EvoEvo Deliverable 5.1

Guillaume Beslon, Jonas Abernot, Sergio Peignier, Christophe Rigotti

EvoEvo Deliverable 5.1

Impact obtained from EvoEvo mechanisms on data stream cluster analysis

Due date: M36
Person in charge: Guillaume Beslon
Partner in charge: INRIA
Workpackage: WP5 (EvoEvo applications)
Deliverable description: Impact obtained from EvoEvo mechanisms on data stream cluster analysis: A report on the development and results of the data stream cluster analysis software. The report should identify the strengths and weaknesses of the EvoEvo approach for this application.

Revisions:

Revision no.	Revision description	Date	Person in charge
1.0	First version	19/09/16	J. Abernot - S. Peignier (INRIA)
1.1	Corrections and complement	14/10/16	C. Rigotti (INRIA)
1.2	Corrections and validation	24/10/16	G. Beslon (INRIA)

Abstract

Subspace clustering is a data mining task that searches for objects that share similar features and at the same time looks for the subspaces where these similarities appear. For this reason Subspace clustering is recognized as more general and complicated than standard clustering, since this last task requires only to detect groups of similar objects or clusters. In this report we present ChameleoClust$^+$, an evolutionary algorithm to tackle the subspace clustering problem. ChameleoClust$^+$ is a bio-inspired algorithm implementing an evolvable genome structure, including several bio-like features such as a variable genome length, both functional and non-functional elements and mutation operators including chromosomal rearrangements. The main purpose of the design of ChameleoClust$^+$ is to take advantage of the large degree of freedom provided by its evolvable structure to detect various number of clusters in subspaces of various dimensions. This algorithm was assessed and compared to the state of the art methods, with satisfying results, on a reference benchmark using both real world and synthetic datasets. While other algorithms may need more complex parameter setting, ChameleoClust$^+$ needs to set only one subspace clustering ad-hoc parameter: the maximal number of clusters. This single parameter is responsible for setting the maximal level of detail of the subspace clustering, and is a quite intuitive parameter. The remaining parameters of ChameleoClust$^+$ are related to the evolution strategy (population size, mutation rate, ...) and it is possible to use a single setting for them, that turns out to be effective enough for all the benchmark datasets. A sensitivity analysis has also been carried out to study the impact of each parameter on the subspace clustering quality. This report also presents *Evowave*, an application of ChameleoClust$^+$ to analyze a real dynamic stream.

1 Introduction

Clustering is a data mining task that aims to group objects sharing similar characteristics into a same cluster over the whole data space. Usually similarity between objects is determined using a distance function. Subspace clustering purpose does not only imply identifying groups of similar objects, it also aims to detect the subspaces where similarity occurs. Subspace clustering can be conceived as *"similarity examined under different representations"* [1]. It is for this reason that subspace clustering is recognized as a more complicated and general task than standard clustering. Moreover retrieving meaningful subspaces turns out to be particularly useful while dealing with high dimensional data [2].

Several evolutionary clustering approaches have been proposed [3], however very few of them address the subspace clustering task. Two earlier approaches [4] and [5] require non-evolutionary steps to tackle this problem. Instead, the algorithm presented in the report addresses the subspace clustering task relying on evolution of evolution principles. According to [6] knowledge from evolutionary and molecular biology should be taken into account in the interest of conceiving better bio-inspired optimization algorithms. Among important phenomena in evolutionary biology, the dynamic evolution of the genome structure appears as a promising source of advances for bio-inspired optimization. Important phenomena such as the variable genome length or the variable percentages of coding or functional elements within the genome are related to

the evolution of genome structures phenomenon [7]. Several studies have shown for instance that an evolvable genome structure allows evolution to shape the effects of evolution principles themselves (e.g. mutations), phenomenon known as *evolution of evolution* (EvoEvo) [8]. Among the state-of-the-art formalisms used for *in silico* experimental evolution reviewed in [8], two models enable genome structure evolution: [7] and [9]. Both formalisms have inspired key aspects of our work.

In this report, we present ChameleoClust$^+$, an evolutionary algorithm that takes advantage of a genome having an evolvable structure to tackle the subspace clustering problem. This algorithm is an extension of the work presented in [10] (best paper award in the category Evolutionary Machine Learning at the conference GECCO-2015).

ChameleoClust$^+$ genome is a *coarse-grained* genome, inspired on [9], and defined as a list of tuples of numbers. The genome is mapped at the phenotype level by using the genome tuples to denote core point locations in different dimensions, which are then used to build the subspace clusters. Furthermore the genome also contains a variable proportion of non-functional elements as in [7]. During replications the genome undergoes both local mutations and large random rearrangements similar to those used in [7] and [9], namely: large deletions and duplications. Local mutations modify the genome elements and rearrangements modify the genome length and the proportion of non-functional elements. The key intuition in the design of the ChameleoClust$^+$ algorithm is to take advantage of such an evolvable structure to detect various number of clusters in subspaces of various dimensions. In addition, ChameleoClust$^+$ takes advantage of the genetic memory through evolution to evaluate the fitness over a sliding dataset sample, leading to an important reduction of the execution time, without effective degradation of the clustering quality. Moreover the use of a sliding dataset sample extends the possibility of using the algorithm for the analyze of data streams.

In order to assess ChameleoClust$^+$ we used the reference subspace clustering evaluation framework presented in [11]. ChameleoClust$^+$ was compared to state-of-the-art algorithms on both real and synthetic datasets. The experiments show that ChameleoClust$^+$ obtains competitive results with a single parameter related to the domain, i.e., the maximal number of clusters. We also carried out a sensitivity analysis, by varying the main parameters one-at-a-time to study their impact on the cluster quality. Finally, the capacities of ChameleoClust$^+$ to perform subspace clustering on data streams were assessed using a real world data stream containing the description of different wifi contexts in an application called *Evowave*.

The rest of the report is organized as follows. The next section introduces the proposed algorithm, and Sections 3 and 4 describe respectively the evaluation method and results. Section 5 presents the real world stream analysis. Strength and Weakness of the method are discussed both in Sections 4 and 5. And we conclude in Section 6.

2 ChameleoClust$^+$

ChameleoClust$^+$ includes several bio-like features such as a variable genome length and organization, presence of both functional and non-functional tuples,

and variation operators including large chromosomal rearrangements. These features, inspired by the *in silico* experimental evolution formalisms of [7] and [9], give the algorithm a large degree of freedom by making the genome structure evolvable. ChameleoClust$^+$ takes advantage of this structural flexibility to build subspace clustering with various number of clusters and in subspaces having different numbers of dimensions.

2.1 Dataset and clusters

A dataset $\mathcal{S} = \{s_1, s_2 \dots\}$ is a set of objects. Each object has a unique identifier and is described in $\mathbb{R}^D$ by D features (the coordinates of the objects). The size of S is the number of objects in S, and D is the number of dimensions (i.e., the dimensionality) of $\mathcal{S}$. Each dimension is represented by a number from 1 to D and the set of all dimensions of the dataset is denoted $\mathcal{D} = \{1, \dots, D\}$. The algorithm takes as input a dataset S and a parameter c_{max} that is the maximal number of desired clusters. The algorithm outputs a subspace clustering in the form of a set of disjoints clusters, where each cluster is defined as a set of objects and a set of dimensions.

2.2 Overall clustering principle

Each individual encodes in its genome a subspace clustering. More precisely a genome defines a set of so called *core points* located in various subspaces having possibly less than D dimensions. If the objects of the dataset tends to form groups around these core points, then a high fitness is associated to the corresponding individual. The reproduction (including selection and mutations) is performed for a whole generation in a synchronized way. After a given number of generations the process is stopped and the subspace clustering corresponding to the individual having the highest fitness is retained.

2.3 Preprocessing

As in many typical clustering problems, the first step is to standardize the dataset to ensure that all features could have similar impact on the distance computation during the clustering. Thus each feature value x is replaced by its z-score: $z = \frac{x-\mu}{\sigma}$, where μ is the dataset mean and σ is dataset standard deviation for the given feature. After standardization, data values in different dimensions are independent of the original offset and scale, and all features have the same unitary standard deviation and a zero mean (i.e., the entire dataset is centered around $\mathcal{O}$). Finally the maximal value among all absolute values of the z-score of all features is computed and is noted x_{max} in the rest of the report.

2.4 Genome structure

A genome Γ is a list $[\gamma_1, \dots, \gamma_i, \dots, \gamma_n]$ of tuples of the form $\gamma_i = \langle g, c, d, x \rangle$, where $g \in \{0, 1\}$ indicates if γ is a functional tuple of the genome ($g = 1$) or not ($g = 0$), and c, d, x are used to define the phenotype only if $g = 1$. The previous elements have the following specific domains: $c \in \{1, \dots, c_{max}\}$, $d \in \{1, \dots, D\}$ and $x \in ValCoord$, with $ValCoord = \{j \times x_{max}/1000 \mid j \in \{-1000, \dots, 1000\}\}$, i.e. all values from $-x_{max}$ to x_{max} with step $x_{max}/1000$. The genome structure

previously defined is evolvable: The number of functional and non-functional elements and their respective positions in the genome may change. In Section 4.1 we show the adaptation of the genome size and the ratio of functional elements to each particular dataset and Section 4.4 depicts that non-functional tuples also have a beneficial impact on the subspace clustering quality.

2.5 Phenotype

A phenotype Φ is simply a set of core points. Informally a core point is a specific point, around which objects can be grouped to form a subspace cluster. The number of core points cannot exceed the maximal number of desired clusters c_{max}. Each core point is identified by a number $c \in [1, c_{max}]$ and is denoted p_c. The intuition of the genotype-phenotype mapping is that each functional element of the genome $\langle 1, c, d, x\rangle$ is a contribution of value x to the location of core point p_c in dimension d. More precisely, let x_d be the coordinate of p_c for dimension d, then x_d is the sum of all the values x contained in a tuple of the form $\langle 1, c, d, x\rangle$ in the genome Γ. The subspace associated to p_c (and for which p_c is defined) is the set of dimensions $\mathcal{D}_{p_c} = \{d \mid \exists x, \langle 1, c, d, x\rangle \in \Gamma\}$, i.e., the dimensions that contribute to p_c in Γ. Notice that the non-functional elements of Γ do not contribute to the phenotype.

For a given dataset $\mathcal{S}$, a phenotype Φ defines a subspace clustering of $\mathcal{S}$, by associating each object of $\mathcal{S}$ to the best matching core point in Φ. A non empty set of objects associated to a core point p_c forms a cluster in subspace $\mathcal{D}_{p_c}$. The precise definition of the notion of *best match* is given in the section 2.7 hereafter.

Notice that the length of the genome can be different among individuals, leading to phenotypes containing different numbers of core points in various subspaces and thus defining subspace clustering models with different number of clusters in subspaces having different number of dimensions. Notice also that the genotype to phenotype mapping is not bijective, and the same phenotype can be obtained from different genotypes containing different functional or non-functional elements.

2.6 Mutation operators

Each new genome is copied from a parent and modified by biologically inspired mutation operators of two kinds: Global rearrangements and point mutations. These operators are general mutation operators, they are not guided by some criteria related to the subspace-clustering task, and both functional and non-functional elements can be impacted by mutations. For a genome Γ, an application of the point mutation operator is defined as follows.

- **Point substitution**: Let $\gamma_i \in \Gamma$ of the form $\gamma_i = \langle g, c, d, x\rangle$, denote an element uniformly drawn in the genome and let $k \in \{1, 2, 3, 4\}$ a value chosen uniformly. The point substitution operator modifies the k-th element of the tuple γ_i and replace it with a new random number drawn uniformly in its associated range:

$$\gamma_i \leftarrow \begin{cases} \langle \mathcal{U}(\{0,1\}), c, d, x\rangle & \textbf{if } k = 1 \\ \langle g, \mathcal{U}(\{1, \ldots, c_{max}\}), d, x\rangle & \textbf{if } k = 2 \\ \langle g, c, \mathcal{U}(\{1, \ldots, D\}), x\rangle & \textbf{if } k = 3 \\ \langle g, c, d, \mathcal{U}(ValCoord)\rangle & \textbf{if } k = 4 \end{cases}$$

where $\mathcal{U}$ denotes the uniform random selection of a element in a set.

For the rearrangements, Γ is considered as being circular (as bacterial genomes). This means that the tuple γ_n is adjacent to the tuple γ_1. In order to define the possible rearrangements let us define two basic operators.

- Sublist extraction operator:

$$[\gamma_1, \ldots, \gamma_n]_{i,j} = \begin{cases} [\gamma_i, \ldots, \gamma_j] & \textbf{if } i < j \\ [\gamma_i] & \textbf{if } i = j \\ [\,] \text{ (the empty list)} & \textbf{if } i > j \end{cases}$$

- List concatenation operator:

$$[\gamma_1, \ldots, \gamma_n] + [\gamma'_1, \ldots, \gamma'_m] = [\gamma_1, \ldots, \gamma_n, \gamma'_1, \ldots, \gamma'_m]$$

Rearrangements are responsible for increasing or decreasing the genome length. The model uses two kinds of rearrangements: Large deletions and large duplications. For one application of a rearrangement operation on a genome $\Gamma = [\gamma_1, \ldots, \gamma_n]$, a portion of Γ bounded by two tuples $\gamma_i, \gamma_j \in \Gamma$ is considered, where i and j are uniformly chosen in $\{1, \ldots, n\}$. The two rearrangement operators can then be defined as follows:

- **Large deletions**: The segment between tuples γ_i and γ_j is excised.

  If $i \leq j$:

  $\Gamma \leftarrow \Gamma_{1,i-1} + \Gamma_{j+1,n}$

  If $i > j$, because of genome circularity, we have:

  $\Gamma \leftarrow \Gamma_{j+1,i-1}$

- **Large duplications** : The segment between tuples γ_i and γ_j is copied and inserted at the location of a third tuple γ_p (uniformly chosen).

  If $i \leq j$:

  $\Gamma \leftarrow \Gamma_{1,p} + \Gamma_{i,j} + \Gamma_{p+1,n}$

  If $i > j$, because of genome circularity, we have:

  $\Gamma \leftarrow \Gamma_{1,p} + \Gamma_{j,n} + \Gamma_{1,i} + \Gamma_{p+1,n}$

During the reproduction of an individual, the whole mutation stage is defined as follows. For each of the two kinds of rearrangement operations, the total number of rearrangements is drawn from a binomial law $\mathcal{B}(L, u_m)$ where L is the genome size and u_m is the mutation rate (same rate for all mutation operators). Then the corresponding number of large deletions and large duplications are performed in a random order. Once all rearrangements have been applied, the number of point substitutions is drawn from a binomial law $\mathcal{B}(L', u_m)$ where L' is the genome size after applying the rearrangement operations. Then all these point substitutions are carried out.

2.7 Fitness

The fitness of a individual of phenotype Φ is related to the quality of the subspace clustering defined by Φ over a given dataset. This quality measure is a distance-based measure reflecting how the objects in the dataset tend to form groups around the core points of Φ. In [12] and [13] it has been shown that distance comparisons are less meaningful when dimensionality increases, this effect is called the *concentration effect* of the distances. Furthermore, distances do not have the same meaning in subspaces with different numbers of dimensions: And thus it is not fair to compare distances calculated in subspaces with different dimensionality.

