Evolutionary Subspace Clustering Using Variable Genome Length

Sergio Peignier, Christophe Rigotti, Guillaume Beslon

Evolutionary Subspace Clustering Using Variable Genome Length

Sergio Peignier[1], Christophe Rigotti[2], and Guillaume Beslon[2]

[1] Univ Lyon, INSA-Lyon, INRA, BF2I, UMR0203, F-69621, Villeurbanne, France
sergio.peignier@insa-lyon.fr
[2] Univ Lyon, INSA-Lyon, CNRS, INRIA, LIRIS, UMR5205, F-69621, Villeurbanne, France
christophe.rigotti@insa-lyon.fr, guillaume.beslon@inria.fr

Abstract. Subspace clustering is a data mining task that groups similar data objects and at the same time searches the subspaces where similarities appear. For this reason, subspace clustering is recognized as more general and complicated than standard clustering. In this paper, we present ChameleoClust$^+$, a bio-inspired evolutionary subspace clustering algorithm that takes advantage of an evolvable genome structure to detect various numbers of clusters located in different subspaces. ChameleoClust$^+$ incorporates several bio-like features such as a variable genome length, both functional and non-functional elements and mutation operators including large rearrangements. It was assessed and compared to the state-of-the-art methods on a reference benchmark using both real world and synthetic datasets. While other algorithms may need complex parameter settings, ChameleoClust$^+$ needs to set only one subspace clustering ad-hoc and intuitive parameter: the maximal number of clusters. The remaining parameters of ChameleoClust$^+$ are related to the evolution strategy (e.g., population size, mutation rate) and a single setting for all of them turned out to be effective 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.

Keywords: Evolutionary algorithm, Subspace clustering, Variable genome length.

1 Introduction

Clustering is a data mining task that aims to group objects sharing similar characteristics into sets (i.e., the clusters) over the whole data space. A related problem is the subspace clustering one, which purpose is not only to identify groups of similar objects, but also to detect the subspaces where these similarities occur. Retrieving such subspaces turns out to be particularly useful while dealing with high dimensional data (Kriegel et al., 2009). Subspace clustering can be conceived as *"similarity examined under different representations"* (Patrikainen

and Meila, 2006). For this reason it is recognized as a more complicated and general task than standard clustering.

Several evolutionary clustering approaches have been proposed (Hruschka et al., 2009), however very few of them address the subspace clustering task. As described in Section 5, these earlier approaches require non-evolutionary steps to tackle this problem. In order to address the subspace clustering task, we decided not to rely on non-evolutionary stages, but rather to take advantage of an evolvable genome structure. According to (Banzhaf et al., 2006) 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 (Knibbe et al., 2007). 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) (Hindré et al., 2012). Among the state-of-the-art formalisms used for *in silico* experimental evolution reviewed in (Hindré et al., 2012), two models enable genome structure evolution: (Knibbe et al., 2007) and (Crombach and Hogeweg, 2007). Both formalisms have inspired key aspects of our work.

In this paper, we present ChameleoClust$^+$, an evolutionary algorithm that takes advantage of a genome having an evolvable structure to tackle the subspace clustering problem. ChameleoClust$^+$ genome is a *coarse-grained* genome, inspired on (Crombach and Hogeweg, 2007), and is 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, and to build the subspace clusters around these core points. Furthermore, the genome also contains a variable proportion of non-functional elements as in (Knibbe et al., 2007). During replications the genome undergoes both local mutations and large random rearrangements similar to those used in (Knibbe et al., 2007) and (Crombach and Hogeweg, 2007), namely: large deletions and duplications. Local mutations modify the genome elements, while rearrangements modify the genome length, and both can change 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 order to assess the algorithm, we used the reference subspace clustering evaluation framework presented in (Müller et al., 2009), and compared it to state-of-the-art algorithms on both real and synthetic datasets. The experiments show that ChameleoClust$^+$ obtains competitive results. Moreover, these results can be achieved with a single parameter related to the domain: the maximal number of clusters. In addition, for each generation the computational complexity is linear with respect to the number of objects and to the number of

dimensions, and a fast overall evolutionary convergence is observed. This enables to keep the time consumption low on the datasets of the evaluation framework.

