a subset of points of the original data set D that is being considered for the clustering algorithms. Similarly, "non-selected points" will denote the set of points that is not being considered in the clustering algorithms.

It is clear that the selection of hard points to cluster (those farthest from the centroids) generates poor SSE results, specially when the underlying clustering solution is bad. Therefore, to find good overall solutions, it is required to find hard points to cluster, i.e., a set of points that are the hardest to cluster for every possible centroid distribution. A good solution for those points should yield also a good overall clustering result.

This work develops a way for iteratively building a hard set of selected points to cluster given a current centroid solution. To select these hard points, the first step is to formalize the notion of "easy" selected points in relation to a clustering solution. These easy points need to be exchanged with harder points to improve the representativity of the selected point. The *easiness criterion* considers how much a selected point influences a fixed clustering solution, i.e., how the point contributes to the overall clustering result given a centroid solution. The smaller the contribution (the closer they are from the closest centroid) the more probable must be the swapping of that point to a random, non-selected point.

(a) Original data set - All points will be used as input for the clustering algorithm. The "x" marks the calculated centroids

(b) A good point selection - The clustering result in the sampled (selected points) set is the same as the original dataset

(c) A bad point selection - The clustering result in the sampled set (selected points) is worse than the original dataset

Figure 1. Example to demonstrate that some subsampling choices result in a far worse clustering result. Here, the doted grey circles represent points being discarded (non-selected points) and solid black circles points being considered (selected points). Figure (c) displays a clustering result that yields a poor SSE result in the original data set.

After some points swapping, the algorithm performs the adaptation of the centroids to the newly selected points by updating the position of the centroid. In this step the selected points are fixed and the clustering algorithms try to find a good solution for the new set of selected points.

The observed dynamic here is that centroids will migrate to the region that has the hardest points, making the otherwise hard selected points easier. The centroids tend to chase regions of selected points, while selected points flee from the centroids.

Once the new clustering solution is found, the algorithm loops to the first step and repeats the point selection procedure. This two-step co-evolutionary procedure is repeated until a given number of iterations is passed or convergence is reached.

The population of selected points subsets is called *Population* A. As previously stated, it encodes which points of the original data set will be used for clustering the data, i.e., a small set of representative points that contains sufficient information for minimizing the SSE of the whole population. The goal of the individuals in population A is to maximize the cost of the clustering solutions generated by individuals of population B. The Population A also maintains the best clustering solution for each individual, i.e., the clustering solution that minimizes the SSE of the selected points.

Individuals in population A suffer two modification operations: *crossover* and *mutation*. *Crossover* works by randomly exchanging selected points of two individuals sampled from the population A and generating one new individual. *Mutation* works by randomly picking a point $\vec{a}$ of population A, and randomly exchanging the value of one of its values.

Population B encodes a set of traditional clustering algorithms that aims to minimize the clustering cost (SSE) given all individuals of population A. While population A evolves towards the set of difficult points to cluster in the SSE sense, population B evolves towards selecting the most efficient algorithms for clustering the points of population A. A small number of algorithms is maintained in the algorithm pool and in each evaluation cycle of population B, one randomly selected inactive algorithm and one randomly selected active algorithm are tested against one randomly selected individual of population A, if the inactive algorithm has a smaller SSE result, it becomes active, entering population B while the old active algorithm leaves population B.

For choosing both the best points and clustering algorithms, the fitness function for a given individual must be defined. The fitness of a given individual x of population A, $f_p(\vec{x})$, is calculated via the min-max method: the fitness of $\vec{x}$ is the best clustering result (the minimum SSE) considering all algorithms in population B. The fitness of a individual y in the population B is the worst result (the largest SSE) resulting in running algorithm y in all individuals of population A.

For population A, the bigger the value of the fitness function, the better the individual is. Thus, after generating some new individuals via crossover and mutation, population is trimmed to its original size SA_{pop}, keeping only the individuals with biggest fitness. Population A evolves towards a sample of difficult points to cluster. If the sample size is big enough to represent the whole population, the result of the clustering in the sample will converge to the result in the whole data set.