It has been shown in [13] that the Manhattan distance is robust to the concentration effect. In the ChameleoClust$^+$ algorithm, the distance used is the *Manhattan segmental distance* introduced in [14] for the well known subspace clustering algorithm PROCLUS. It is a normalized version of the classic Manhattan distance to compare distances in subspaces with different number of dimensions. Let y_1 and y_2 be two points in a space over the set of dimension $\mathcal{D}$, and $y_{1,i}$ (resp. $y_{2,i}$) denotes the coordinate of y_1 (resp. y_2) in the dimension i of $\mathcal{D}$. Then, the Manhattan segmental distance is:

$$d_{\mathcal{D}}(y_1, y_2) = \sum_{i \in \mathcal{D}} \frac{|y_{1,i} - y_{2,i}|}{|\mathcal{D}|}$$

This distance is used here to define a function $\mathcal{E}(x, p_c)$ to assess the mismatch of the assignment of an object $x \in \mathcal{S}_{\mathcal{F}}$ in space $\mathcal{D}$ to a core point p_c in subspace $\mathcal{D}_{p_c}$. The highest is $\mathcal{E}(x, p_c)$, the worst is the association of x to p_c. This function is defined by:

$$\mathcal{E}(x, p_c) = \frac{|\mathcal{D}_{p_c}| \cdot d_{\mathcal{D}_{p_c}}(x, p_c) + |\mathcal{D} \setminus \mathcal{D}_{p_c}| \cdot d_{\mathcal{D} \setminus \mathcal{D}_{p_c}}(x, \mathcal{O})}{| \mathcal{D} |}$$

Where $\mathcal{O}$ is the origin of the entire space.The mismatch evaluation $\mathcal{E}(x, p_c)$ increases with the distance between the core point p_c and the object x (term $d_{\mathcal{D}}(x, p_c)$). $\mathcal{E}(x, p_c)$ also increases if the subspace $\mathcal{D}_{p_c}$ has not enough dimensions to explain the shift of x with respect to $\mathcal{O}$ (term $d_{\mathcal{D}' \setminus \mathcal{D}}(x, \mathcal{O})$). The value $\mathcal{E}(x, p_c)$ is then simply the average of $d_{\mathcal{D}_{p_c}}(x, p_c)$ and $d_{\mathcal{D} \setminus \mathcal{D}_{p_c}}(x, \mathcal{O})$ weighted by their respective subspace dimensionalities.

To evaluate the fitness of an individual with phenotype Φ, each object x in the dataset $\mathcal{S}$ is assigned to the core point $p_c \in \Phi$ for which $\mathcal{E}(x, p_c)$ is minimal (in the rare cases where several core points lead to the same minimal value, then one of them is chosen nondeterministically). Let $\mathcal{S}_{p_c}$ be the set of objects associated to p_c, then if $\mathcal{S}_{p_c}$ is not empty, the core point p_c defines the subspace cluster $\langle \mathcal{S}_{p_c}, \mathcal{D}_{p_c} \rangle$, otherwise p_c defines no cluster.

The fitness $\mathcal{F}$ is then defined as the opposite of the average of the mismatches computed for the best possible assignments for the dataset objects:

$$\mathcal{F}(\Phi, \mathcal{S}) = -\frac{\sum_{p_c \in \Phi} \sum_{x \in \mathcal{S}_{p_c}} \mathcal{E}(x, p_c)}{|\mathcal{S}_{\mathcal{F}}|}$$

The fitness function $\mathcal{F}(\Phi, \mathcal{S})$ goes to 0 when the evaluation of the mismatches between objects and core points tends to 0 (perfect match), and is strongly negative when objects and core points are poorly related. Notice that a core point

p_c with no associated object ($\mathcal{S}_{p_c} = \emptyset$) is not penalized, and its corresponding functional elements in the genome may then be preserved for further exploration during evolution.

To guide the search, it is not necessary to evaluate the fitness over the whole input dataset $\mathcal{S}$, but it is sufficient to evaluate it over a sample $\mathcal{S}^t \subseteq \mathcal{S}$. To avoid misleading consequences of poor sample selections (i.e., sample not very representative of $\mathcal{S}$) the sample $\mathcal{S}^t$ can be changed at each generation t. As shown is Section 4, for reasonable sizes of $\mathcal{S}^t$, this leads to an important reduction of the execution time, without effective degradation of the clustering quality. For this purpose, let us build a list $\mathcal{L} = [x_1, x_2 \dots]$ containing all the dataset points in a random order. At each generation t the population is fed with a set of elements of $\mathcal{L}$ of size ω defined as :

$$\mathcal{S}^t = \bigcup_{k=t\times\omega}^{t\times\omega+\omega} \{x \text{ in } \mathcal{L} \text{ at index } (k \bmod |\mathcal{L}|)\}$$

$\mathcal{S}^t$ is simply the set of objects in $\mathcal{L}$ from index $t \times \omega$ to index $t \times \omega + \omega$, restarting from the beginning of $\mathcal{L}$ when the last element is reached.

The following diagram illustrates the procedure used to build the sliding sample with objects from the static dataset to compute the fitness population at each generation.

$$\underbrace{x_1, x_2, \dots}_{\mathcal{S}^1} \underbrace{\dots\dots\dots}_{\mathcal{S}^2} \dots \underbrace{x_{|\mathcal{L}|}, x_1, \dots\dots}_{\mathcal{S}^t}, x_{|\mathcal{L}|}, \dots$$

The incorporation of a sliding dataset sample also extends the possibility of using the algorithm for the analyze of data streams. In this context we consider the data stream as a list $\mathcal{L} = [x_1, x_2 \dots]$ of data stream objects arriving in chronological order. When a new object x_t arrives, the data sample used to feed the population is updated using a first-in-first-out method and is defined as:

$$\mathcal{S}^t = \bigcup_{k=t-\omega}^{t} \{x \text{ in } \mathcal{L} \text{ at index } k\}$$

$\mathcal{S}^t$ is simply the set of ω most recent objects from the stream. Thereupon the same data sample $\mathcal{S}^t$ is used during τ generations before a new data stream object is inserted into the sample. The procedure used to build the sliding sample with data stream objects is illustrated in the diagram hereafter.

$$\underbrace{x_1, x_2, \dots\dots\dots\dots}_{\mathcal{S}^\omega} \dots \underbrace{x_{t-\omega}, \dots, x_t}_{\mathcal{S}^t} \dots\dots\dots$$

2.8 Population

Each individual can be perceived as an asexual artificial organism containing a single chromosome. The population evolves during T generations. At each generation the population is completely renewed but its size N remains constant over time. As in the evolution simulation model of [7] we rely on an exponential ranking selection [15] in order to use the same distribution for the selection of the individuals all over the evolution (i.e., the selection is not directly related

to fitness values but to ranks). In this selection scheme, the individuals of the current generation are ranked according to their fitness, in increasing order of performance (the worst has rank 1 and the best rank N). Then for each of the N individuals of the offspring generation, the parent of this individual is determined by a trial over a N classes multinomial law, where each class is associated to an individual of the current generation. For this multinomial law, an individual α has a success probability $p_\alpha = (s-1)\frac{s^{(N-r_\alpha)}}{s^N-1}$ where r_α is the rank of the individual α and s the selection pressure parameter.

In order to avoid the best fitness to decrease ChameleoClust$^+$ uses an elitist selection method. More precisely, it always adds in the next generation an unchanged copy of the best current individual, and performs the random reproduction using only $N-1$ trials. In Section 4.4, we will show that elitism has most of the time a beneficial impact on the subspace cluster quality. For the first generation all genomes have the same size, denoted $|\Gamma_{init}|$, and contain only non-functional elements. The genomes of these initial individuals are drawn independently, and filled with random tuples of the form:
$\langle 0, \mathcal{U}(\{1,\ldots,c_{max}\}), \mathcal{U}(\{1,\ldots,D\}), \mathcal{U}(ValCoord\})\rangle$.

3 Experimental setup

3.1 Experimental protocol

In order to evaluate and compare ChameleoClust$^+$ to state-of-the-art algorithms, we used the evaluation framework of reference designed for subspace clustering and described in [11]. This evaluation framework relies on a systematic approach to compare the results of representative algorithms that address the major subspace clustering paradigms. The comparison detailed in [11] was made using different evaluation measures on both real and synthetic datasets. We clustered with ChameleoClust$^+$ the same datasets and computed the same quality measures.

In the framework of [11], as each algorithm requires several parameters (from 2 to 9), they are executed with many different parameter settings to explore the parameter space. Then, using an external labeling of the objects, only the subspace clusterings that are among the best (with respect to the external labeling) are retained. So, the results reported for these algorithms are in some sense the best possible subspace clusterings that could be achieved if we were able to find the most appropriated parameter values. Since generally no external labeling is available when we search for clusters, parameter tuning is most of the time a difficult task and these high quality subspace clusterings are likely to be hard to obtain.

An important point to notice, is that for ChameleoClust$^+$ we did not perform any parameter optimization using external information, but we simply followed the parameter setting guideline presented in Section 3.3. Then, we ran ChameleoClust$^+$ and took the subspace clustering defined by the individual of the last generation having the best fitness. Since the algorithm is non-deterministic, we ran it 10 times in the same conditions and report the minimal, maximal and mean values of the measures over these 10 runs. So, we compare clusterings effectively found by ChameleoClust$^+$ to the best clusterings that could potentially be found by the other algorithms. All experiments were run

on a quad-core Intel 2.67GHz CPU running Linux Ubuntu 14.04, using a single core and less than 250 MB of RAM.

3.2 Datasets

We studied ChameleoClust$^+$ performances on real world data using the six benchmark datasets selected in [11] for their representativity: *breast, diabetes, liver, glass, shape, pendigits* and *vowel* (most of them coming from the UCI archive [16]). These datasets have different dimensionalities and contain different numbers of objects. These objects are already structured in classes, and the class membership is used by quality measures to assess the cluster *purity*. However the number of classes does not necessarily reflect the number of subspace clusters, since even within a class the objects can form several clusters in different subspaces.

We also ran ChameleoClust$^+$ on the 16 synthetic benchmark datasets provided by [11]. These datasets are particularly useful to study the algorithm performances, as the true clusters and their subspaces are known. Each dataset contains 10 hidden subspace clusters laying in subspaces having 50%, 60% and 80% of the total dimensions of the dataset. Seven synthetic datasets were generated in [11] to study scalability with respect to the dataset dimensionality: *D05, D10, D15, D20, D25, D50* and *D75* with 5, 10, 15, 20, 25, 50 and 75 dimensions respectively. These datasets have about 1500 objects each and about 10% of noise objects. In addition to the previous datasets, five synthetic datasets were built to analyse scalability with respect to the dataset size: *S1500, S2500, S3500, S4500* and *S5500* with 1500, 2500, 3500, 4500 and 5500 objects respectively. For these datasets, the number of dimensions was set equal to 20 and the percentage of noise objects close to 10%. Finally four datasets were generated to study the capacity to cope with noise: *N10, N30, N50* and *N70* with 10%, 30%, 50% and 70% of noise objects in the dataset respectively. These datasets were made by adding noise points to the dataset *D20*.

All datasets and additional description are made available by the authors of [11] at http://dme.rwth-aachen.de/openSubspace/evaluation.

3.3 Parameter setting

Sliding sample size The dataset sample used to compute the fitness at each generation should contain enough objects in order to be representative of the entire dataset, but needs to be small enough in order to reduce the runtime. Indeed, if the dataset sample is too small, there are chances that some clusters of the dataset will not be represented by any point in the sample, or that they will be represented by inaccurate points, in this case the data sample is non-representative of the entire dataset and the subspace clusters produced are likely to be inaccurate. We have decided to set the sliding sample size to 10% of the dataset size in order to speed-up the algorithm while keeping enough points to ensure that the sample is representative enough.

Selection pressure Let α be an individual of the current generation and β be an individual of the next generation, according to Section 2.8, α has the probability $p_\alpha = (s-1)\frac{s^{(N-r_\alpha)}}{s^N-1}$ to be the parent of β. For the best individual

($r_\alpha = N$), the previous expression simplifies to $p_\alpha = \frac{s-1}{s^N-1}$. Thus, the selection pressure was set to $s = 0.5$ so that with a large population ($N \gg 1$) the best individual has a success probability close to $p_\alpha \simeq 0.5$. Therefore each individual of the next generation has one chance out of two to descend from the best individual and thus explore its neighborhood and the same chance to descend from a different individual and explore potentially different solutions.

Initial genome size Initial genomes contain only non-functional tuple and their initial size was chosen to be equal to $|\Gamma_{init}| = 200$. This genome size matches with the amount of tuples required to build a typical subspace clustering model, e.g., 10 clusters in 20 dimensions or 20 clusters in 10 dimensions. As the genome size and the genome structure is not constrained and is able to evolve (as illustrated in Figure 3b), the initial genome size is not a determining choice for the algorithm.

A sensitivity analysis performed in Section 4.3 shows that the result quality is not substantially modified for a large range of the three previous parameters.

Mutation rate The mutation rate was set according to its impact on the number of replications that actually produce genomes that are different from or identical to the parental genome. Let φ be the probability that no mutation of any types (substitution, deletion, duplication) occurs during one replication of an individual. As defined in Section 2.6, the number of mutations of a given type that take place during one replication follows a binomial distribution $\mathcal{B}(|\Gamma|, u_m)$. Thus the probability that no mutation of one type occurs is equal to $(1-u_m)^{|\Gamma|}$ and $\varphi = (1-u_m)^{3|\Gamma|}$.

φ depends strongly on the mutation rate and the genome length as illustrated in the figure 1. Indeed, when the mutation rate is too low genomes are extremely invariable regardless of their respective lengths, i.e., $\varphi \simeq 1$. Consequently, when the mutation rate is too low, genomes are likely to evolve too slowly. On the contrary when the mutation rate is too large genomes are extremely variable regardless of their respective lengths, i.e., $\varphi \simeq 0$. Consequently, when the mutation rate is too high, genomes are susceptible to evolve improperly because of drastic changes. Besides the previous effect, for intermediate mutation rates Figure 1 illustrates that the genome variability estimated by the mutation probability increases together with the genome length, longer genomes being more variable than shorter ones.

In order to tune properly the mutation rate, we consider a range of plausible genome sizes that individuals could grow in order to tackle the subspace-clustering problem. Let us take $|\Gamma_{min}| = 50$ as a minimal reasonable genome length (e.g., Γ_{min} can encode 10 clusters in subspaces having 5 dimensions or 5 clusters in subspaces having 10 dimensions and is a quite small clustering model). Let us take $|\Gamma_{max}| = 400$ as a maximal reasonable genome length (e.g., Γ_{max} can encode 20 clusters in subspaces having 20 dimensions in average). Γ_{max} is also the more variable genome we consider.

A sensitivity analysis performed in Section 4.3 show that the results quality are not substantially modified close to the mutation rates range defined previously. However mutation rates chosen far outside the given range lead to poorer results.

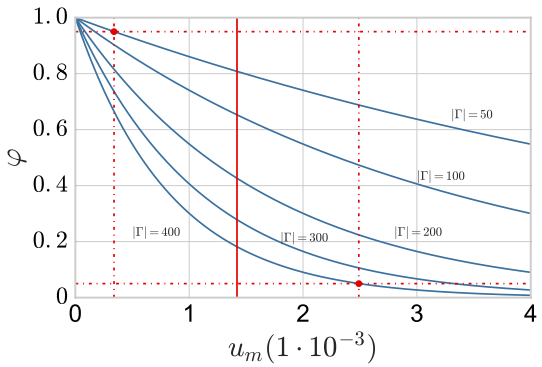


Figure 1: φ value computed as a function of the mutation rate u_m for different genome sizes. The suitable chosen range of genomic variability and its related mutation rate range are delimited by dashed lines. The retained mutation rate is marked by a vertical plain line.

A suitable range of mutation rates should allow the less variable genomes to evolve fast enough and should not lead the more variable genomes to jump too far in the genomes space. We decided to work with mutation rates that allow Γ_{min} to have at most 95% of chances to avoid mutations and Γ_{max} to have at least 5% of chances to avoid mutations. From the expression of φ, we have $u_m = 1 - \varphi^{\frac{1}{3\times|\Gamma|}}$, and thus $u_{max} = 1 - 0.05^{\frac{1}{3\times|\Gamma_{max}|}} \approx 0.00249$ and $u_{min} = 1 - 0.95^{\frac{1}{3\times|\Gamma_{min}|}} \approx 0.00034$. Let us set the mutation rate to $u_m = \frac{u_{min}+u_{max}}{2} \approx 0.00142$

Datasets used for empirical setting of the other parameters In order to adjust the remaining parameters we decided to analyze fitness convergence curves on three typical datasets. ChameleoClust$^+$ parameters were not adjusted using any external evaluation (e.g., comparison to a "true" labeling of reference), but only using the fitness measure (internal to the algorithm). We have decided to use *shape* and *pendigits*, two real world datasets and *D20* a synthetic dataset. Both real world datasets have enough dimensions and identified classes, *shape* is the smallest real world dataset of the framework (160 objects) and *pendigits* is the largest world dataset of the framework (7494 objects). We have decided to use the synthetic dataset *D20* because it has enough dimensions and dataset objects and because it is somehow representative of the synthetic datasets. Indeed the four datasets (*N10*, *N30*, *N50* and *N70*) used by [11] to assess the capacities to deal with noise points were build from *D20*, the datasets generated to evaluate the scalability with respect to the dataset size have all 20 dimensions and 10% of noise points as *D20*.