The main contribution of this paper is to show that, using an evolvable genome structure, a single stage fully evolutionary approach can consistently deliver subspace clusters of very good quality, requiring only an easy parameter setting and limited time resources.

The rest of the paper 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 related work and we conclude with a summary 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 (Knibbe et al., 2007) and (Crombach and Hogeweg, 2007), 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 using 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 the origin $\mathcal{O}$). 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 paper.

2.4 Genome structure

A genome Γ is a list $[\gamma_1, \ldots, \gamma_i, \ldots, \gamma_n]$ 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: $c \in \{1, \ldots, c_{max}\}$, $d \in \{1, \ldots, D\}$ and $x \in ValCoord$, with $ValCoord = \{j \times x_{max}/1000 \mid j \in \{-1000, \ldots, 1000\}\}$, 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 of the number of functional elements for different datasets.

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 a maximal number of desired clusters specified as a parameter 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. For a given core point index $c \in [1, c_{max}]$, the subspace associated to core point p_c is the set $\mathcal{D}_{p_c}$ containing the dimensions that contribute to the location of p_c, i.e., the set of all the dimensions d in $\mathcal{D}$ such that there exists at least one functional element of the form $\langle 1, c, d, x \rangle$ in Γ, where x can be any coordinate value.

However, if we have some knowledge about the cluster locations, then other mappings than the z-score could be more appropriate. For instance, if we know that there are several clusters located close to the center of the whole dataset, then a sigmoid function (with range equal to the open interval $(-1, 1)$) can enforce the separation of these clusters.

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 $\gamma_i = \langle g, c, d, x\rangle$ be an element uniformly drawn in the genome, and let $k \in \{1, 2, 3, 4\}$ be 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 (Beyer et al., 1999) and (Aggarwal et al., 2001) it has been shown that distance comparisons are less meaningful when dimensionality increases, this effect is called the *concentration effect* of the distances. It has been shown in (Aggarwal et al., 2001) that the Manhattan distance is robust to this effect. In the ChameleoClust$^+$ algorithm, the distance used is the *Manhattan segmental distance* introduced in (Aggarwal et al., 1999) 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 $x \in \mathcal{S}$ 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 $d_{\mathcal{D}_{p_c}}(x, p_c)$). It 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 $d_{\mathcal{D} \setminus \mathcal{D}_{p_c}}(x, \mathcal{O})$). 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 of the dataset objects:

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

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.

The computation cost of $\mathcal{F}(\Phi, \mathcal{S})$ is proportional to the size of the dataset $\mathcal{S}$, but to guide the search it is not necessary to evaluate the fitness over the whole input dataset $\mathcal{S}$, and it can be sufficient to evaluate it over a sample. This strategy is used in ChameleoClust$^+$, with an incrementally changing sample to avoid the possible misleading consequences of a poor single sample selection (i.e., sample not very representative of $\mathcal{S}$). This is defined as follows. Let t be the index of the current generation (starting at $t = 0$). Let $\mathcal{L} = [x_0, x_1 \dots]$ be a list containing all the dataset objects in a random order. For generation t, the fitness value of an individual is then $\mathcal{F}(\Phi, \mathcal{S}^t)$ where $\mathcal{S}^t$ is a subset of $\mathcal{S}$ of size ω defined by:

$$\mathcal{S}^t = \bigcup_{k=t\times\omega}^{t\times\omega+\omega-1} \{x \text{ in } \mathcal{L} \text{ at index } (k \bmod |\mathcal{L}|)\}$$

$\mathcal{S}^t$ is simply the set of objects in $\mathcal{L}$ from index $t \times \omega$ to index $t \times \omega + \omega - 1$, restarting from the beginning of $\mathcal{L}$ when the last element is reached.

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.

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 (Knibbe et al., 2007) we rely on an exponential ranking selection (Blickle and Thiele, 1996) 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. This procedure is sketched in Figure 1.