Figure 2 shows a representation of the procedure where the individual x_k is being evaluated against all individuals of population B in order to calculate its fitness. In other words, the clustering cost of the selected points in x_k is evaluated against all clustering algorithms existing in population B in the SSE sense. The fitness of individual x_k is the smallest SSE considering all solutions in population B. The bigger the value of the fitness of x_k, the better the individual. In other words, the most difficult set of points to group in the SSE sense is the one with greatest fitness.

Figure 3 illustrates how selecting points that worsen the current solution found by algorithms in population B improve the overall clustering result.

After the co-evolutionary procedure has finished, the best algorithm of Population B is applied on the best individual set of points on population A. Then, the k formed clusters are used for defining to which cluster the remaining points of the dataset are included. These points belong to the cluster with closest centroid. Finally, the overall SSE is computed.

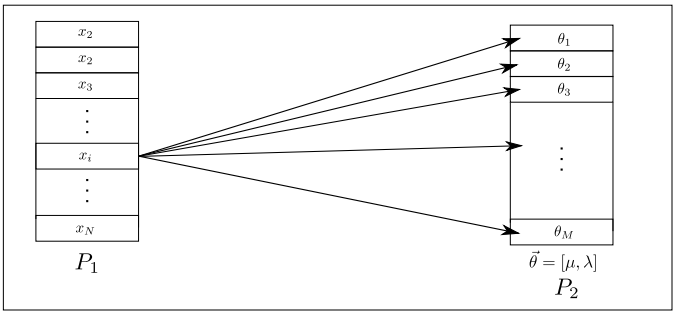
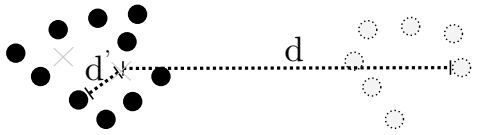
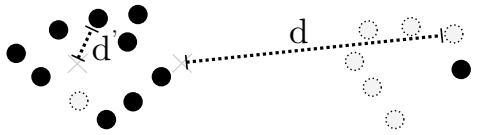


Figure 2. Min-max representation

(a) Initial set of points - Bad point choices leading to poor clustering results



(b) Random swapping of points - Exchanging the near point (with distance d' to a distant point in relation to the closest centroid, with distance d).



(c) After the swap, the clustering algorithm is run again and the centroids walk towards a better configuration. The resulting swapping improved the clustering result slightly - Another swapping selection is displayed, the point with distance d' will be replaced by the point with distance d in the next iteration.

(d) The result of the previous point swapping yields a much better overall clustering. The generated clusters in the whole population would generate the optimum solution in this example.

Figure 3. The Figures 3(a)-3(d) illustrate how the swapping procedure of points iteratively constructs a better solution on the original data set by choosing data points that worsen the solution found in the selected set.

The next subsection describes the details of the *COCLU* approach.

B. The COCLU Algorithm

The *COCLU* algorithm uses the basic framework of the general co-evolutionary algorithm for solving constrained optimization problems. Two populations A and B, with a coupled objective function $f(\vec{x}, y)$ ($\vec{x}$ coming from population A and y coming from population B), compete towards opposite

goals, an individual $\vec{x}$, of population A, tries to maximize the function $f(\vec{x}, B)$, defined as follows:

$$f(\vec{x}, B) = \min(f(\vec{x}, y), \forall y \in B) \tag{4}$$

While an individual y, of population B, tries to minimize the function $f(A, y)$, defined as follows:

$$f(A, y) = \max(f(\vec{x}, y), \forall x \in A) \tag{5}$$

Both populations will go through the process of finding new solutions until they converge, i.e., no improvement is possible in either one of them. This point is called the *saddle point* of the min-max problem [24]. The generic classical co-evolutionary algorithm for finding the saddle point of the min-max problem is described in Algorithm 1.