Population size Figure 2 illustrates that the larger the population is, the higher the fitness values are. Indeed a larger population has a higher exploration power, and is more likely to reach optimal solutions. However these improvements reach a plateau and tend to be less significant. Figure 2 illustrates that an appropriate fitness convergence is reached with a population size set to $N = 300$.

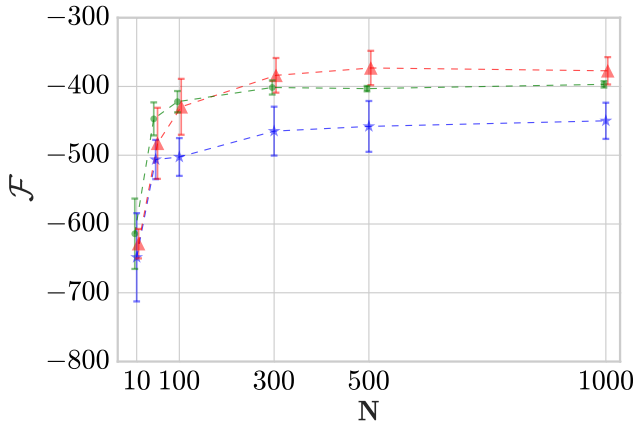
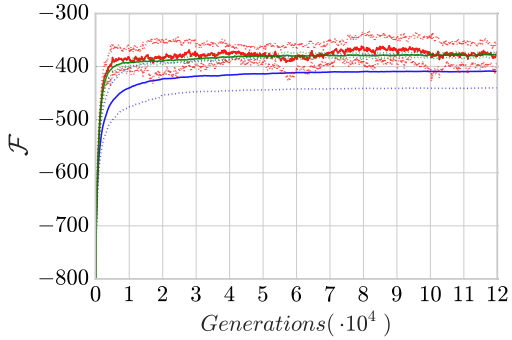


Figure 2: Mean fitness values ± standard deviation for the best individual of the last generation for each one of the 10 runs on *shape* (red), *pendigits* (blue) and *D20* (green) as a function of the population size.

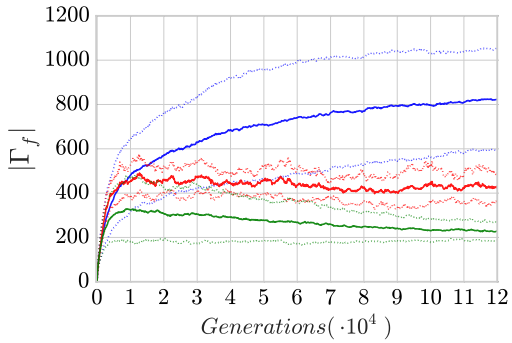
Number of generations At 5000 generations the algorithm achieved a good convergence for fitness as illustrated in Figure 3a, where this convergence seems complete for *shape* and *D20* datasets, and nearly complete for the *pendigits* dataset. A careful setting of the number of generations is not required before performing the subspace clustering, because the user can monitor the fitness curve during the process in order to stop it when the fitness convergence reaches a plateau. However, as detecting such plateaux is somewhat subjective, here we decided to evaluate ChameleoClust$^+$ with an early stopping at 5000 generations for all the experiments. Notice that, as could be expected and as shown by the sensitivity analysis carried out in Section 4.3, allowing the algorithm to evolve during more generations does not have a negative impact on the clustering quality and can still slightly improve it.

Figures 3b and 3c illustrate that the early generations are characterized by a fast evolution of the genome structure, and particularly of the number of functional tuples in the genome and the fraction of functional tuples in the genome. At 5000 generations the algorithm has already been able to take advantage of the genome structure evolution, and particularly of the evolution of the functional tuples of the genome, which are directly related to the subspace clustering model encoded by the individual. Readers may notice that the convergence of the genome structure may be slower than the fitness convergence. However the main point with regard to the subspace clustering problem is to obtain well positioned core points (i.e., to have an optimized phenotype), and consequently it is not necessary to run the algorithm until a stable genome structure is reached, but the algorithm can be stopped earlier, as soon as a stable fitness is obtained (Figure 3a).

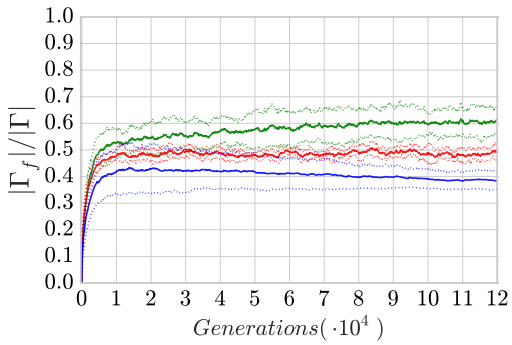
Maximal number of subspace clusters c_{max} is the maximal number of subspace clusters that are built, and is the only parameter that required to be tuned, since for all the other parameters the same setting was used for **all** datasets. Moreover, this parameter does not require a fine tuning since ChameleoClust$^+$ adapts the number of subspace clusters between 1 and c_{max}.



(a) Evolution of the fitness values.



(b) Evolution of the number functional tuples.



(c) Evolution of the percentage of functional tuples.

Figure 3: Evolution of the mean ± standard deviation of different measures for the best individuals for 10 runs over the real world datasets *shape* (red) and *pendigits* (blue) and the synthetic dataset *D20* (green).

In order to set this parameter we first executed ChameleoClust$^+$ with $c_{max} = 10$. When the algorithm outputs exactly c_{max} clusters, this means that the algorithm is likely to have been limited by a too low value set for c_{max}. In this case, the clustering was repeated with increasing values of c_{max}, with an increment of 10, until ChameleoClust$^+$ output less than c_{max} clusters. Only the last value of c_{max} is retained, allowing then ChameleoClust$^+$ to regulate the number of clusters built. Using this procedure, for the real world datasets the c_{max} parameter was set to 10 for *breast* and *glass*, to 20 for *shape* and *pendigits*, to 30 for *liver* and *diabetes* and finally to 40 for *vowel*. For the synthetic datasets, the same procedure, leads to set c_{max} to 30 for *D05*, the dataset having 5 dimensions, and to 20 for the fifteen other datasets.

3.4 Evaluation measures

In order to compare our algorithm to the others, we used the same standard evaluation measures for clusters and subspace clusters as [11]: entropy, accuracy, F1, RNIA and CE (extension of *Clustering Error* to subspace clustering). We performed also the same simple transformation of entropy and RNIA, by computing $\overline{RNIA} = 1 - RNIA$ and $\overline{entropy} = 1 - entropy$ to have all evaluation measures ranging from 0 (low quality) to 1 (high quality). The three first measures (entropy, accuracy and F1) reflect how well objects that should have been grouped together were effectively grouped. The two last measures, RNIA and CE introduced in [1], take into account the way the objects are grouped and also relevancy of the subspaces found by the algorithm. For these measures, when the *true* dimensions of the subspace clusters are not known (for real datasets), then as in [11] all dimensions have been considered as relevant, but then the interpretation of these measures should remain cautions since the true sets of dimensions are likely to be smaller. Of course this does not apply to the synthetic datasets, since for them the reference clusters and their dimensions are known. We refer the reader to [11] for a detailed presentation of the evaluation measures.

4 Experimental results

4.1 Real dataset

We computed the minimum, the maximum and the mean of the evaluation measures over 10 standard runs of ChameleoClust$^+$ using the same parameter setting for all datasets as justified and given in Section 3.3, except of course for the parameter specifying the maximum number of clusters (c_{max}) that was tuned according to the simple procedure also given in Section 3.3. As explained in Section 3.1, these results are compared to the ones provided by [11], that represent the best possible outputs that could be produced by the main subspace clustering approaches over their respective parameter space. More precisely, for these other algorithms, on each real dataset only two outputs were retained: 1) the one computed for the parameter setting that maximizes the F_1 measure, and 2) the one obtained when maximizing the accuracy. These two outputs led in the result tables to two values for each measure, the smallest of the two being called best *min* and the other best *max*. For all datasets we also

Table 1: Results for the *shape* real dataset: 17 dimensions, 9 classes, 160 objects

	$F1$		*Accuracy*		CE		$\overline{RNIA}$		$\overline{Entropy}$		*Coverage*		$NumClusters$		$AvgDim$		*Runtime*	
	max	min	max	min	max	min	max	min	max	min	max	min	max	min	max	min	max	min
CLIQUE	0.31	0.31	**0.76**	**0.76**	0.01	0.01	0.07	0.07	0.66	0.66	**1.00**	**1.00**	486	486	**3.3**	**3.3**	**235**	**235**
DOC	0.90	0.83	**0.79**	**0.54**	**0.56**	**0.38**	0.90	0.82	0.93	0.86	**1.00**	**1.00**	53	29	**13.8**	**12.8**	2E+06	86500
MINECLUS	0.94	0.86	**0.79**	**0.60**	**0.58**	**0.46**	1.00	1.00	0.93	0.82	**1.00**	**1.00**	64	32	**17.0**	**17.0**	46703	3266
SCHISM	0.51	0.30	**0.74**	**0.49**	0.10	0.00	0.26	0.01	**0.85**	**0.55**	**1.00**	**0.92**	8835	90	**6.0**	**3.9**	712964	9031
SUBCLU	0.36	0.29	0.70	0.64	0.00	0.00	0.05	0.04	0.89	0.88	**1.00**	**1.00**	3468	3337	**4.5**	**4.1**	4063	1891
FIRES	0.36	0.36	0.51	0.44	0.20	0.13	0.25	0.20	0.88	0.82	0.45	0.39	**10**	**5**	**7.6**	**5.3**	63	47
INSCY	**0.84**	**0.59**	**0.76**	**0.48**	0.18	0.16	0.37	0.24	0.94	0.87	0.88	0.82	185	48	**9.8**	**9.5**	22578	11531
PROCLUS	0.84	0.81	**0.72**	**0.71**	0.25	0.18	0.61	0.37	0.93	0.91	0.89	0.79	34	34	**13.0**	**7.0**	**593**	**469**
P3C	0.51	0.51	0.61	0.61	0.14	0.14	0.17	0.17	0.80	0.80	0.66	0.66	**9**	**9**	**4.1**	**4.1**	140	140
STATPC	0.43	0.43	**0.74**	**0.74**	**0.45**	**0.45**	0.55	0.55	0.56	0.56	0.92	0.92	**9**	**9**	**17.0**	**17.0**	**250**	**171**
CHAMELEOCLUST$^+$	**0.75**	**0.63**	**0.80**	**0.71**	**0.54**	**0.49**	**0.78**	**0.71**	**0.77**	**0.67**	**1**	**1**	**14**	**10**	**12.40**	**10.79**	**462**	**252**
mean	0.68		0.75		0.52		0.76		0.72		1		12.0		11.72		339	

give the number of subspace clusters found, the average dimensionality of these clusters, and their coverage. The coverage is here the percentage of objects of the dataset that were associated to clusters, and could be less than 100%. This is the case for algorithms that identified some objects as outliers or as reflecting noise, and also for algorithms that were not able to identify a cluster for these objects. Finally even though ChameleoClust$^+$ has been executed on a computer (2.67GHz CPU) different from the one used by [11] (2.3GHz CPU), we report the runtimes, since at least their orders of magnitude can still be compared.

In order to illustrate the performances of ChameleoClust$^+$ we focus on dataset *shape* in Table 1. For the sake of completeness the results obtained on the other datasets are given in the Appendix. In Table 1, when an algorithm has a best possible run with a higher evaluation than ChameleoClust$^+$ the result is highlighted in grey, and if the evaluation is similar to ChameleoClust$^+$ then the result is simply emphasized in bold.

For *Accuracy* and *CE* ChameleoClust$^+$ (together with DOC and MINECLUS) has among the best results, while its parameters were not optimized using the class labels to maximize the *Accuracy*.

For F_1 and $\overline{RNIA}$ the best possible runs of DOC and MINECLUS are observed with better results than standard runs of ChameleoClust$^+$, but they tend to split the dataset in more clusters (same behavior also on the synthetic datasets) and have runtimes considerably higher than ChameleoClust$^+$. The best possible runs of PROCLUS achieve better results than ChameleoClust$^+$ for F_1, but their coverage falls to about 80% to 90% leaving an important part of the dataset outside of the clusters.

Looking at entropy many algorithms have best possible runs leading to a better $\overline{entropy}$ than ChameleoClust$^+$. However, in clustering tasks, the entropy cannot be interpreted regardless of the number of clusters, because usually the entropy quality measure tends to improve when the number of clusters increases. Indeed, by definition of the entropy measure, the best entropy is obtained for the extreme case where we have one cluster per object. ChameleoClust$^+$ and three other algorithms (FIRES, P3C, STATPC) are able to avoid the spreading of the data over too many clusters, but at the cost of a degradation of the entropy measure. Notice that among them, ChameleoClust$^+$ is the only one to obtain such a reasonable number of clusters with a 100% coverage.

Regulation of the subspace clustering The mutational operators defined on Section 2.4 and 2.6 allow the ChameleoClust$^+$ genome structure to evolve, reaching potentially different genome sizes and different percentages of functional tuples according to each dataset. This allows ChameleoClust$^+$ to adapt,

Table 2: Average number of clusters and average dimensionality per cluster found for each dataset

Dataset	*NumClusters*	*AvgDim*	$\|\Gamma\|$	$\|\Gamma_f\|$
breast	5.1	12.15	733.2	276.8
diabetes	25.1	3.85	453.9	181.2
glass	6.9	6.18	504.3	184.7
liver	24.3	2.06	172.6	98.8
pendigits	11.6	10.01	1093.9	379.6
shape	12.0	11.72	926.1	409.7
vowel	28.0	5.41	749.6	331.3

for each dataset, the amount of information encoded within its genome. In addition, the genotype-phenotype mapping, detailed in Section 2.5, permits ChameleoClust$^+$ to encode different number of clusters described in subspaces with different dimensionalities. Let us analyze more precisely to which extent ChameleoClust$^+$ takes advantage of these degrees of freedom.

Before describing the results obtained by ChameleoClust$^+$, it should be noticed that most of the time the number of classes within a dataset does not correspond to the number of clusters found by the algorithms. Indeed, there is no integrity requirement enforcing the objects of a class to be grouped as a single cluster in space, and consequently it is not surprising to obtain more clusters than classes. Moreover, in some cases, a few algorithms found a very large number of clusters (sometimes even more clusters than objects), this behavior being due to their ability to output overlapping clusters.

Table 2 summarize the average number of clusters, their average dimensionalities, the average genome length and the average number of functional tuples in the genome for each one of the seven real world datasets. The subspace clustering models produced by ChameleoClust$^+$ are very different for each dataset: the average number of clusters produced varies between 5.1 for *breast* dataset to 28.0 for *vowel* dataset and the average dimensionality of the subspaces found varies between 2.06 for *liver* to 12.15 for *breast*.

Similarly the average genome length varies from 172.6 for *liver* to 1093.9 for *pendigits* and the average number of functional tuples goes from 98.8 for *liver* up to 409.7 for *shape*. For all datasets, the number of clusters and the average dimensionalities of the subspaces found by ChameleoClust$^+$ are coherent with the number of clusters found by the other algorithms.

Broader comparison For almost every dataset, the performances of ChameleoClust$^+$ are competitive with respect to the best possible runs of the other algorithms. In order to compare ChameleoClust$^+$ and the state-of-the-art algorithms in a broader way we decided to focus on the real world datasets and to analyze the following evaluation measures: the coverage, the number of clusters found and different quality measures (F1, Accuracy, CE, RNIA and Entropy). For each real world dataset and each one of the measures presented above:

- We sorted the highest measures (column best max) obtained by the different algorithms and then we ranked the methods according to their measures.
- The same was performed with the lowest measures (column best min) for the different algorithms.