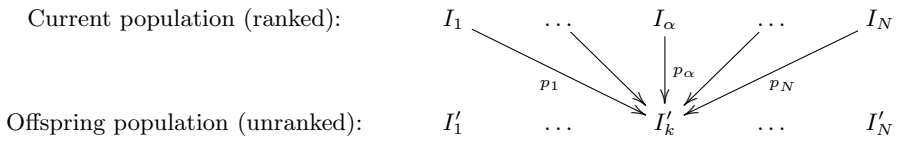


Fig. 1: Parent of the new individual I'_k drawn from current population according to this population ranking, and with probabilities $p_1 < \ldots < p_\alpha < \ldots < p_N$.

In order to avoid the possible decrease of the best fitness during the evolution, the algorithm 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.

For the initialization of the population, the N individuals of the first generation have genomes of the same size, denoted $|\Gamma_{init}|$, and containing only non-functional elements. These N genomes are drawn independently, and filled with random tuples of the form $\langle 0, \mathcal{U}(\{1,\ldots,c_{max}\}), \mathcal{U}(\{1,\ldots,D\}), \mathcal{U}(ValCoord\})\rangle$.

2.9 Time complexity

Let $|\Gamma|$ be the maximal genome size among all individuals in the current generation. In this section we distinguish the time complexity related to the fitness computation and the complexity related to the reproduction operations.

Fitness computation In order to compute the fitness of an individual, the algorithm first needs to build the phenotype of this individual from its genome. Considering that only the set of functional tuples, denoted Γ_f, contribute to the phenotype, only these tuples should be selected, this search having a complexity of $\mathcal{O}(|\Gamma|)$. Once each functional tuple has been retrieved, ChameleoClust$^+$ proceeds to sort them by cluster and dimension to obtain the phenotype, this operation has a complexity of $\mathcal{O}(|\Gamma_f| \times ln(|\Gamma_f|))$. Once the phenotype has been built, the algorithm associates each one of the ω objects in $\mathcal{S}^t$ to the core point it matches the best. Since, in the worst case, each element in Γ_f can define a core point, this operation has a complexity of $\mathcal{O}(|\Gamma_f| \times D \times \omega)$. Thus, for each individual, the time complexity related to the fitness computation is $\mathcal{O}(|\Gamma| + |\Gamma_f| \times ln(|\Gamma_f|) + |\Gamma_f| \times D \times \omega)$. In the worst case all tuples are functional, i.e., $|\Gamma| = |\Gamma_f|$, and the complexity is $\mathcal{O}(|\Gamma| \times (D \times \omega + ln(|\Gamma|))$. Then, the complexity of the fitness computation over the whole population is given by:

$$\mathcal{O}(N \times |\Gamma| \times (D \times \omega + ln(|\Gamma|)))$$

Reproduction operations When the individual fitnesses have been computed, ChameleoClust$^+$ proceeds to rank the individuals in order to give them a reproduction probability. This operation has a complexity of $\mathcal{O}(N \times ln(N))$. The genomes of the individuals of the new generation are initialized by copying the genomes of their parents, this operation having a complexity of $\mathcal{O}(N \times |\Gamma|)$. Then, these genomes are modified by rearrangements (large duplications and large deletions) followed by point mutations. Let L_m be the maximal genome size reached during the rearrangement steps for all individuals. For one genome, the numbers of duplications and of deletions are drawn from a binomial law and cannot be greater than $|\Gamma|$, leading for their application to a complexity of $\mathcal{O}(|\Gamma| \times L_m)$. In a similar way, the number of point mutations is bounded by L_m, and the complexity of the application of this operator is $\mathcal{O}(L_m)$. The expression of the complexity of the reproduction operations for the whole population is then $\mathcal{O}(N \times ln(N) + N \times |\Gamma| + N \times (|\Gamma| \times L_m + L_m))$, rewritten simply as:

$$\mathcal{O}(N \times (ln(N) + |\Gamma| + |\Gamma| \times L_m + L_m))$$

It should be noticed that this worst case is not reached in the experiments. Indeed, for a genome of size $|\Gamma|$, the number of rearrangements (for both kinds) is not $|\Gamma|$, but is only $|\Gamma| \times u_m$ on average, where effective parameter settings correspond to low values of u_m ($u_m \ll 1$), as shown in the next section.