Algorithm 1 The generic co-evolutionary algorithm for solving a min-max problem

```
procedure CoEvo(Number of Cycles (MaxCycles) ,
Number of Iterations on Population A (MaxGenA),
Number of Iterations on Population B (MaxGenB))
    Initialize Population A
    Initialize Population B
    for each k = 1 to MaxCycles do
        for each j = 1 to MaxGenA do
            Evaluate new Population A
            Generate new Population A
        end for
        for each j = 1 to MaxGenB do
            Evaluate new Population B
            Generate new Population B
        end for
    end for
    Return the best individual of population A given some
quality metric.
end procedure
```

The encoding of the individuals in the population and the details of procedures of initialization, generation and evolution of populations A and B will be described in the following subsections.

1) Encoding of the Individuals: Each individual in population A corresponds to a s-sized subset of distinct point indexes of the original data set. The encoding of these individuals is a vector of size s containing in each position a unique data point index of the original data set. This representation is convenient for both performance and operation definition reasons. The parameter s regulates the summarization ratio of the co-evolutionary algorithm. If s is too big, there is no performance gain in executing the co-evolutionary algorithm.

The definition of population A is formalized below.

$$\vec{x_i} \in A, i > 0, i \leq SA_{pop} \tag{6}$$

$$\vec{x_i} \in \mathbb{N}^s, s \leq n \tag{7}$$

$$\vec{x_{i,j}} \neq \vec{x_{i,k}} \forall i, j, k, (j \neq k) \tag{8}$$

Individuals in population B, on the other hand, are simply a fixed-sized set of clustering algorithms extracted from a pre-defined algorithm pool.

2) Initialization of the Populations:

a) Population A: Initialization of population A is a simple random uniform sub-sample of size s of the data set D, performed multiple times. The only caveat is to remove repetitions from the generated sub-sample.

b) Population B: The initialization of the population is simple: one needs only to sample s_b algorithms from the algorithm pool to initialize the B population and randomly select an individual of population A for solving the reduced clustering problem using the selected clustering algorithms.

3) Fitness Evaluation: Evaluation of the fitness of the individuals in both populations is similar; one has to select an individual in one population and iterate over the other population looking for the minimum (for an individual of population A) or the maximum (for an individual of population B).

4) Generation of Population: The generation of the new individuals of population A consists of simply applying mutation and crossover, and then selecting the best individuals to keep in the population. These operations are described below.

a) Mutation of individuals in population A: A number of N_{mut} individuals of the population A (of selected points) is picked at random and a point of the individual is randomly picked as a pivot. After that, another point is picked from the non-selected set of points to exchange the first selected point. If the replacing procedure yields a smaller *SSE*, the mutation is accepted. This process is repeated n_{tries} times. The new individual is appended to the end of the current population.

b) Crossover of individuals in population A: Crossover is implemented by randomly choosing two individuals using roulette wheel selection and building one new individual by randomly picking the attributes of both individuals, always verifying if the resulting individual is valid, i.e., has no selected point appearing twice.

c) Selection of individuals in population A: After mutation and crossover, the selection of individuals is performed. This selection is merely the elimination of the most unfit individuals so that the population returns to its original size.

Population B evolves through replacement of algorithms in it. In every iteration of the co-evolutionary algorithm on population B, the N_{unfit} most unfit clustering algorithms are replaced by N_{unfit} clustering algorithms of the algorithm pool, selected at random. This approach has two important characteristics: it automatically chooses the more efficient algorithms to the data and avoids local minima by always trying new solutions for the existent individuals. Once a new algorithm is selected, it randomly chooses one individual of population A, proportional to its fitness for finding a solution for the reduced clustering problem. Once a solution for the reduced clustering problem is found, it replaces the old clustering solution if its SSE is smaller than one of the current clustering algorithm present in the population.

C. Clustering algorithms used for population B

In theory, any clustering algorithm could be used as the co-clustering engine, however, it is suggested to select a set of widely spread algorithms in the literature, having attested performance in small data sets and fast response times.

IV. Clustering Results

Preliminary experiments for comparing the COCLU approach with some traditional clustering algorithms are described in this section. Before presenting the results, the benchmark datasets are described, with the original source and some considerations. Also, the parameters values used by the algorithms are shown.