The different quality measures and the coverages were sorted in increasing order, whereas the runtimes and the number of clusters obtained were sorted in decreasing order. While the choice of an increasing sorting order for the evaluation measure and the choice of a decreasing order for the runtime is straightforward, the sorting order for the coverage and the number of clusters needs further explanations. We decided to sort the coverages in increasing order so the methods that build less representative models excluding too many points will have lower ranks. We also decided to sort the number of clusters in decreasing order, indeed, the fewer the clusters in the clustering model, the easier their interpretation, so methods building a few of easily interpretable clusters will be characterized by a higher rank. After having ranked the algorithms according to their highest and lowest measures:

- We compute the mean algorithm ranks related to their highest measure by averaging the respective ranks along the different datasets.
- We compute the mean algorithm ranks related to their lowest measure by averaging the respective ranks along the different datasets.
- We compute the mean algorithm rank by averaging the algorithm mean ranks related to their lowest and highest measures.

The average ranks of the different algorithms for the highest and the lowest results obtained for each one of the evaluation measures are illustrated in Figure 4 (colored dots). This figure also shows the average rank of each method (red stars). ChameleoClust$^+$ has the second best average ranking and is among the best ranked algorithms together with MINECLUS, DOC and PROCLUS. In addition ChameleoClust$^+$ ranks are not widely dispersed, which means that ChameleoClust$^+$ has also a good compromise between the different evaluation measures and is competitive with respect to the best results of the state-of-the-art algorithms.

In order to compare ChameleoClust$^+$, MINECLUS, DOC and PROCLUS, the best ranked algorithms, we decided to analyze more precisely the number of clusters they produce, their coverage and their runtimes. The table 3 summarize the number of datasets where the highest and lowest number of clusters found for each algorithm is interpretable (100 clusters or less), the number of datasets where the highest and lowest coverage of each algorithm is reasonable and the amount of excluded data points is not too high (coverage of at least 95%) and the number of datasets where the shortest and longest execution of each algorithm last for a reasonable time (one hour or less). We will focus on ChameleoClust$^+$, MINECLUS, DOC and PROCLUS results but the other algorithm results are also presented for the sake of completeness. ChameleoClust$^+$, MINECLUS, DOC and PROCLUS produced for each dataset an interpretable number of clusters, PROCLUS and DOC usually produced lower coverage in order to increase the results quality, finally MINECLUS and DOC had higher run times and last for more than one hour while processing different datasets. ChameleoClust$^+$ produces good quality results together with low runtime and high coverage.

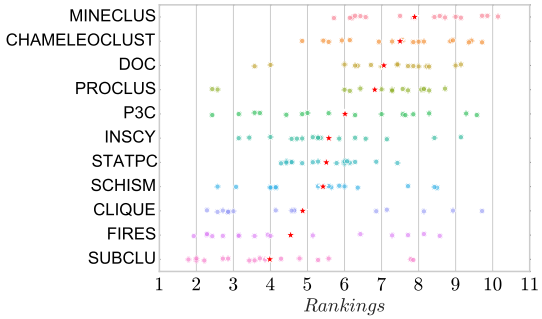


Figure 4: Mean over the different datasets of the ranking of each algorithm for the maximum and the minimum value obtained for each evaluation measure: Accuracy, Entropy, F1, CE, RNIA, Number of cluster, Coverage, Runtime (colored dots) and average ranking for each method (red stars).

Table 3: Number of datasets where the conditions on runtime (less than one hour), coverage (more than 0.95%) and number of clusters (less than 100) were fulfilled

Evaluation	MINECLUS	CHAMELEOCLUST	DOC	PROCLUS	P3C	INSCY	STATPC	SCHISM	CLIQUE	FIRES	SUBCLU
$max(NumClusters) \leq 100$	7	7	7	7	7	1	3	1	0	7	0
$min(NumClusters) \leq 100$	7	7	7	7	7	4	7	4	2	7	2
$max(Coverage) \geq 95\%$	7	7	4	0	3	1	3	7	7	0	6
$min(Coverage) \geq 95\%$	4	7	1	0	0	0	1	2	7	0	5
$max(Runtime) \leq 1h$	2	6	0	6	5	1	3	2	2	5	1
$min(Runtime) \leq 1h$	4	6	2	6	5	2	3	4	7	6	4

4.2 Synthetic data

ChameleoClust$^+$ was executed 10 times on each of the 16 synthetic datasets. For each dataset we kept the run reaching the highest fitness (for the best individual) among the 10 runs (notice that this selection is made without using any external labeling, but only the fitness values). Then for each evaluation measure, we plotted the measure value obtained with respect to the number of clusters found by each of the 16 selected runs. The results are shown in Figure 5. For each evaluation measure we also plotted in blue the shape of the area where the other algorithm results lay (as reported in [11]). Again for these other algorithms, their results correspond to there best runs over the parameter space. More precisely, for each quality measure, the results were collected as follows. For an algorithm and a given dataset the parameter space of the algorithm were explored, and using the external labeling, only the execution leading to the highest value of the measure has been retained. In the plots of the figure 5, good performances correspond to regions where the outputs contain about 10 clusters (the real number of clusters) and reach a high value for the quality measures. For almost every synthetic dataset the number of clusters found by ChameleoClust$^+$ is very close to the real number. ChameleoClust$^+$ always found between 6 and 25 clusters. As reported in [11] the other algorithms found between 5 and 50 clusters, excepted a few cases where much more clusters were found (up to more than several thousands). Most of the evaluation measures for ChameleoClust$^+$ are comparable to the ones reported in [11]. Keeping in mind that for the other algorithms only the best values of the evaluation measures over the parameter spaces were retained while the algorithm presented in this report

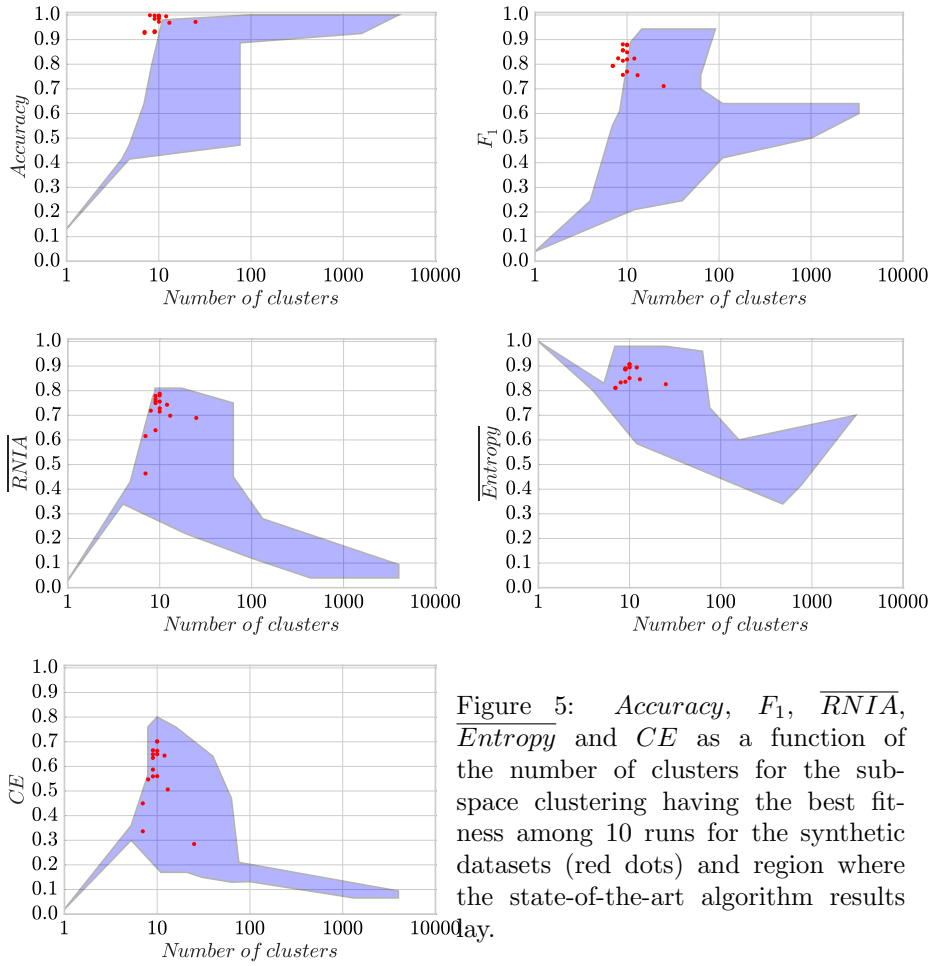


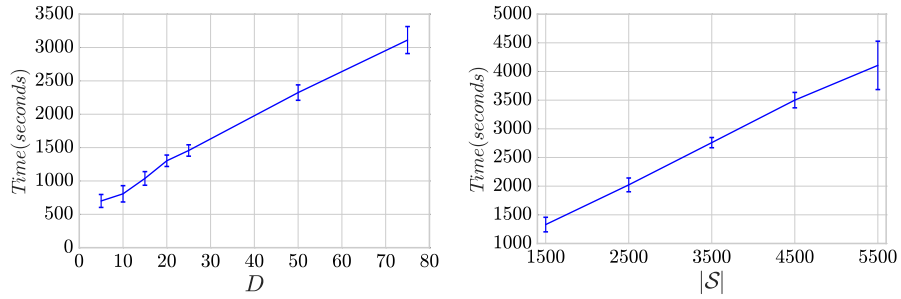
Figure 5: *Accuracy*, F_1, $\overline{RNIA}$, $\overline{Entropy}$ and CE as a function of the number of clusters for the subspace clustering having the best fitness among 10 runs for the synthetic datasets (red dots) and region where the state-of-the-art algorithm results lay.

uses only a same standard parameter setting for the evolutionary parameters, we could claim that ChameleoClust$^+$ realistic results are at east as good, or even better than the best results of the state-of-the-art algorithms.

In addition, we give Figure 6a and Figure 6b the runtime of ChameleoClust$^+$ with respect to the number of dimensions and the number of objects of the synthetic datasets. The corresponding curves show that the algorithm scales rather linearly in both cases.

4.3 Sensitivity analysis

In order to study the impact of the different parameters on the quality of the subspace clustering models obtained, a sensitivity analysis of the parameters has been carried out by varying the main parameters values one-at-a-time. For each parameter setting the execution was repeated 10 times and the average and standard deviation of the two main measures used for subspace clustering are given. As in Section 3.3, we consider the three representative datasets *shape*, *pendigits* and *D20* to carry out the sensitivity analysis. The parameters were



(a) Runtime vs. dimensionality of the dataset.

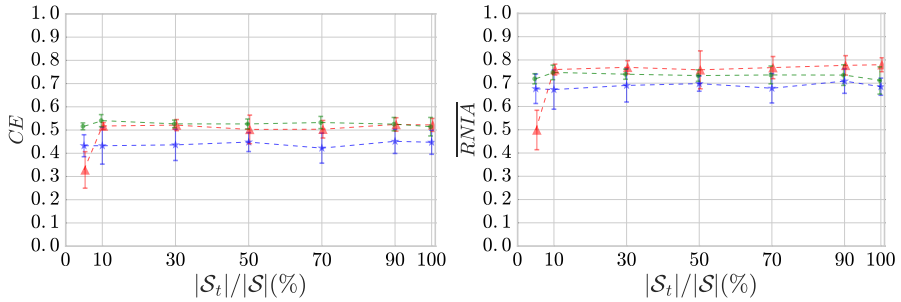
(b) Runtime vs. number of objects in the dataset.

Figure 6: Average ± standard deviation of the mean runtime of ChameleoClust$^+$ on each synthetic dataset.

set to the default values specified in Section 3.3,the sliding sample size $\mathcal{S}_\mathcal{F}$ was set to 10% of the dataset size, the selection pressure to $s = 0.5$, the initial genome size to $|\Gamma_{init}| = 200$ elements, the mutation rate to $u_m = 0.00142$, the population size to $N = 300$ individuals, we let ChameleoClust$^+$ running during 5000 generations and finally the maximal number of subspace clusters was set to $c_{max} = 20$ for the three datasets. The corresponding fitness curves for different population sizes and for number of generations have been provided in Figure 2 and 3a respectively. The fitness curves obtained modifying the remaining parameters are given for the sake of completeness in the Appendix as Figures 19a, 19b, 19c and 19d.

Sliding sample size The results obtained on the three datasets for sample sizes of 5%, 10%, 30%, 50%, 70%, 90% and 100% of the dataset size, are given in Figure 7a and Figure 7b. These curves show that the impact of the dataset sample size on the subspace cluster quality is low when the sliding sample used to compute the fitness is about 10% of the dataset size or more. As could be expected, using a small ratio on a small dataset leads to the most important degradations. This is the case for the smallest one, *shape*, that contains only 160 objects, and for which a 5% sample contains only 8 objects.

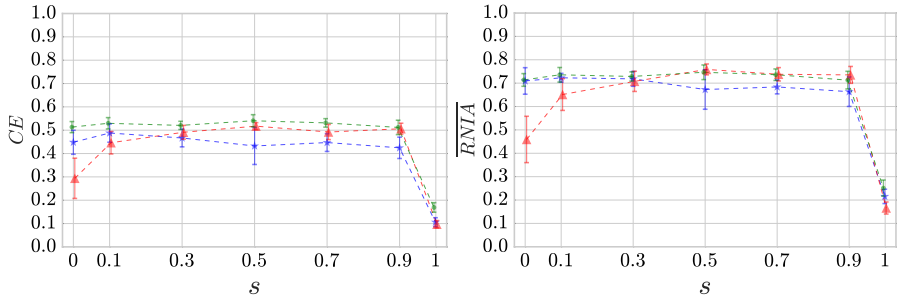
Selection pressure Figure 8a and Figure 8b present the results obtained when varying the selection pressure (values 0, 0.1, 0.3, 0.5, 0.7, 0.9 and 0.999). This change has a weak impact on the subspace cluster quality for s in $[0.1, \ldots, 0.9]$. This is not the case when the selection pressure is low ($s > 0.9$), according to Section 2.8 almost the same reproduction probabilities are assigned to each individual, and thus promising individuals have almost the same number of children as unadapted ones. Consequently, the algorithm allocates almost the same exploration resources to all individuals, and a degradation of the clustering quality is observed in the figures 8a and 8b. When the selection pressure is high ($s < 0.1$), almost the complete future generation comes from the best individual of the present generation (individual having a very high reproduction probability). In this case, the genetic variability within the new generation is reduced, and again Figure 8 shows a decrease of the cluster quality measures.



(a) CE vs. dataset sample size.

(b) RNIA vs. dataset sample size.

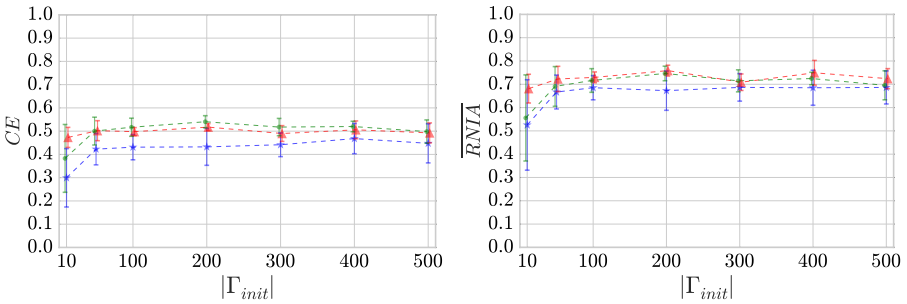
Figure 7: Mean ± standard deviation of quality measures for the best individual of the last generation for each one of the 10 runs on *shape* (red), *pendigits* (blue) and *D20* (green) as a function of the dataset sample size relative to the dataset size $\frac{|\mathcal{S}_t|}{|\mathcal{S}|}$ (percentage of the dataset size).



(a) CE vs. selection pressure.

(b) RNIA vs. selection pressure

Figure 8: Mean ± standard deviation of quality measures for the best individual of the last generation for each one of the 10 runs on *shape* (red), *pendigits* (blue) and *D20* (green) as a function of the selection pressure parameter s.