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 (Müller et al., 2009). 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 (Müller et al., 2009) 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 (Müller et al., 2009), 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 (Müller et al., 2009) for their representativity: *breast, diabetes, liver, glass, shape, pendigits* and *vowel* (most of them coming from the UCI archive (Bache and Lichman, 2013)). 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 (Müller et al., 2009). 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 (Müller et al., 2009) 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 analyze 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 built by adding noise points to the dataset *D20*.

All datasets and additional description are made available by the authors of (Müller et al., 2009) 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. The sliding sample size ω was set to 10% of the dataset size and this setting turned out to be an interesting trade-off, as shown in Section 4.3 Figure 8. Of course, while the fitness is computed only on this sample, the final association of objects to clusters (using the core points defined by the best individual) and the evaluation of this clustering are still performed on the whole dataset.

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 $p_\alpha = \frac{s-1}{s^N-1}$. 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 explore the neighborhood of the best current individual, and the same chance to descend from another one, exploring then potentially different solutions.

Initial genome size The genomes are initialized with random tuples denoting non-functional elements (see Section 2.8) and the size of these initial genomes 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 are not constrained and are able to evolve (as illustrated in figures 4b and 4c), 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 2. 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 2 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.

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 $u_m = \frac{u_{min}+u_{max}}{2} \approx 0.00142$

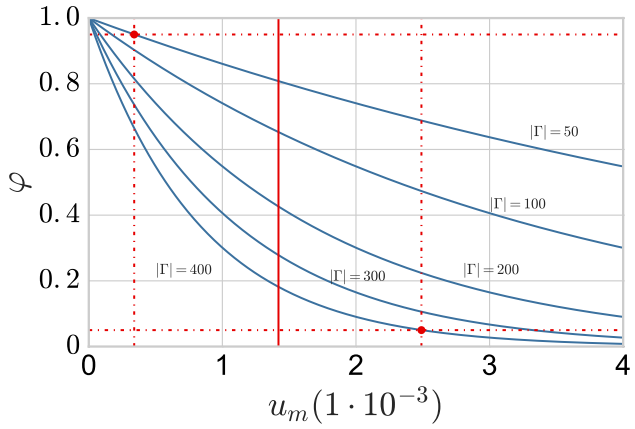


Fig. 2: φ value computed as a function of the mutation rate u_m for different genome sizes. The suitable chosen range of genomic variability and its related mutation rate range are delimited by dashed lines. The retained mutation rate is marked by a vertical plain line.

Population size and number of generations In order to adjust these parameters, we analyzed the fitness value and its convergence curves on three datasets: *shape* and *pendigits*, that are respectively the smallest and the largest of the real datasets, and dataset *D20* a typical synthetic dataset of the framework (20 dimensions and 10% of noise points).

For the population size, Figure 3 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 then tend to be less significant. Figure 3 illustrates that an appropriate fitness convergence is reached with a population size set to $N = 300$ or greater.

For the number of generations, at 5000 the algorithm achieved a good convergence for fitness. This is illustrated in Figure 4a, 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.

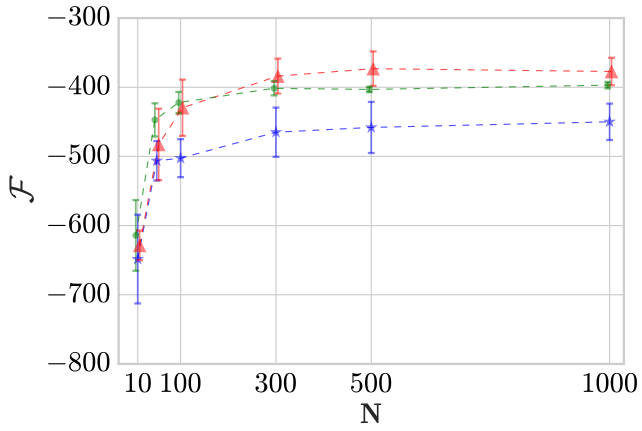


Fig. 3: Mean fitness values ± standard deviation for the best individual of the last generation for each one of the 10 runs on *shape* (red), *pendigits* (blue) and *D20* (green) as a function of the population size.