A. Benchmark data sets

Four benchmark datasets extracted from the literature were used. Since there is no dataset that contains the number of clusters to be formed, the number of clusters varies in the interval between 2 and 29 for evaluating the algorithms. Table I presents the details of the chosen benchmark data sets used in this work.

Each algorithm was run 10 times for each test to measure the median and variance of their SSE value (when there is some stochastic behavior in the algorithm). Some tests could not be run due to the impossibility of setting up important algorithm parameters or because of the excessive running times.

B. Setup of algorithms

The default set of parameter values used for all data sets is shown below. These values are the recommended set up found in the literature.

1) CLARANS: The parameters $maxNeighbors$ and $numLocals$ follow the recommendation of the seminal paper of *CLARANS* [12] which suggests using 2 for $numLocals$ and the following expression for $maxNeighbors$:

$$maxNeighbor = k \cdot (|D| - k) \cdot 0.0125 \qquad (9)$$

2) BIRCH: The *BIRCH* algorithm's most important parameters (the closeness parameter and the compactness parameters) are also the most sensitive and hard to estimate, since they have direct relation to the nature of the dataset. Several experiments were performed to find the values described in Table II. The same value was used for both parameters, since it was observed no performance gain in selecting them independently.

3) CURE: The *CURE* algorithm has only one relevant parameter, the size of the representative set. According to the literature, a good value for this parameter is 10.

4) DBSCAN: The *similarity factor* controls indirectly how many clusters the algorithm will form. This value had to be indirectly set up with a local search algorithm.The value for $MinPts$ was set up based on suggestions taken from the literature and experimentation on the available datasets. A good value for this parameter was found to be 2 [11].

5) COCLU: All test runs used the parameters values presented in Table III. These values were estimated in preliminary runs of the algorithm on the datasets. The algorithm pool was compound by k-means [1], *CURE*, *CLARANS* and Spectral [27] clustering algorithms.

C. Results

This section presents the results of algorithms for all datasets. The *DBSCAN* algorithm could only partially run on the BRD14051 dataset. In the others datasets, the algorithm failed to find the proper value for the *similarity factor*, generating different values of k in every occasion. Therefore, its results on these datasets were omitted.

Figures 4 and 6 shows similar results of the algorithms on the BRD14051 and PLA85900, respectively. Both *COCLU* and *CLARANS* algorithms achieved the best SSE values. The *BIRCH* algorithm was the third best algorithm, achieving very close results. The *CURE* algorithm had the worst results with an erratic behavior, probably due to poor summarization of the data.

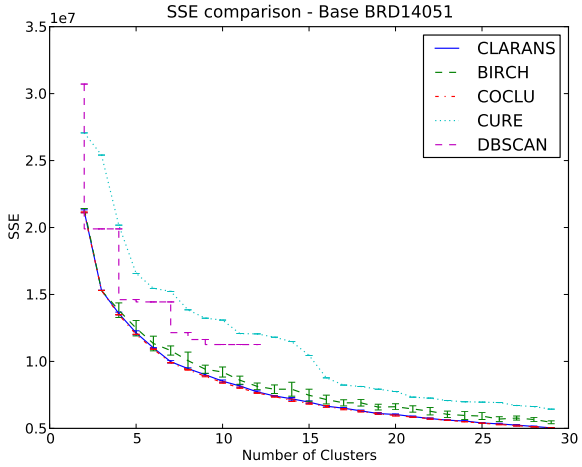


Figure 4. The SSE of all tested algorithms on dataset BRD14051

Figure 5 presents the results without the *CURE* algorithm since it could not perform well on the Shuttle dataset. From these results it is clear that the *COCLU* algorithm had a better clustering result the most part of the experiment, only being slightly worse in the beginning of the experiments.

In the MiniBooNE dataset the only algorithms that were capable of successfully returning valid results in reasonable running times were the *CLARANS* and *COCLU* algorithms. The results observed on Figure 7 are different from those observed in the previous tests. *CLARANS* and *COCLU* had similar results but both present a linear behavior.