(a) CE vs. initial genome size.

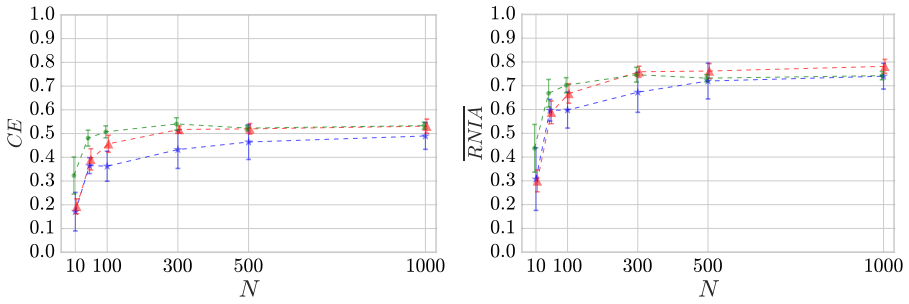
(b) RNIA vs. initial genome size.

Figure 9: Mean ± standard deviation of quality measures for the best individual of the last generation for each one of the 10 runs on *shape* (red), *pendigits* (blue) and *D20* (green) as a function of the initial genome size.

Initial genome length Set of values used: 10, 50, 100, 200, 300, 400, 500. As illustrated in Figure 9a and Figure 9b, the impact of the initial genome size is minor when the initial size is at least equal to 50. Indeed, ChameleoClust$^+$ genome size is evolvable and can be modified by large deletions and large duplications, consequently the initial size does not have a considerable impact on the algorithm quality. However we must note that small genomes have a higher probability to undergo replication without mutations and tend to evolve very slowly as it has been discussed in the paragraph dedicated to the mutation rate of the section 3.3. Consequently it is harder for genomes to reach suitable genome lengths and as evolution tends to be slower for smaller genomes, results tend to be poorer.

Population size Set of values used: 10, 50, 100, 300, 500, 1000. As illustrated in Figure 10a and Figure 10b the larger the population the better the results. Indeed smaller populations may only explore a small portion of the solution space and tend also to have a smaller genetic variability leading to poorer results. However increasing the population size tends also to increase the algorithm runtimes. In order to achieve some sort of trade-off between improving the exploration power through increasing the population size and keeping reasonable runtimes, we must notice that improvements induced by larger populations turn to be minor when the population is already large enough to have a proper degree of exploration of the solution domain.

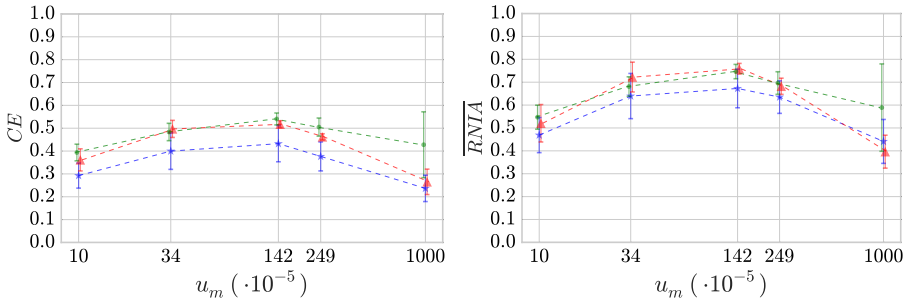
Mutation rate Set of values used: 0.0001, 0.00034, 0.00142, 0.00249, 0.01. We decided to test the mutation rates delimitating the suitable mutation rate range defined in Section 3.3 ($u_m = 0.00034$ and $u_m = 0.00249$), the default mutation rate ($u_m = 0.00142$) and two values outside the suitable mutation rate range ($u_m = 0.01$ and $u_m = 0.0001$). If the mutation rate is chosen inside the boundary defined in Section 3.3, it does not have a major impact on the subspace clusters quality whenever, as showed in Figure 11a. However, if we choose a mutation rate far outside the boundary, the subspace clusters quality decreases. On one hand when the mutation rate is too low, the evolution process will become very slow, as most of the individuals do not mutate at all, on the



(a) CE vs. population size.

(b) RNIA vs. population size.

Figure 10: Mean ± standard deviation of quality measures for the best individual of the last generation for each one of the 10 runs on *shape* (red), *pendigits* (blue) and *D20* (green) as a function of the population size N.



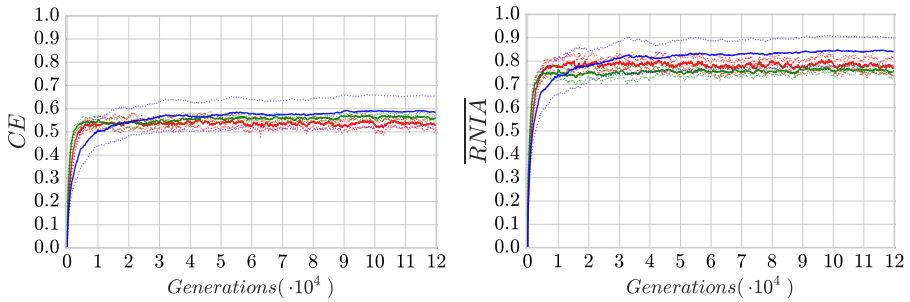
(a) CE vs. mutation rate.

(b) RNIA vs. mutation rate.

Figure 11: Mean ± standard deviation of quality measures for the best individual of the last generation for each one of the 10 runs on *shape* (red), *pendigits* (blue) and *D20* (green) as a function of the mutation rate u_m.

other hand, when the mutation rate is too high, the mutations are too brutal and it becomes harder for the organisms to converge towards a suitable subspace clustering.

Number of generations We have run ChameleoClust$^+$ 10 times for each chosen dataset over 120000 generations. The different evaluation measures were computed each 100 generations. As illustrated in Figures 12a and 12b, the more generations we let the algorithm evolve the better are the results. However the improvements tend to be less significant and results reach finally a plateau. Consequently we can conclude that it is not necessary to wait for the genome structure to converge to get good quality subspace clusters. Indeed the fitness seems to be a good enough witness to notice when accurate subspace clusters are reached. As discussed in the paragraph related to the number of generations of the section 3.3, the earlier generations are characterized by a fast evolution of the genome structure and of the subspace clusters quality. Well positioned core points are rapidly found, and it is not necessary to wait for too many generations



(a) Evolution of CE quality measure. (b) Evolution of RNIA quality measure.

Figure 12: Evolution of the mean ± standard deviation of quality measures for the best individual for 10 runs of ChameleoClust$^+$ for *shape* (red), *pendigits* (blue) and *D20* (green).

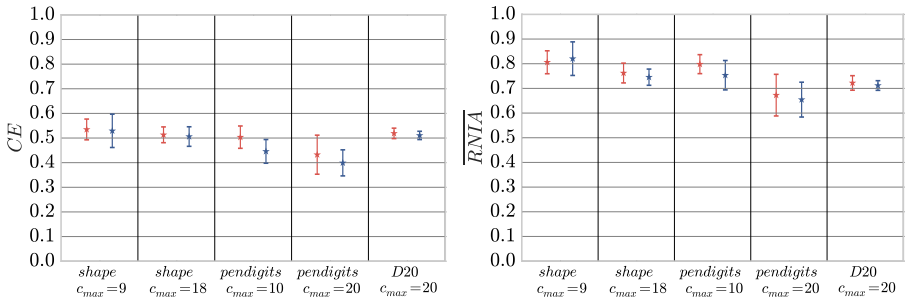
to get good results. However slightly better results can be achieved by allowing the algorithm to evolve during more generations.

4.4 Alternative models

Aside from its evolvable genome size driven by large duplications and deletions, ChameleoClust$^+$ is determined by two further strategic choices, its elitist reproduction method and mainly the presence of non-functional elements. In this section both strategic choices impacts are studied using the three datasets previously chosen, i.e., *pendigits*, *shape* and *D20*. We have decided to carry out the comparison between the different choices under the best possible conditions regarding the free parameter c_{max}, i.e., setting it equal to real number of groups for the synthetic dataset ($c_{max} = 10$ for *D20*), and to the number of classes ($c_{max} = 9$ for *shape* and $c_{max} = 10$ for *pendigits*) and also to twice the number of classes for the real world datasets ($c_{max} = 18$ for *shape* and $c_{max} = 20$ for *pendigits*), as the real number of groups is not necessarily equal to the number of classes. In both cases we will focus on the RNIA and CE quality measures in order to analyze the impacts of the choices (fitness curves are provided in the Appendix as Figure 20b).

Elitism In order to study the impact of an elitist reproduction method, we executed ChameleoClust$^+$ 10 times for each dataset and each value of c_{max}. One serie of runs was carried out using the usual parameters setting while the other serie of runs was obtained switching off elitism. Figures 13a and 13b illustrate respectively the impact of elitism on CE and RNIA in the conditions detailed above. In most of the cases, the different quality measures increased slightly when elitism was incorporated to the algorithm. Elitism ensures that the best subspace clustering found in the present generation will not be lost during reproduction. Even though its effect is not shown to be very important on the figures, it seems to have a slightly positive interest here.

Non-functional elements In order to study the impact of non-functional elements, we ran ChameleoClust$^+$ 10 times for each dataset and each value



(a) Impact of elitism on CE.

(b) Impact of elitism on RNIA.

Figure 13: Mean ± standard deviation of quality measures for 10 runs on *shape*, *pendigits* and *D20*, with (red) and without (blue) elitism. For each real world dataset two c_{max} values where tested: the number of classes in the dataset and twice this number and the real number of cluster was used as c_{max} value for the synthetic dataset.

of c_{max}. One serie of runs has been achieved using the standard parameter setting previously defined while the second serie has been achieved avoiding the existence of non-functional elements (i.e. $\forall \gamma_i = \langle g_i, c_i, d_i, x_i \rangle \in \Gamma,\ g_i = 1$). Figures 14a and 14b illustrate respectively the impact of non-functional elements on CE and RNIA (the associated fitness curves are provided in the Appendix as Figure 20a). In most of the cases, the different quality measures increased considerably when non-functional elements were incorporated to the algorithm.

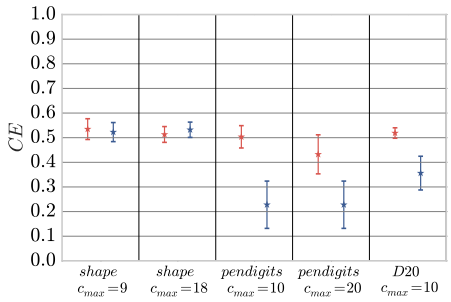
5 Real world stream analysis

The experiments on static datasets were completed with experiments on a dynamic data stream, within task 5.1 of the EvoEvo project (Document DOW p.33), leading to the Evowave demonstrator introduced in section 6.3.1 of the mid-term dissemination report (Deliverable 6.5). Even though the problem addressed is still a subspace-clustering problem, new difficulties are added by the context of a dynamic stream:

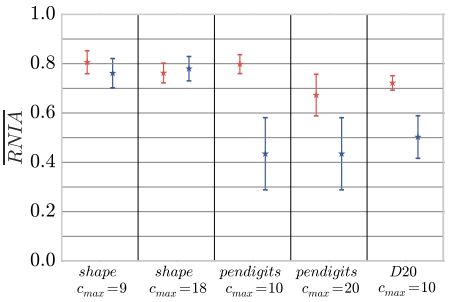
1. Number of classes may change over time.
2. The location of the classes is also susceptible to change over time.
3. Descriptors of the incoming objects are also susceptible to change (i.e., features can appear or disappear).

The goal of these experiments is to assess the efficiency of the EvoEvo approach to tackle these new difficulties.

The real-world data stream used for these experiments is the wifi environment in which a micro-computer is immersed, i.e., the strength of the signal from every wifi antenna in the neighborhood. This environment depends especially on available routers and other computers, so it is linked to the context of use of the computer : work, teaching, house, etc.



(a) Impact of non-functional tuples on CE.



(b) Impact of non-functional tuples on RNIA.

Figure 14: Mean ± standard deviation of quality measures for 10 runs on *shape*, *pendigits* and *D20*, with (red) and without (blue) non-functional tuples. For each real world dataset two c_{max} values where tested: the number of classes in the dataset and twice this number and the real number of cluster was used as c_{max} value for the synthetic dataset.

If the wifi signals from different contexts are dissimilar enough from each other, we expect that ChameleoClust$^+$ should be able to discriminate different contexts from the data. This corresponds to a dynamic stream problem as new classes, i.e., context of use of the computers, are always susceptible to appear/disappear, and the present wifi antennas are never the same in different contexts (features appearing/disappearing). This example is also challenging regarding the high dimensionality and noise level of the data. A precise description of the experimental setup is given in section 5.1, and results of the experiments are given and discussed in section 5.2.

This report is sided by the Evowave package, which combines several materials. It includes anonymized data and necessary materials to repeat the experiments, as well as code to run new experiments. Installation instructions and requirements are given in section 5.3.

5.1 Workflow

The experimental setup is split in four steps :

- Acquisition of data
- Preprocessing
- Clustering using ChameleoClust$^+$
- Visualization of the resulting clusters and comparison with ground-truth

This section details each part of the workflow.

Acquisition

Principle of acquisition Every wifi-card is permanently listening for input communications from any wifi device. Hardware and software are then used to filter/accept these communications. However, every communications attempt that reaches the wifi-card can be recorded and their characteristics (e.g., MAC address of the source) can be obtained via specific software tools. The **tshark** utility is one of these. It is free and open-source piece of software distributed as a command-line tool, and allows to collect during a given period of time information about messages reaching the wifi-card of a computer. Especially, it allows to identify the sender of the message, via its MAC address, which is an unique identifier of network interface of the sender, and the intensity of the signal received. These are the two characteristics used in our experiments. Indeed, we used an approximation of the distance between the computer and the sender, estimated from the signal intensity. The signal intensity gives information about the distance between the computer used and the sender, because the more distant is the computer from a sender, the lower is the intensity of the received signal. These distance can be estimated using the classical Friis transmission equation.

The **tshark** utility handles only the raw part of the acquisitions of the signals. The scheduling of the acquisitions and the distance estimation is carried out by a software component developed within the EvoEvo project (a Python program called **WifiCapture.py**). This program contains both an interface to the **tshark** software and the necessary code to perform the Friis computation. Installation details and examples of commands are given in section 5.3.

Dataset The data stream is made of data collected between January 2015 and June 2016. These acquisition were performed using the same computer in 5 different places (4 different buildings of the University and a private house), each places being visited many times. This places are used as ground-truth classes to evaluate the output of our stream analysis using ChameleoClust$^+$ to identify similar wifi context in the stream. At a finer evaluation grain, these classes are split into sub-classes corresponding to different rooms in the same building. Here the wifi signals measured for sub-classes of the same class are likely to be very similar, and this is used to assess the ability of the algo to differentiate between very close groups. A few acquisitions were also performed in other locations so as to introduce noise/outliers in the data. These acquisition are tag under a specific label *outliers* in the ground-truth.

The duration of each acquisition is 30 minutes, alternating between 1-minute periods of scan to collect wifi signal intensity and 2-minutes periods of pause. The result of each of these 1-minute periods is the set of the MAC addresses of the wifi devices that communicate at some moment in this period. To each MAC address is also associated a list of values. For an address XX, these values represent the estimated distances of the source XX computed for each intensity measured for the communication/message coming from XX during the period. The acquisition were repeated from January 2015 to June 2016 several times per week. The results is is a set of file containing information about the time and location of the acquisition and MAC address/list of distances pairs.

Preprocessing The main goal of our preprocessing step is to reduce the dimensionality and to anonymized the data. Indeed, in this setup each MAC

address identify a dimension, and the distance between wifi-cards is the value associated to this dimension. The length of MAC addresses (12 hexadecimals numbers), thus the raw dimensionality of the full space is $16^12 = 3e10^14$. Even though not every possible mac address is present in our collect, tens of thousands of them appear. Our solution, detailed below, implies both filtering out some dimensions and arbitrarily gathering some of the remaining solutions (projecting the whole space over no more than 256 dimensions).