Figures 4b and 4c illustrate that the early generations are characterized by a fast evolution of the genome structure, and particularly of the number of functional tuples and of the fraction of functional tuples. At 5000 generations, the algorithm has already been able to take advantage of the genome structure evolution to encode a subspace clustering model having a fitness value close to the maximum reached Figure 4a. 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.

Maximal number of subspace clusters c_{max} is the maximal number of subspace clusters that can be built, and it was the only parameter that required to be tuned. The other parameters are related to the evolution strategy (population size, mutation rate, ...) and for all of them the single setting established previously in this section turned out to be effective for all the benchmark datasets. However, c_{max} does not require a fine tuning since ChameleoClust$^+$ can adapt the number of subspace clusters between 1 and c_{max}. 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 pro-

cedure, 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.

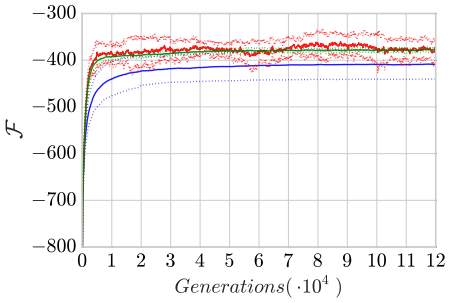
3.4 Evaluation measures

In order to compare our algorithm to the others, we used the same standard evaluation measures for clusters and subspace clusters as (Müller et al., 2009): 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 (Patrikainen and Meila, 2006), 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 (Müller et al., 2009) all dimensions have been considered as relevant, but then the interpretation of these measures should remain cautious 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 (Müller et al., 2009) for a detailed presentation of the evaluation measures.

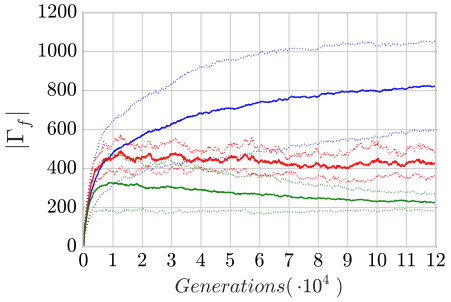
4 Experimental results

4.1 Real datasets

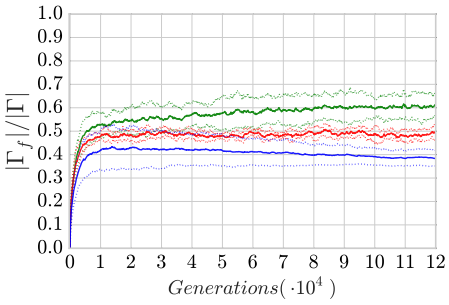
We computed the minimum, the maximum and the mean of the evaluation measures over 10 standard runs of ChameleoClust$^+$ using the same parameter setting for all datasets as justified and given in Section 3.3, except of course for the parameter specifying the maximum number of clusters (c_{max}) that 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 (Müller et al., 2009), 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 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



(a) Evolution of the fitness values.



(b) Evolution of the number of functional tuples.



(c) Evolution of the percentage of functional tuples.

Fig. 4: Evolution of the mean ± standard deviation (dashed lines) of different measures for the best individuals of each generation for 10 runs over the real world datasets *shape* (red) and *pendigits* (blue) and the synthetic dataset *D20* (green).

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 (Müller et al., 2009) (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 that reproduces the results obtained in (Müller et al., 2009) completed by the results of ChameleoClust$^+$. For the sake of completeness, the detailed evaluation on the other datasets is given in the Appendix 7.3. In Table 1, when an algorithm has a best possible run with a higher evaluation than ChameleoClust$^+$ the result is highlighted in gray, 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, 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

Table 1: Results on *shape* dataset: 17 dimensions, 9 classes, 160 objects, from (Müller et al., 2009) completed by results of ChameleoClust$^+$

	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 (10 runs) $\longrightarrow$	0.68		0.75		0.52		0.76		0.72		1		12.0		11.72		339	

correspond to the number of clusters found by the algorithms. Indeed, there is no constraint requiring that the objects of a class form a single group in their feature 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 summarizes (over 10 runs) 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 the *breast* dataset to 28.0 for the *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 (see Table 1 for dataset *shape* and Appendix 7.3 for the others).