V. Conclusion and Future Work

This work proposes a novel clustering algorithm for large datasets, called *COCLU*, for dealing with the problem of clustering large datasets of continuous points in metric spaces. The idea is to apply classical algorithms that perform well in

Table I
DATASET FEATURES - THE NUMBER OS CLUSTERS VARY BETWEEN 2 AND 29 FOR TESTING. ONLY THE TRAINNING DATASETS WERE USED IN THE ALGORITHM'S EVALUATION

Base name	Size	Dimensions	Original source
BRD14051	14051	2	[25]
Shuttle	43500	9	[26]
PLA85900	85900	2	[25]
MiniBooNE_PID	130065	50	[26]

Table II
VALUES OF THE MOST IMPORTANT PARAMETERS OF THE BIRCH ALGORITHM FOR ALL DATA SETS

Dataset	Closeness and compactness parameter
BRD14051	9,000
PLA33810	$4 \cdot 10^7$
Shuttle	86
PLA85900	700,000
MiniBooNE	$9 \cdot 10^{28}$

Table III
PARAMETER VALUES OF THE COCLU ALGORITHM FOR ALL DATA SETS

Parameter	Value	Description
$MaxCycles$	50	Number of outer iterations of the min-max algorithm (Algorithm 1)
$MaxGensA$	10	Number of iterations over the population A while population B is frozen. (Algorithm 1)
$MaxGenB$	1	Number of iterations over the population B while population A is frozen. (Algorithm 1)
SA_{pop}	4	Size of the population A
$ncross$	1	Number of crossovers on each iteration of the outer loop
$nmut$	1	Number of mutations on each iteration of the outer loop
s	$0.05 \cdot \lvert D \rvert$	The number of points that need to be considered by a given individual of the population
SB_{pop}	1	Size of the population B of clustering algorithms

small datasets on a subset of the original data. The problem of choosing a good, representative subset of points is solved by applying a co-evolutive min-max approach for the problem of selecting representative points.

Preliminary experiments performed in this work indicate that *CLARANS* and *COCLU* algorithms were always among the best algorithms in regard to solution quality. In addition, both *CLARANS* and *COCLU* algorithms had very competitive running times (not presented in this paper due to space restrictions). The *COCLU* algorithm had longer running times in the smaller datasets than in the larger ones. The relative running times of the *COCLU* algorithm compared to the *CLARANS* algorithm decreased as the size of the data sets increased, i.e., the bigger the dataset, the less advantageous is to use the *CLARANS* algorithm.

Given that *COCLU* performed very similarly to its constituent algorithm *CLARANS* in the experiments, it is important to understand how often *CLARANS* is selected by the co-evolution framework. The risk is *COCLU* be basically

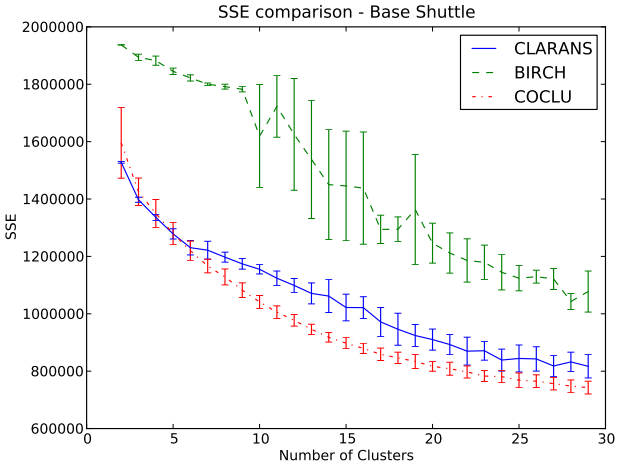


Figure 5. The SSE of all tested algorithms, with the exception of the DBSCAN and CURE algorithms (for a better visualization), on data set Shuttle