First, each MAC address from which we have received less than two distance measures per second are considered not very active sources, thus not representative of the environment, and filtered out. Then, for each 1-minute scan, for each MAC address the mean of the distance measures estimated during this period is computed. Data for a 1-minute scan is now only a list of rather active MAC addresses associated with an average distance measure. The dimensions are then gathered. Only the last byte of each MAC address is kept and used as an identifier of the dimension. If two or more than two MAC addresses share the same last byte value, the corresponding distance measure is the mean of their distances. Given that only the last byte of the MAC address are kept, the number of dimensions is at most $D = 256$. These two steps are performed using the program **WifiLog.py**, developed within the project. The script **csv2h5.py** is then used to format data for the next steps.

For this application, distances should be considered more in term of relative changes rather than in term of absolute changes. Indeed, a change of distance from 1 meter to 10 meters away from our device is more important than a change from 101 meters to 110 meters. Therefore the next transformation made is to take the logarithm base-10 of the feature values, rather than the raw feature values themselves.

Intuitively the wifi environment is likely to be highly determined to the near surrounding rather than by very distant sources. So small distance are more important to identify a context. In ChameleoClust$^+$, it is the opposite, large feature values are considered as being more important than smaller ones (see section 2.7). Thus the following computation is operated to reverse the scale of the values: Let H be the set of the maximal distances observed in each dimension. Let x_{max} denote the median of the base-10 logarithm of the elements in H. Each feature value x is then replaced by $z = x_{max} - x$.

Finally, if a record does not contain a value for a given dimension, we consider that the value is equal to zero, leading to a feature value $z = x_{max} - 0$. It is equivalent to consider the corresponding antenna to be very far from the computer. These lasts steps and appropriate formatting operations for ChameleoClust$^+$ are performed by the **preprocessing_logMax-log.py** program. Examples of use of each program are given in section 5.3.

Visualization tool In order to achieve a better analysis of the performance of ChameleoClust$^+$ on dynamic streams such as the one described above, we have developed a set of visualization functions. This visualization tool includes a contingency table that depicts the cluster membership of the data points in the current sliding window with respect to their class-membership in the ground-truth. This tool also enables to track the evolution of the location of the core-points (center of the clusters) produced by the algorithm. The visualization

tool has been used to record videos of the execution of ChameleoClust$^+$ (videos provided in the Evowave package).

The central element in the visualization tool is a contingency matrix that illustrates the cluster membership of the sliding window data points with respect to their class-membership. The rows of the contingency table correspond to the class-membership of the data points and the columns correspond to the cluster-membership. The contingency table has c_{max} columns (ChameleoClust$^+$ parameter), one column for each of the core-point identifiers that could be used by ChameleoClust$^+$ to describe clusters. Empty columns are associated to core-points that are not encoded in the genome or to core-points encoded in the genome that describe no cluster (no data points associated to this core-point).

Aligned with each column, we represent the coordinates of the corresponding core-point in the 256-dimensional space in a radar chart diagram. Half of the radar chart are placed above and half below the contengency matrix. Here, a radar chart diagram represent one core-point, and consists of 256 equiangular radii, each radius representing the location of this core-point along one of the 256 dimensions. For each core-point we also represent as black radii in this report (and in white in the video) the median location of the data points associated to the corresponding core-point, and the thickness of the beam is used to denote the number of points associated to the location. Moreover we represent two circles in the diagrams, the inner circle corresponds to a zero coordinate value and the outer circle corresponds to the median of the highest values along all dimensions. Using this visualization tool it is possible to distinguish the subspace of each core-point, the coordinates of each core-point along each dimension and the median location of the data points associated to each core-point.

Figure 15 illustrates a typical output of the visualization tool. In this example 5 classes are present in the sliding window: The classes 2.A, 3.A, 3.B, 4.B, 4.C and 5. These classes are represented by 6 clusters: cluster 0, cluster 3, cluster 4, cluster 5, cluster 7 and cluster 8. Class 3.A is mainly described by cluster 5, class 3.B is mainly split in two clusters: cluster 3 and cluster 8, class 5 is described by cluster 7, class 2.A by cluster 4 (only one point). The remaining core-points (1, 2, 6 and 9) are encoded in the genome but denote no cluster. For example the cluster 2 exists in a two dimensional subspace (2 beams in its radar chart) but contains no data point (as can be seen in the column elow in the contengency table) and thus does not denote a cluster. The radar chart diagrams of the different clusters illustrates the fact that each cluster has potentially a different subspace. For example cluster 7 exists in a one dimensional subspace while cluster 3 or cluster 5 are described in higher dimensional subspaces.

5.2 Results

We executed 5 independent runs of ChameleoClust$^+$ on the data stream built as described in the previous section, and containing 2951 objects. During a run we used a sliding windows, and each 10 generations the oldest object in the sliding sample was replaced by the next in the stream (First-In-First-Out method). Before updating the sliding sample, different measures describing the genotype and the model of the best individual were stored and several evaluation measures were computed (F_1, *Accuracy*, CE and *entropy*). Given that the clusters subspaces were not known beforehand, we could not rely on

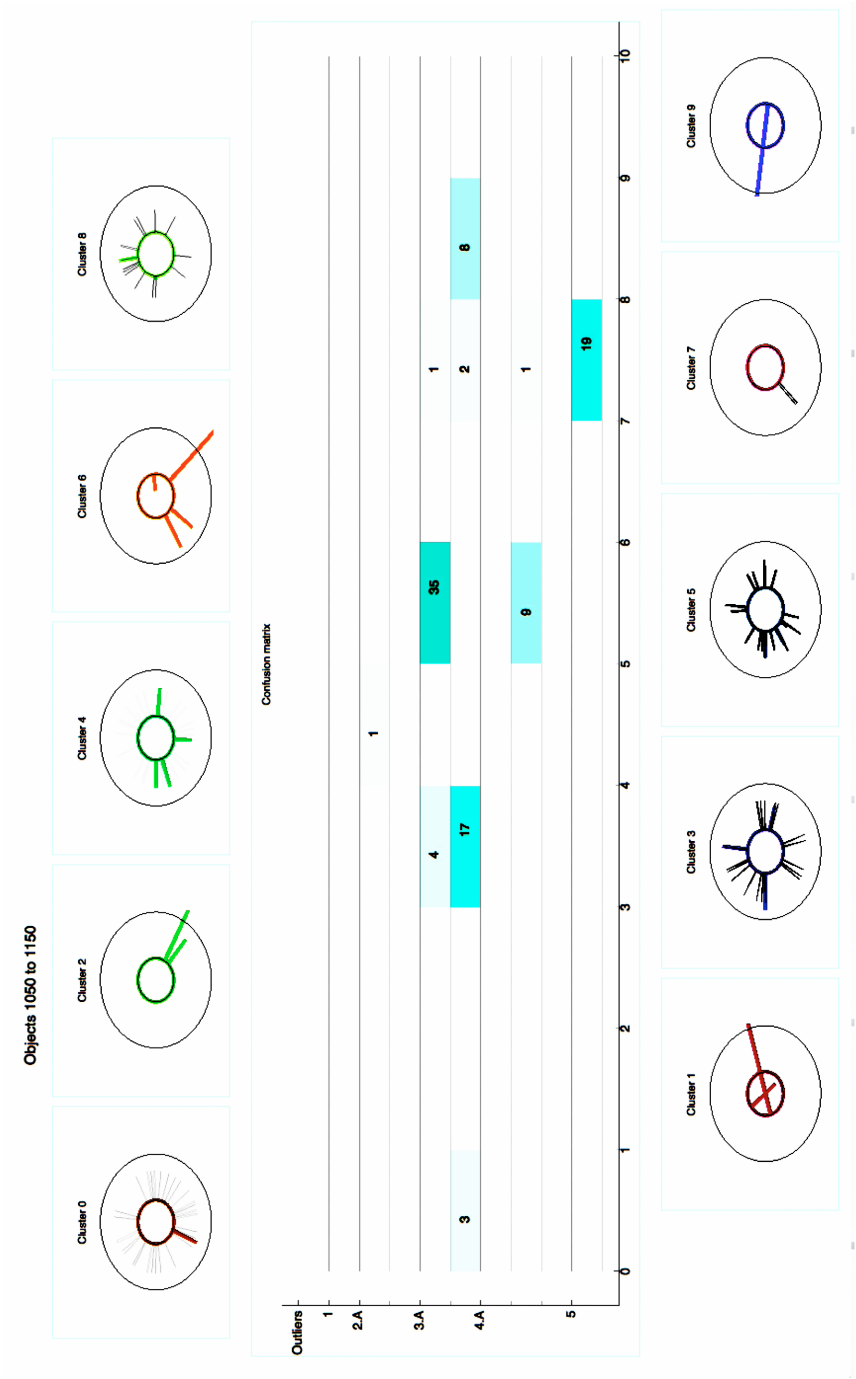


Figure 15: Illustration of the visualization tool output.

subspace clustering evaluation measure ($RNIA$ and subspace version of CE) for this study.

Experimental results obtained with the default parameter setting of ChameleoClust$^+$ presented Section 3.3 The average of different evaluation measures (*Accuracy*, F_1, *Entropy* and CE), the mean number of clusters, the mean dimensionality of the clusters and the mean number of genes are represented in Figure 16 as functions of the location of the sliding sample along the data stream (red curves). According to Figure 16, the genome structure of the organisms adapts to the changes in the data stream. The same applies to the best individual subspace clustering model, indeed the number of clusters and the cluster dimensionalities evolve along the data stream. Despite the changes in the data stream, the quality of the subspace clusters produced by the individuals tends to remain interesting. The algorithm seems to be adaptable enough to cope with the changes in the data stream by adapting the subspace clustering models encoded in the genomes.

Figure 17 illustrates three snapshots of the execution of the algorithm at different positions in the stream, using the visualization tool introduced in the previous section. Notice that the genome structure contains far more genes than those needed to encode the model. This can be seen especially, in the two first snapshots (top and middle) of Figure 17 (at positions 970-1070 and 1010-1110 in the stream) where only two core-points denote clusters (clusters 5 and 7). The other core-points correspond to no data point, even though they have high dimensionalities and many genes involved in their description. Such core-points appear to be inappropriate while dealing with such a dynamic environment.

ChameleoClust$^+$ cluster evolution Following the dynamics of the core-points of the best individuals along evolution, we notice that core-points that manage to capture data points have usually small subspaces. Then, the subspace dimensionalities of these core-points increase little by little when obtaining a coordinate along a new dimension can offer a selective advantage (a better fitness). However, while the data stream changes, it may happened that no more points in the current sample are associated to one of these core-points. Then this core-point stops being under selection pressure and is free to drift randomly. It can accumulate more and more changes and drifts around randomly. Increasing the dimensionality of such an empty core-point can turn it to be too specific to catch any data point further in the stream. This can be seen for core-point 3 in Figure 17 with the snapshot 970-1070 and the snapshot 1010-1110. Hopefully, when the number of empty core-points increases, there are more chances that big deletions remove genes associated to core-points that are not directly responsible for the good fitness. This can remove many dimensions of a core-point that is associated to no data point, and then can lead to its reuse to form a non-empty clusters. That this has been the case for core-point 3 in Figure 17 between the snapshot 1010-1110 and the snapshot 1050-1150.

phenotype and individuals are more likely to be able to reuse core-points that acquired a small subspace. Indeed, in Figure 17 we can see that the genomes underwent big deletions between the snapshot 1010-1110 and the snapshot 1050-1150, that allowed them to reuse core-points 2 and 3 to form non-empty clusters. Moreover core-points 5 and 9 have only a few dimensions and

could be reused easily. A video of the execution of the algorithm with the parameter settings established in Section 3.3 and using a seed equal to zero for the pseudo-random numbers generator is provided in the Evowave package (**EvoWave_run_default_params.mov**).

Notice that the genome of the best individual accumulates elements encoding empty core-points (for points 970-1070, 1010-1110 in the stream), such core-points become very specific and unlikely to describe clusters. However after undergoing big deletions the genome is able to reduce empty core-points dimensionalities so this core-points can describe clusters.

Direction for future improvements A possible way to limit the accumulation of many dimensions in empty (useless) clusters is to limit the promotion of non-functional elements to functional ones. In order to have some preliminary evidences of the effect of such a modification, we run ChameleoClust$^+$ 5 times while reducing the probability of transition from non-functional elements to functional ones. Let $\gamma_i \in \Gamma$ of the form $\gamma_i = \langle g, c, d, x \rangle$, denotes an element uniformly drawn in the genome and let $k \in \{1, 2, 3, 4\}$ a value chosen uniformly. In ChameleoClust$^+$, the point substitution operator modifies the k-th element of the tuple γ_i and replaces it with a new random number drawn uniformly in its associated range. For $k \in \{2, 3, 4\}$ we still use the definition made in Section 2.6, but here we redefine it when $k = 1$. In the definition in Section 2.6, the new value of g should be drawn uniformly in $\{0, 1\}$, now we change it as follows. If the current value of g is 1, then as before its new value is drawn from $\{0, 1\}$ uniformly, but if the current value of g is 0, then it is replaced by a 1 with a probability of 0.1 and stay unchanged with a probability of 0.9. Over the five runs, we computed again the average of different evaluation measures, the mean number of clusters, the mean dimensionality of the clusters and the mean number of genes. The results are presented in Figure 16 as a function of the location of the sliding sample in the data stream using green curves. It seems that the quality measures tend to be higher most of the time along the stream when compared to the previous results in red. This can suggest that the individuals are now able to adapt more quickly to the changes in the stream because they have less core-points corresponding to empty clusters with high dimensionalities. This change in the number and dimensionality of the core-points is supported by Figure 18 that gives the three new snapshots obtained at the same locations in the stream as the ones in Figure 17. A video of the execution with the modified probability for the mutation of g (but same parameters, and random seed also set to 0) is provided in the Evowave package (**EvoWave_run_modified_params.mov**). Further promising work will be to study more deeply this modification of the probability of promotion from non-functional elements to functional ones, and in particular to investigate its relationship to the speed of the changes that occur in the data stream itself.

5.3 Software package

We provide a package for ChameleoClust$^+$ and Evowave containing the programs and the data, that enables the replication of the experiments and the realization of new ones. Except for the free program **tshark**, all programs and scripts were developed within the EvoEvo project. To run new experiments from scratch, the user will have to install its own version of **tshark**.