Broader comparison For almost every dataset, the performances of ChameleoClust$^+$ are competitive with respect to the best possible runs of the other algorithms. To compare these approaches in a broader way, we ranked them according to the eight following evaluation criteria: the coverage, the number of clusters found, the quality measures (F1, Accuracy, CE, RNIA and Entropy), and the runtime. For each real world dataset and each criterion we ranked the eleven algorithms with respect to the column best *max*, from rank 1 for the lowest performance to rank 11 for the highest one. The ranking for the coverage and for the number of clusters needs further precisions. For the coverage, a method that built less representative models (excluding too many points) had a lower rank with respect to a method that covered a larger part of the dataset. For the number of clusters,

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

the fewer the clusters in the clustering model, the easier their interpretation, so methods that built a reduced number of clusters had a higher rank.

Then, for each of the eight criteria we computed the average rank over the seven datasets, obtaining for each algorithm eight average ranks. The same was also performed with the column best *min*. The average ranks of the different algorithms are given in Figure 5 (colored dots). The figure also reports the mean of the average rank of each method (red stars), showing that ChameleoClust$^+$ has the second best mean. However, it should be noticed that there is no ever winning algorithm.

For the four algorithms having the best means (i.e., MINECLUS, ChameleoClust$^+$, DOC and PROCLUS) we compared more precisely the number of clusters they produced, their coverage and their runtimes. The table 3 summarizes for each algorithm: (1) The number of datasets where the highest and lowest number of clusters found remain interpretable (100 clusters or less). (2) The number of datasets where the highest and lowest coverage are acceptable, i.e., the amount of excluded data points are not too high (coverage of at least 95%). And (3) the number of datasets where the shortest and longest execution last for a reasonable time (one hour or less). The results obtained by the other algorithms are also presented for the sake of completeness. MINECLUS, ChameleoClust$^+$, DOC and PROCLUS produced for each dataset an interpretable number of clusters, but PROCLUS and DOC usually produced lower coverage. MINECLUS and DOC had higher run times and last for more than one hour for several datasets. ChameleoClust$^+$ produced good quality results together with low runtimes and high coverage.

However, the different approaches have different characteristics (e.g., global cluster shapes, distance-based/density-based, 100% coverage or not) and, as observed previously, no method leads to the best results on all datasets. For exploratory analysis of the data, a good strategy is to apply several methods from different families. In particular using at least one of the approaches that tend to build hyper-spherical shaped clusters. In the comparison, this corresponds to the algorithms PROCLUS, P3C, STATPC and ChameleoClust$^+$. Among

these four methods, ChameleoClust$^+$ always reaches a 100% coverage, while the others, on most datasets, do not cluster more than 95% of the objects, as shown Table 3. Thus, beyond an easy parameter setting and good performances, ChameleoClust$^+$ is an interesting choice to find hyper-spherical shaped clusters in subspaces with a 100% coverage of the data.

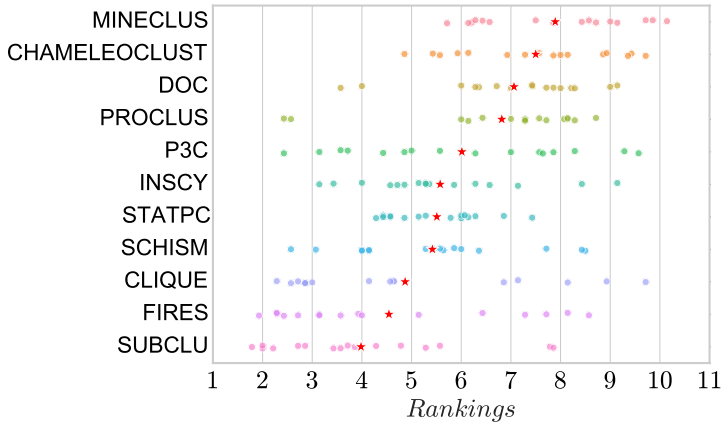


Fig. 5: Mean over the different datasets of the ranking of each algorithm for the maximum and the minimum value obtained for each evaluation measure: Accuracy, Entropy, F1, CE, RNIA, Number of cluster, Coverage, Runtime (colored dots) and average ranking for each method (red stars).