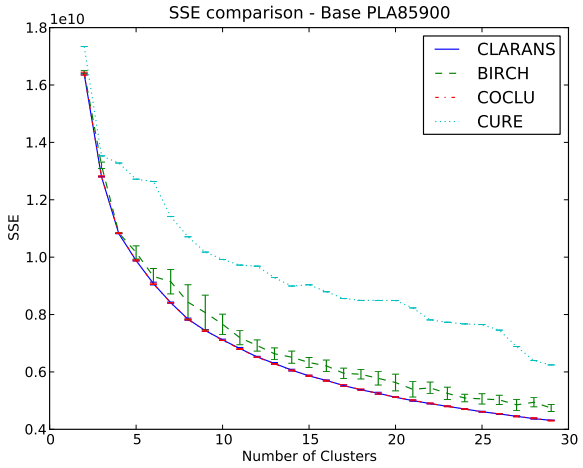


Figure 6. The SSE of all tested algorithms, with the exception of the DBSCAN algorithm, on data set PLA85900

CLARANS, which would falsify the usefulness of the new co-evolution framework.

Nevertheless, the premise that no single clustering algorithm is better for every base is widely known. Thus, an algorithm capable of unifying the decisions of many different heuristics is relevant. The *COCLU* algorithm may fill the gap between clustering techniques and datasets by using co-evolution for selecting both points and algorithms to consider in the overall

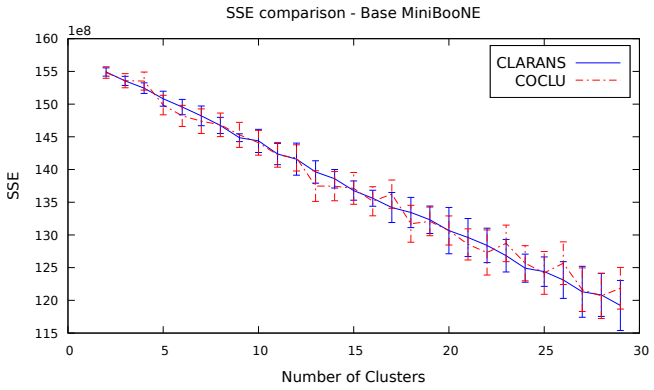


Figure 7. The SSE of CLARANS and COCLU, on data set MiniBooNE

clustering process. This automatic selection yields a robust algorithm for clustering many different types of datasets, given that the adequate clustering algorithm is present in the population pool.

Future works should perform a wider and more robust experimental investigation including larger and synthetic datasets, other clustering algorithms such as improved versions of k-means and different evaluation measures, other than SSE. In particular, it must investigate what extent a co-evolutionary framework is strictly adding value here and whether the same could be achieved through use of a single clustering algorithm.

The use of the min-max approach for solving complex problems often leads to algorithms with poor scalability capabilities. Fixing one individual of population A, all the fitness calculations on the opposite population may be done at the same time, reducing the runtime of the algorithm greatly. With this enhancement, more tests can be executed with even bigger datasets with different characteristics to verify if the observed behaviour is maintained.

More investigation should also be done on selecting clustering algorithms for the core of the *COCLU* algorithm. Changing the current set of algorithms (k-means, *CURE*, *CLARANS* and *Spectral clustering*) may improve the final result of the algorithm.

References

[1] J. B. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability* (L. M. L. Cam and J. Neyman, eds.), vol. 1, pp. 281–297, University of California Press, 1967.

[2] P. Franti, O. Virmajoki, and T. Kaukoranta, "Branch-and-bound technique for solving optimal clustering," in *Proceedings of the 16th International Conference on Pattern Recognition, 2002*, vol. 2, pp. 233–235, IEEE, 2002.

[3] R. Ostrovsky and Y. Rabani, "Polynomial time approximation schemes for geometric k-clustering," in *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, (New York, NY, USA), pp. 349–358, ACM, 2000.

[4] S. Das, A. Abraham, and A. Konar, *Metaheuristic clustering*, vol. 178. Springer, 2009.

[5] Z. Michalewicz, "Evolutionary computation techniques for nonlinear programming problems," in *International Transactions in Operational Research*, vol. 1, (University of North Carolina, USA), pp. 223–240, Elsevier Science, 1994.