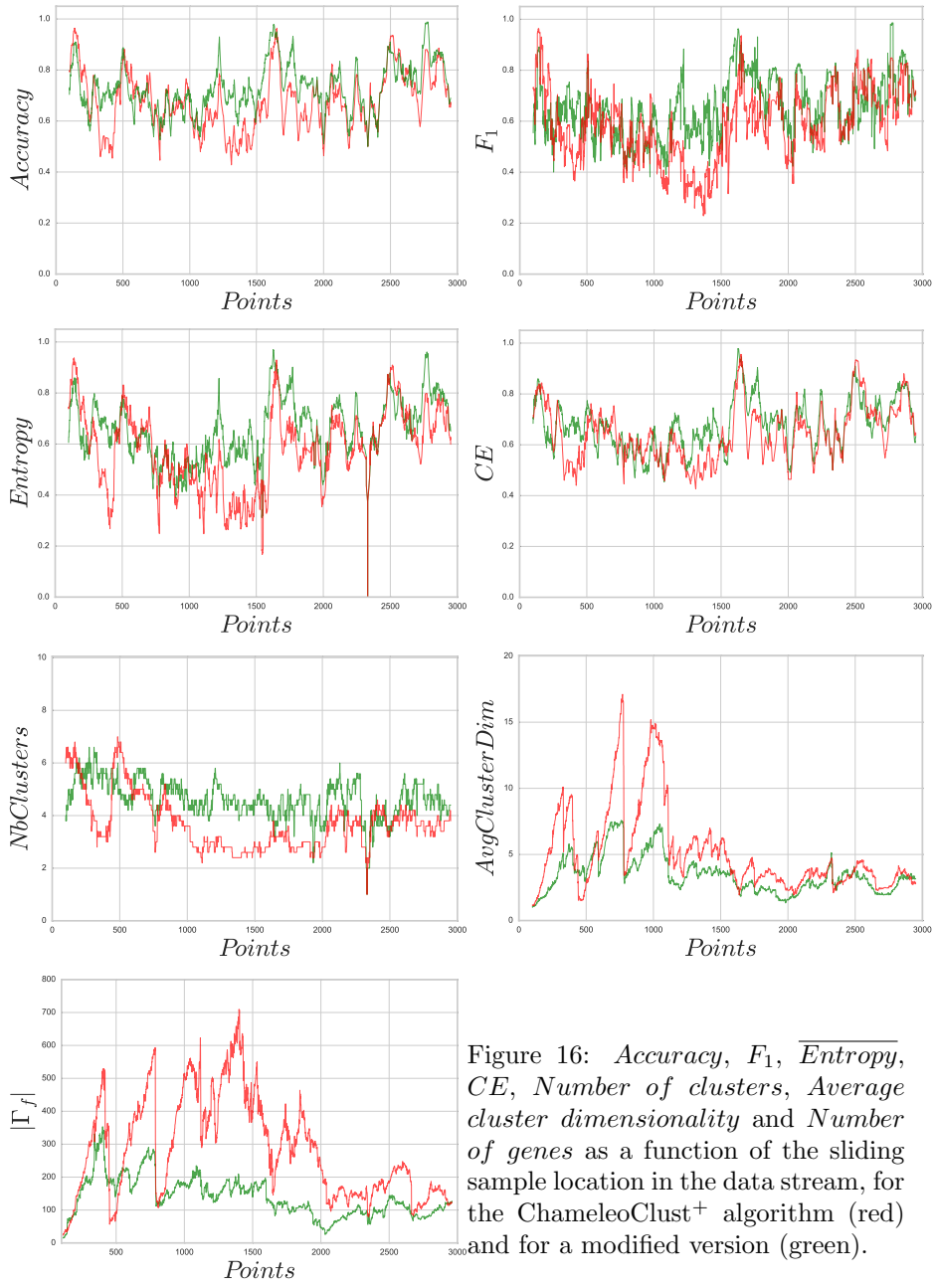


Figure 16: *Accuracy*, F_1, $\overline{Entropy}$, *CE*, *Number of clusters*, *Average cluster dimensionality* and *Number of genes* as a function of the sliding sample location in the data stream, for the ChameleoClust$^+$ algorithm (red) and for a modified version (green).

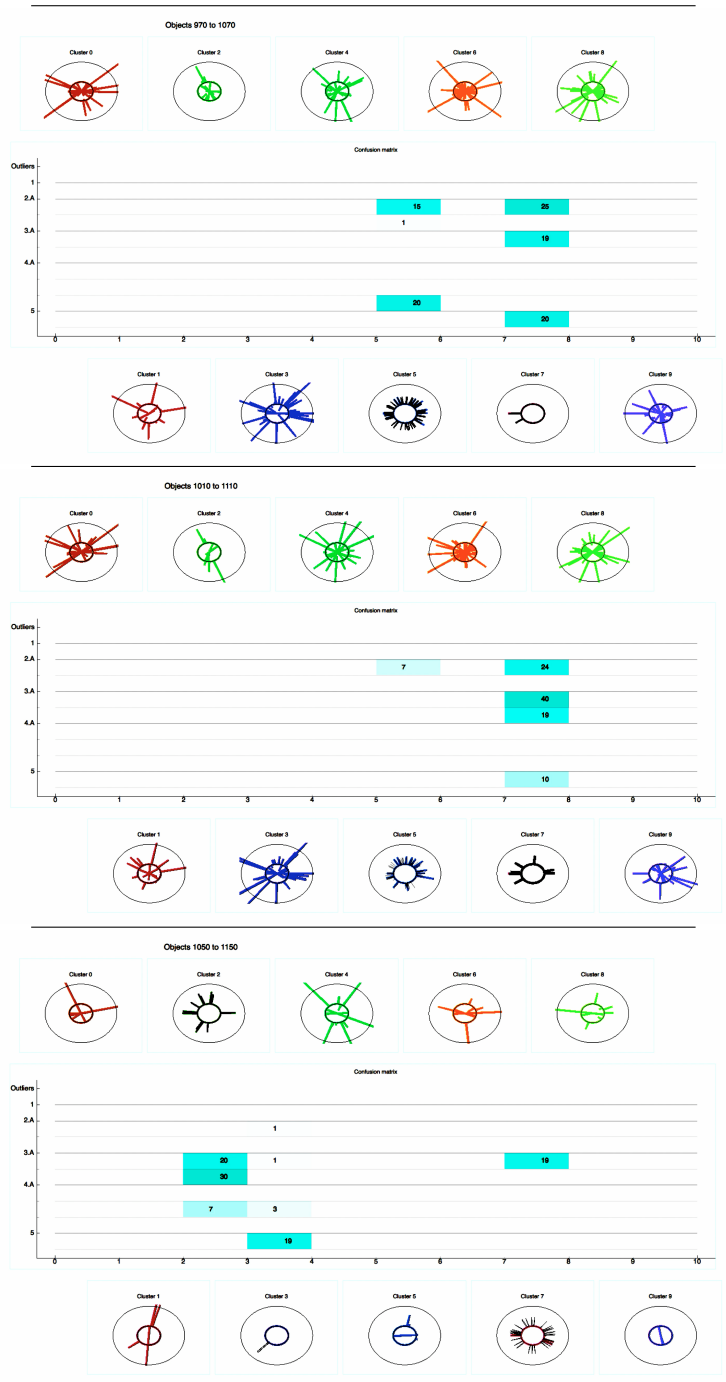


Figure 17: Snapshots of the program execution respectively for points 970-1070, 1010-1110 and 1050-1150 of the data stream.

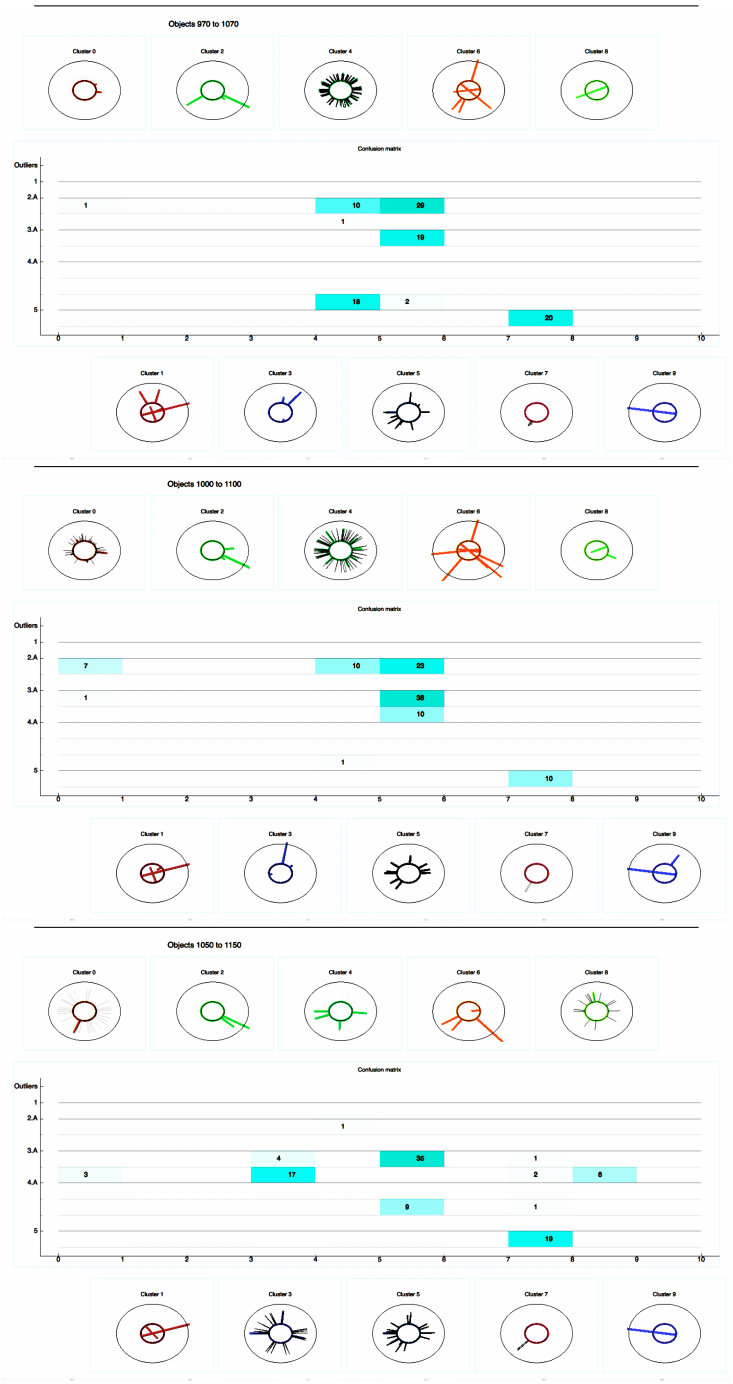


Figure 18: Snapshots of the execution of a modified version of ChameleoClust^{+} respectively for points 970-1070, 1010-1110 and 1050-1150 of the data stream.

Implementation of ChameleoClust$^+$ The ChameleoClust$^+$ algorithm has been implemented in C++ as a Python library, and thus can be easily called from Python programs. The syntax of the calls intents to be similar to the one used by scikit-learn, a free machine learning library for Python. A ChameleoClust$^+$ object can be created specifying the desired parameters or using the default parameters defined in Section 3.3. Once the ChameleoClust$^+$ instance has been created, two specific methods can be used to produce rather a subspace clustering model for a static dataset or for a data stream. Users can eventually specify some parameters related to the evolution process such as the number of generations to be computed each time the data sample is modified and the size of the part of the data sample to be changed.

Package installation and content Installation details and example usages are given in the README.txt file at the root directory of the Evowave package. A Python script (chameleoclust_example.py), a toy dataset (pendigits.h5 [16]) and a README file are provided with the ChameleoClust$^+$ library and can be used as introductory example to use ChameleoClust$^+$ on a static dataset. The package contains also all programs developed in the project for the Evowave application and the data stream corresponding to the results presented in this Section 5.

6 Conclusion

We have presented ChameleoClust$^+$, a new evolutionary algorithm that is based on a variable genome length, using both functional and non-functional elements and relying on mutation operators that include chromosomal rearrangements. This evolvable genome structure allows to tackle the subspace clustering problem over dynamic data stream and to detect clusters that vary over time. These changes can be on the number of clusters, on the location of the clusters and also in the set of dimensions used by each cluster.

ChameleoClust$^+$ was shown to obtain very good performances when compared to the well-established subspace clustering methods using a reference benchmark containing both real world and synthetic datasets. ChameleoClust$^+$ also turned out to be very well suited to analyze real dynamic streams within the Evowave application. A dedicated visualization tool of the evolution process enabled to monitor and assess the subspace clusters, showing that the large degree of freedom of ChameleoClust$^+$, provided by its evolvable genome structure, allows it to adapt the clusters to the changes appearing within the stream over time.

Moreover, the parameter setting of ChameleoClust$^+$ was very easy. Indeed, only the maximal number of clusters needed to be changed, and the other parameters, that are them related to the evolutionary process, could be set to the same default values for all reported experiments. However, even though these settings were satisfactory, based on complementary experiments on a dynamic stream, we pointed out a very promising extension. This future direction of work is based on the reduction of the probability of promotion from non-functional elements to functional ones that showed preliminary evidences of performance improvement. The key hint will be here to relate the setting of this probability to the speed of the changes within the stream itself.

References

[1] A. Patrikainen and M. Meila, "Comparing subspace clusterings." *IEEE Transactions on Knowledge and Data Engineering*, pp. 902–916, 2006.

[2] H.-P. Kriegel, P. Kröger, and A. Zimek, "Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering," *ACM Transactions on Knowledge Discovery from Data*, vol. 3, no. 1, pp. 1:1–1:58, Mar. 2009.

[3] E. R. Hruschka, R. J. G. B. Campello, A. A. Freitas, and A. C. P. L. F. de Carvalho, "A survey of evolutionary algorithms for clustering," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 39, no. 2, pp. 133–155, 2009.

[4] I. A. Sarafis, P. W. Trinder, and A. Zalzala, "Towards effective subspace clustering with an evolutionary algorithm," in *Proc. of the IEEE Congress on Evolutionary Computation (CEC 2003)*, 2003, pp. 797–806.

[5] A. Vahdat, M. I. Heywood, and A. N. Zincir-Heywood, "Bottom-up evolutionary subspace clustering." in *Proc. of the IEEE Congress on Evolutionary Computation (CEC 2010)*, 2010, pp. 1–8.

[6] W. Banzhaf, G. Beslon, S. Christensen, A. James, F. Képès, V. Lefort, F. Julian, M. Radman, and J. J. Ramsden, "Guidelines: From artificial evolution to computational evolution: a research agenda," *Nature Reviews Genetics*, vol. 7, no. 9, pp. 729–735, 2006.

[7] C. Knibbe, A. Coulon, O. Mazet, J.-M. Fayard, and G. Beslon, "A Long-Term Evolutionary Pressure on the Amount of Noncoding DNA," *Molecular Biology and Evolution*, vol. 24, no. 10, pp. 2344–2353, Oct. 2007.

[8] T. Hindré, C. Knibbe, G. Beslon, and D. Schneider, "New insights into bacterial adaptation through in vivo and in silico experimental evolution," *Nature Reviews Microbiology*, vol. 10, pp. 352–365, May 2012.

[9] A. Crombach and P. Hogeweg, "Chromosome rearrangements and the evolution of genome structuring and adaptability." *Molecular Biology and Evolution*, vol. 24, no. 5, pp. 1130–9, 2007.

[10] S. Peignier, C. Rigotti, and G. Beslon, "Subspace clustering using evolvable genome structure," in *Proc. of the ACM Genetic and Evolutionary Computation Conference (GECCO 2015)*, 2015, pp. 1–8.

[11] E. Müller, S. Günnemann, I. Assent, and T. Seidl, "Evaluating clustering in subspace projections of high dimensional data," in *Proc. 35th Int. Conf. on Very Large Data Bases (VLDB 2009)*, Lyon, France, 2009, pp. 1270–1281.

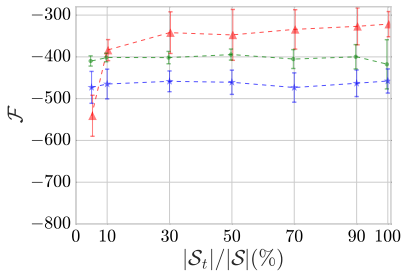
[12] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When is "nearest neighbor" meaningful?" in *Proc. of the 7th Int. Conf. on Database Theory*, London, UK, 1999, pp. 217–235.

[13] C. C. Aggarwal, A. Hinneburg, and D. A. Keim, "On the surprising behavior of distance metrics in high dimensional space," in *Proc. of the 8th Int. Conf. on Database Theory*. Springer, 2001, pp. 420–434.

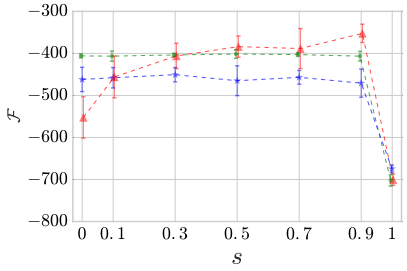
[14] C. C. Aggarwal, J. L. Wolf, P. S. Yu, C. Procopiuc, and J. S. Park, "Fast algorithms for projected clustering," in *Proc. of the 1999 ACM SIGMOD Int. Conf. on Management of Data*, New York, NY, USA, 1999, pp. 61–72.

[15] T. Blickle and L. Thiele, "A comparison of selection schemes used in evolutionary algorithms," *Evolutionary Computation*, vol. 4, no. 4, pp. 361–394, Dec. 1996.

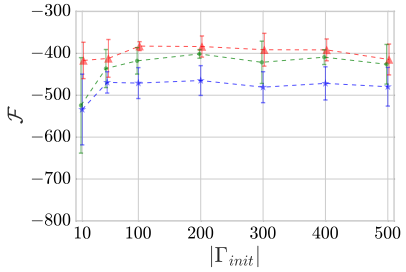
[16] K. Bache and M. Lichman, "UCI machine learning repository," 2013.