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 retained 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 6 as red dots. For each evaluation measure we also plotted in blue the shape of the area where the other algorithm results lay (as reported in (Müller et al., 2009)). 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 true labels, only the execution leading to the highest value of the measure has been retained. In the plots of the figure 6, 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 (Müller et al., 2009) 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). Using the default parameter setting method of Section 3.3, most of the evaluation measures for ChameleoClust$^+$ are comparable to the highest evaluations obtained by (Müller et al., 2009) when exploring the parameter space of the other algorithms using the true clusters to guide the search.

We give, Figure 7a and Figure 7b, the runtime of ChameleoClust$^+$ with respect to the number of dimensions, and to the number of objects of the synthetic datasets. These curves show that the algorithm scales rather linearly in both cases and are consistent with the time complexity obtained in Section 2.9. These confirmations are important in order to infer the sizes of real datasets that could be processed. For example, on the largest real dataset, *pendigits*, having about 7500 objects and 16 dimensions, the runtime of algorithm ChameleoClust$^+$ is less than 4500 seconds (Table 5). This runtime corresponds to a single threaded version of ChameleoClust$^+$, and it could be reduced by handling individuals in parallel since each member of the new generation can be obtained independently. For instance, the computation of the offspring population can be distributed over the 32 cores of a typical workstation, in order to decrease the execution time by a factor of about 1/32. Thus, according to the time complexity given in Section 2.9, showing that the runtime increases linearly with respect to the number of objects (as confirmed by the experiments reported Figure 7b), it is possible to process on a 32 cores hardware a 32 times larger dataset with similar execution times. This means that in a reasonable amount of time of about 4500 seconds, ChameleoClust$^+$ could obtain a subspace clustering for a dataset of $7500 \times 32 = 240000$ objects.