[6] F. van den Bergh and A. Engelbrecht, "A cooperative approach to particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, vol. 8, pp. 225–239, june 2004.

[7] D. A. Augusto, H. J. Barbosa, and N. F. Ebecken, "Coevolution of data samples and classifiers integrated with grammatically-based genetic programming for data classification," in *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, GECCO '08, (New York, NY, USA), pp. 1171–1178, ACM, 2008.

[8] H. J. C. Barbosa, "A genetic algorithm for min-max problems," *Proceedings of the First International Conference on Evolutionary Computation and its Applications*, pp. 99–109, 1996.

[9] K. Alsabti, S. Ranka, and V. Singh, "An efficient k-means clustering algorithm," *Electrical Engineering and Computer Science*, no. 43, 1997.

[10] D. Arthur and S. Vassilvitskii, "How slow is the k-means method?," in *Proceedings of the twenty-second annual symposium on Computational geometry*, SCG '06, (New York, NY, USA), pp. 144–153, ACM, 2006.

[11] M. Ester, H. peter Kriegel, J. S, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," pp. 226–231, AAAI Press, 1996.

[12] R. T. Ng and J. Han, "CLARANS: A Method for Clustering Objects for Spatial Data Mining," *IEEE Trans. on Knowl. and Data Eng.*, vol. 14, pp. 1003–1016, Sept. 2002.

[13] Y. Luo, A. Joshi, A. Phansalkar, L. John, and J. Ghosh, "Analyzing and improving clustering based sampling for microprocessor simulation," 2008.

[14] T. Zhang, R. Ramakrishnan, and M. Livny, "Birch: an efficient data clustering method for very large databases," in *Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, SIGMOD '96, (New York, NY, USA), pp. 103–114, ACM, 1996.

[15] S. Guha, R. Rastogi, and K. Shim, "Cure: an efficient clustering algorithm for large databases," in *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, SIGMOD '98, (New York, NY, USA), pp. 73–84, ACM, 1998.

[16] J. Sander, M. Ester, H.-P. Kriegel, and X. Xu, "Density-based clustering in spatial databases: The algorithm gdbscan and its applications," *Data Min. Knowl. Discov.*, vol. 2, pp. 169–194, June 1998.

[17] J. Poon and M. Maher, "Co-evolution and emergence in design," *Artificial Intelligence in Engineering*, vol. 11, no. 3, pp. 319 – 327, 1997.

[18] M. L. Maher, J. Poon, and S. Boulanger, "Formalising design exploration as co-evolution: A combined gene approach," 1996.

[19] D. Chakrabarti, R. Kumar, and A. Tomkins, "Evolutionary clustering," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '06, (New York, NY, USA), pp. 554–560, ACM, 2006.

[20] Q. He and L. Wang, "An effective co-evolutionary particle swarm optimization for constrained engineering design problems," *Engineering Applications of Artificial Intelligence*, vol. 20, no. 1, pp. 89–99, 2007.

[21] I. S. Dhillon, "Co-clustering documents and words using bipartite spectral graph partitioning," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '01, (New York, NY, USA), pp. 269–274, ACM, 2001.

[22] M. I. Hosny, L. AlHinti, and S. Al-Malak, "A co-evolutionary framework for adaptive multidimensional data clustering," *Intelligent Data Analysis*, vol. 22, no. I, pp. 77–101, 2018.

[23] W. Ding, C.-T. Lin, and M. Prasad, "Hierarchical co-evolutionary clustering tree-based rough feature game equilibrium selection and its application in neonatal cerebral cortex mri," *Expert Systems with Applications*, vol. 101, no. 1, pp. 243–257, 2018.

[24] D.-Z. Du and P. M. Pardalos, *Minimax and applications*, vol. 4. Springer, 1995.

[25] G. Reinelt, "TSPLIB- a traveling salesman problem library," *ORSA Journal of Computing*, vol. 3, no. 4, pp. 376–384, 1991.

[26] A. Frank and A. Asuncion, "UCI machine learning repository," 2010.

[27] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*, pp. 849–856, MIT Press, 2001.