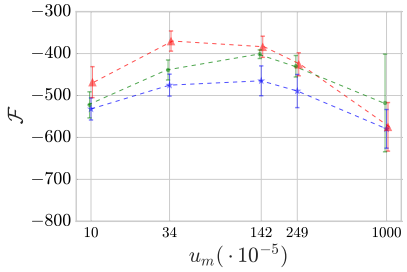
(a) Fitness vs. dataset sample size relative to the dataset size.



(b) Fitness vs. selection pressure.



(c) Fitness vs. initial genome size.



(d) Fitness vs. mutation rate.

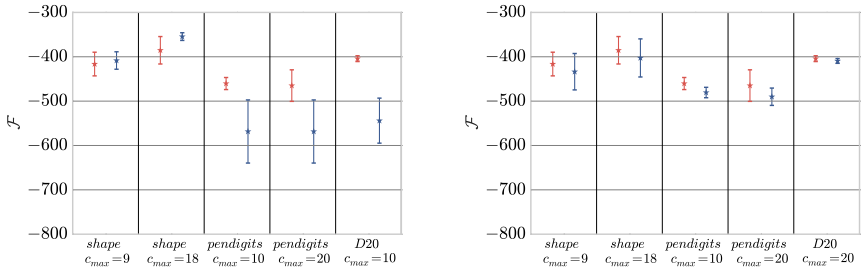
Figure 19: Mean ± standard deviation of the fitness of the best individual of the last generation for each one of the 10 runs on *shape* (red), *pendigits* (blue) and *D20* (green) under different conditions.

7 Appendix

We report in this Appendix complementary results.

Table 4: Results for the *breast* real dataset: 33 dimensions, 2 classes, 198 objects

	F1		*Accuracy*		*CE*		*RNIA*		*Entropy*		*Coverage*		*NumClusters*		*AvgDim*		*Runtime*	
	max	min	max	min	max	min	max	min	max	min	max	min	max	min	max	min	max	min
CLIQUE	0.67	0.67	0.71	0.71	0.02	0.02	0.40	0.40	0.26	0.26	1.00	1.00	107	107	1.7	1.7	453	453
DOC	0.73	0.61	0.81	0.76	0.11	0.04	0.84	0.07	0.46	0.27	1.00	0.80	60	6	27.2	2.8	1E+06	37515
MINECLUS	0.78	0.69	0.78	0.76	0.19	0.18	1.00	1.00	0.56	0.37	1.00	1.00	64	32	33.0	33.0	40359	29437
SCHISM	0.67	0.67	0.75	0.69	0.01	0.01	0.36	0.34	0.35	0.34	1.00	0.99	248	197	2.3	2.2	158749	114609
SUBCLU	0.68	0.51	0.77	0.67	0.02	0.01	0.54	0.04	0.27	0.24	1.00	0.82	357	5	2.0	1.0	5265	16
FIRES	0.49	0.03	0.76	0.76	0.03	0.00	0.05	0.00	1.00	0.01	0.76	0.04	11	1	2.5	1.0	250	31
INSCY	0.74	0.55	0.77	0.76	0.02	0.00	0.24	0.11	0.60	0.39	0.97	0.74	2038	167	11.0	4.4	134373	63484
PROCLUS	0.57	0.52	0.80	0.74	0.51	0.11	0.65	0.43	0.32	0.23	0.89	0.69	9	2	24.0	18.0	703	141
P3C	0.63	0.63	0.77	0.77	0.04	0.04	0.19	0.19	0.36	0.36	0.85	0.85	28	28	6.9	6.9	6281	6281
STATPC	0.41	0.41	0.78	0.78	0.16	0.16	0.33	0.33	0.29	0.29	0.43	0.43	5	5	33.0	33.0	5187	4906
ChameleoClust$^+$	0.60	0.51	0.76	0.76	0.23	0.11	0.53	0.25	0.25	0.22	1	1	8	4	16.75	5.75	339	131
mean	0.56		0.76		0.17		0.40		0.24		1		5.1		12.15		230	



(a) Fitness with (red) and without (blue) elitism.

(b) Fitness with (red) and without (blue) non-functional tuples.

Figure 20: Mean ± standard deviation of the fitness of the best individual of the last generation for 10 runs on *shape*, *pendigits* and *D20* under different conditions. For each real world dataset two c_{max} values where tested: the number of classes in the dataset and twice this number and the real number of cluster was used as c_{max} value for the synthetic dataset.

Table 5: Results for the *pendigits* real dataset: 16 dimensions, 10 classes, 7494 objects

	F1		*Accuracy*		*CE*		$\overline{RNIA}$		$\overline{Entropy}$		*Coverage*		*NumClusters*		*AvgDim*		*Runtime*	
	max	min	max	min	max	min	max	min	max	min	max	min	max	min	max	min	max	min
CLIQUE	0.30	0.17	0.96	0.86	0.06	0.01	0.20	0.06	0.41	0.26	1.00	1.00	1890	36	3.1	1.5	67891	219
DOC	0.52	0.52	0.54	0.54	0.18	0.18	0.35	0.35	0.53	0.53	0.91	0.91	15	15	5.5	5.5	178358	178358
MINECLUS	0.87	0.87	0.86	0.86	0.48	0.48	0.89	0.89	0.82	0.82	1.00	1.00	64	64	12.1	12.1	780167	692651
SCHISM	0.45	0.26	0.93	0.71	0.05	0.01	0.30	0.08	0.50	0.45	1.00	0.93	1092	290	10.1	3.4	5E+08	21266
SUBCLU	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
FIRES	0.45	0.45	0.73	0.73	0.09	0.09	0.33	0.33	0.31	0.31	0.94	0.94	27	27	2.5	2.5	169999	169999
INSCY	0.65	0.48	0.78	0.68	0.07	0.07	0.30	0.28	0.77	0.69	0.91	0.82	262	106	5.3	4.6	2E+06	1E+06
PROCLUS	0.78	0.73	0.74	0.73	0.31	0.27	0.64	0.45	0.90	0.71	0.90	0.74	37	17	14.0	8.0	6045	4250
P3C	0.74	0.74	0.72	0.72	0.28	0.28	0.58	0.58	0.76	0.76	0.90	0.90	31	31	9.0	9.0	2E+06	2E+06
STATPC	0.91	0.32	0.92	0.10	0.09	0.00	0.67	0.11	1.00	0.53	0.99	0.84	4109	56	16.0	16.0	5E+07	3E+06
ChameleoClust$^+$	0.71	0.51	0.74	0.59	0.51	0.30	0.78	0.49	0.68	0.58	1	1	14	10	12.40	7.21	4476	4226
mean	0.64		0.68		0.43		0.67		0.63		1		11.6		10.01		4347	

Table 6: Results for the *diabetes* real dataset: 8 dimensions, 2 classes, 768 objects

	F1		*Accuracy*		*CE*		$\overline{RNIA}$		$\overline{Entropy}$		*Coverage*		*NumClusters*		*AvgDim*		*Runtime*	
	max	min	max	min	max	min	max	min	max	min	max	min	max	min	max	min	max	min
CLIQUE	0.70	0.39	0.72	0.69	0.03	0.01	0.14	0.01	0.23	0.13	1.00	1.00	349	202	4.2	2.4	11953	203
DOC	0.71	0.71	0.72	0.69	0.31	0.26	0.92	0.79	0.31	0.24	1.00	0.93	64	17	8.0	5.1	1E+06	51640
MINECLUS	0.72	0.66	0.71	0.69	0.63	0.13	0.89	0.58	0.29	0.17	0.99	0.96	39	3	6.0	5.2	3578	62
SCHISM	0.70	0.62	0.73	0.68	0.08	0.01	0.36	0.09	0.34	0.20	1.00	0.79	270	21	4.2	3.9	35468	250
SUBCLU	0.74	0.45	0.71	0.68	0.01	0.01	0.01	0.01	0.14	0.11	1.00	1.00	1601	325	4.7	4.0	190122	58718
FIRES	0.52	0.03	0.65	0.64	0.12	0.00	0.27	0.00	0.68	0.00	0.81	0.03	17	1	2.5	1.0	4234	360
INSCY	0.65	0.39	0.70	0.65	0.37	0.11	0.45	0.42	0.44	0.15	0.83	0.73	132	3	6.7	5.7	112093	33531
PROCLUS	0.67	0.61	0.72	0.71	0.34	0.21	0.78	0.69	0.23	0.19	0.92	0.78	9	3	8.0	6.0	360	109
P3C	0.39	0.39	0.66	0.65	0.56	0.11	0.85	0.22	0.09	0.07	0.97	0.88	2	1	7.0	2.0	656	141
STATPC	0.73	0.59	0.70	0.65	0.06	0.00	0.63	0.17	0.72	0.28	0.97	0.75	363	27	8.0	8.0	27749	4657
ChameleoClust$^+$	0.70	0.62	0.73	0.70	0.17	0.09	0.66	0.47	0.28	0.23	1	1	29	19	5.00	2.75	598	438
mean	0.68		0.72		0.13		0.55		0.25		1		25.1		3.85		480	

Table 7: Results for the *glass* real dataset: 9 dimensions, 6 classes, 214 objects

	F1		*Accuracy*		*CE*		$\overline{RNIA}$		$\overline{Entropy}$		*Coverage*		*NumClusters*		*AvgDim*		*Runtime*	
	max	min	max	min	max	min	max	min	max	min	max	min	max	min	max	min	max	min
CLIQUE	0.51	0.31	0.67	0.50	0.02	0.00	0.06	0.00	0.39	0.24	1.00	1.00	6169	175	5.4	3.1	411195	1375
DOC	0.74	0.50	0.63	0.50	0.23	0.13	0.93	0.33	0.72	0.50	0.93	0.91	64	11	9.0	3.3	23172	78
MINECLUS	0.76	0.40	0.52	0.50	0.24	0.19	0.78	0.45	0.72	0.46	1.00	0.87	64	6	7.0	4.3	907	15
SCHISM	0.46	0.39	0.63	0.47	0.11	0.04	0.33	0.20	0.44	0.38	1.00	0.79	158	30	3.9	2.1	313	31
SUBCLU	0.50	0.45	0.65	0.46	0.00	0.00	0.01	0.01	0.42	0.39	1.00	1.00	1648	831	4.9	4.3	14410	4250
FIRES	0.30	0.30	0.49	0.49	0.21	0.21	0.45	0.45	0.40	0.40	0.86	0.86	7	7	2.7	2.7	78	78
INSCY	0.57	0.41	0.65	0.47	0.23	0.09	0.54	0.26	0.67	0.47	0.86	0.79	72	30	5.9	2.7	4703	578
PROCLUS	0.60	0.56	0.60	0.57	0.13	0.05	0.51	0.17	0.76	0.68	0.79	0.57	29	26	8.0	2.0	375	250
P3C	0.28	0.23	0.47	0.39	0.14	0.13	0.30	0.27	0.43	0.38	0.89	0.81	3	2	3.0	3.0	32	31
STATPC	0.75	0.40	0.49	0.36	0.19	0.05	0.67	0.37	0.88	0.36	0.93	0.80	106	27	9.0	9.0	1265	390
ChameleoClust$^+$	0.43	0.28	0.57	0.50	0.43	0.26	0.88	0.55	0.46	0.36	1	1	8	4	7.50	4.75	195	95
mean	0.37		0.54		0.37		0.78		0.42		1		6.9		6.18		154	

Table 8: Results for the *liver* real dataset: 6 dimensions, 2 classes, 345 objects

	F1		*Accuracy*		*CE*		*RNIA*		*Entropy*		*Coverage*		*NumClusters*		*AvgDim*		*Runtime*	
	max	min	max	min	max	min	max	min	max	min	max	min	max	min	max	min	max	min
CLIQUE	0.68	0.65	0.67	0.58	0.08	0.02	0.38	0.03	0.10	0.02	1.00	1.00	1922	19	4.1	1.7	38281	15
DOC	0.67	0.64	0.68	0.58	0.11	0.07	0.51	0.35	0.18	0.11	0.99	0.90	45	13	3.0	1.9	625324	1625
MINECLUS	0.73	0.63	0.65	0.58	0.09	0.09	0.68	0.48	0.33	0.16	0.99	0.92	64	32	4.0	3.7	49563	1954
SCHISM	0.69	0.69	0.68	0.59	0.04	0.03	0.45	0.26	0.10	0.08	0.99	0.99	90	68	2.7	2.1	31	0
SUBCLU	0.68	0.68	0.64	0.58	0.11	0.02	0.68	0.05	0.07	0.02	1.00	1.00	334	64	3.4	1.3	1422	47
FIRES	0.58	0.04	0.58	0.56	0.14	0.00	0.39	0.01	0.37	0.00	0.84	0.03	10	1	3.0	1.0	531	46
INSCY	0.66	0.66	0.62	0.61	0.03	0.03	0.42	0.39	0.21	0.20	0.85	0.81	166	130	2.1	2.1	407	234
PROCLUS	0.53	0.39	0.63	0.63	0.26	0.11	0.66	0.25	0.05	0.05	0.83	0.46	6	2	5.0	3.0	78	31
P3C	0.36	0.35	0.58	0.58	0.55	0.27	0.96	0.47	0.02	0.01	0.98	0.94	2	1	6.0	3.0	172	32
STATPC	0.69	0.57	0.65	0.58	0.23	0.01	0.58	0.37	0.63	0.05	0.77	0.71	159	4	6.0	3.3	1890	781
ChameleoClust$^+$	0.65	0.59	0.68	0.62	0.20	0.10	0.53	0.41	0.14	0.07	1	1	27	22	2.48	1.85	202	158
mean	0.62		0.64		0.14		0.47		0.11		1		24.3		2.06		179	

Table 9: Results for the *vowel* real dataset: 10 dimensions, 11 classes, 990 objects

	F1		*Accuracy*		*CE*		*RNIA*		*Entropy*		*Coverage*		*NumClusters*		*AvgDim*		*Runtime*	
	max	min	max	min	max	min	max	min	max	min	max	min	max	min	max	min	max	min
CLIQUE	0.23	0.17	0.64	0.37	0.05	0.00	0.44	0.01	0.10	0.09	1.00	1.00	3062	267	4.9	1.9	523233	1953
DOC	0.49	0.49	0.44	0.44	0.14	0.14	0.85	0.85	0.58	0.58	0.86	0.86	64	64	10.0	10.0	120015	120015
MINECLUS	0.48	0.43	0.37	0.37	0.09	0.04	0.62	0.34	0.60	0.46	0.98	0.87	64	64	7.2	3.6	7734	5204
SCHISM	0.37	0.23	0.62	0.52	0.05	0.01	0.43	0.11	0.29	0.21	1.00	0.93	494	121	4.3	2.8	23031	391
SUBCLU	0.24	0.18	0.58	0.38	0.04	0.01	0.39	0.04	0.30	0.13	1.00	1.00	10881	709	3.6	2.0	26047	2250
FIRES	0.16	0.14	0.13	0.11	0.02	0.02	0.14	0.13	0.16	0.13	0.50	0.45	32	24	2.1	1.9	563	250
INSCY	0.82	0.33	0.61	0.15	0.09	0.07	0.75	0.26	0.94	0.21	0.90	0.81	163	74	9.5	4.3	75706	39390
PROCLUS	0.49	0.49	0.44	0.44	0.11	0.11	0.53	0.53	0.65	0.65	0.67	0.67	64	64	8.0	8.0	766	766
P3C	0.08	0.05	0.17	0.16	0.12	0.08	0.69	0.43	0.13	0.12	0.98	0.95	3	2	7.0	4.7	1610	625
STATPC	0.22	0.22	0.56	0.56	0.06	0.06	0.12	0.12	0.14	0.14	1.00	1.00	39	39	10.0	10.0	18485	16671
ChameleoClust$^+$	0.41	0.37	0.42	0.38	0.17	0.13	0.65	0.54	0.45	0.40	1	1	33	24	6.00	4.57	995	787
mean	0.39		0.40		0.15		0.60		0.42		1		28.0		5.41		910	