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

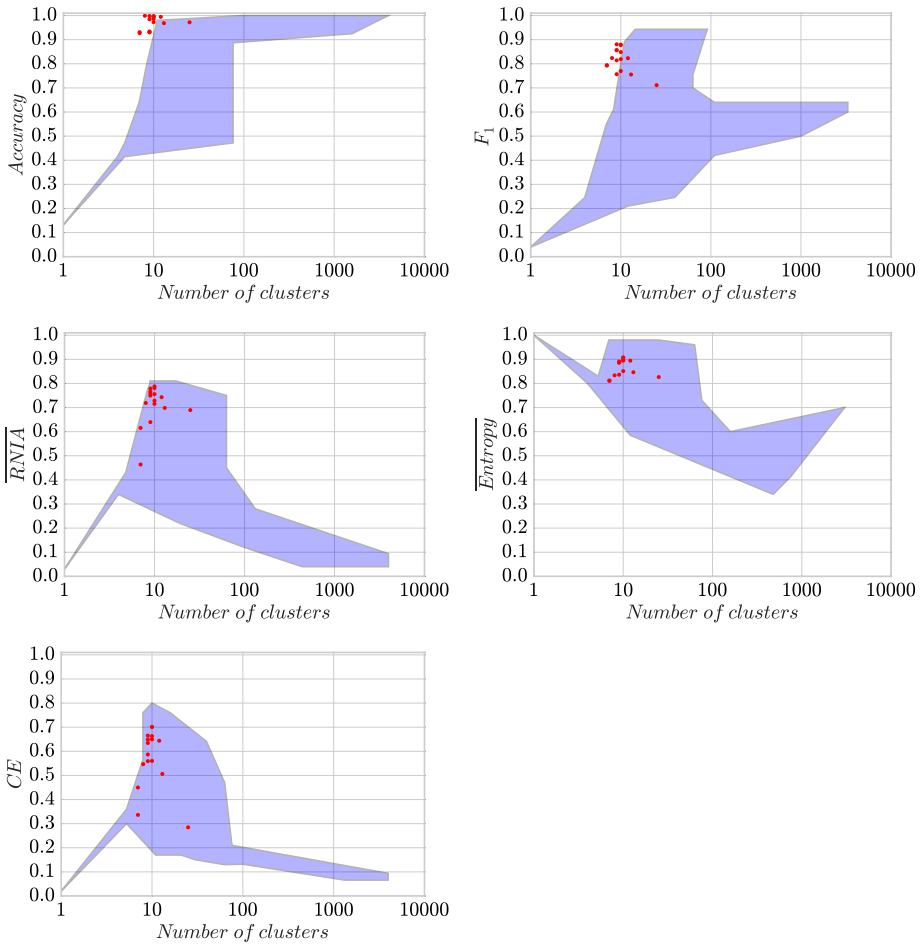
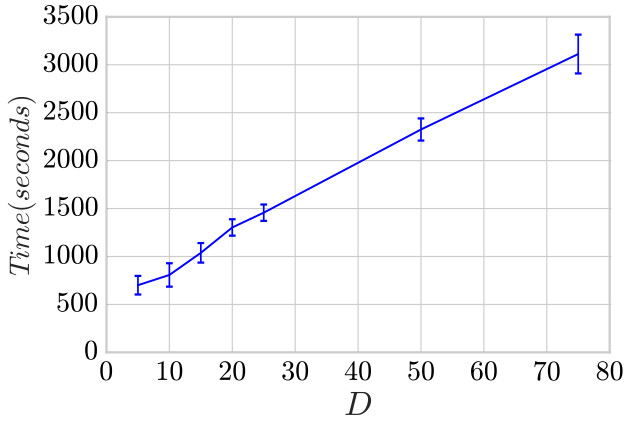
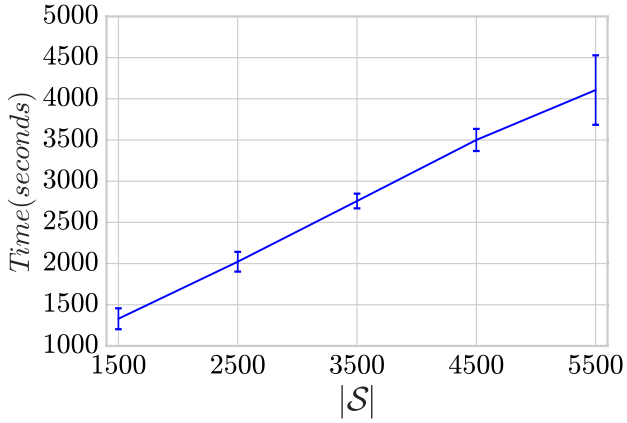


Fig. 6: *Accuracy*, F_1, $\overline{RNIA}$, $\overline{Entropy}$ and CE as a function of the number of clusters for the subspace clustering having the best fitness among 10 runs for the synthetic datasets (red dots) and region where the state-of-the-art algorithm results lay.



(a) Runtime vs. dimensionality of the dataset.



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

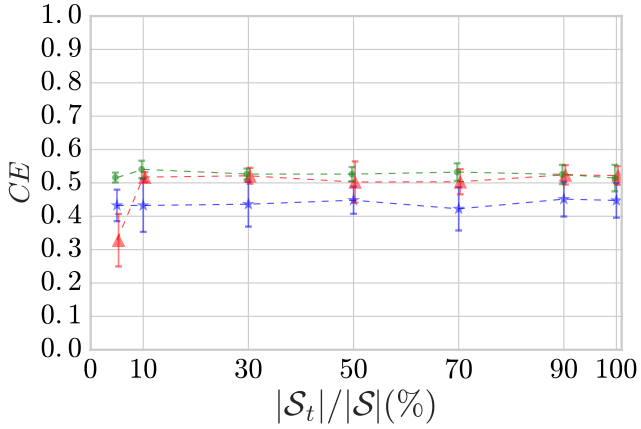
Fig. 7: Mean ± standard deviation of the runtime of ChameleoClust$^+$ (10 runs) on synthetic dataset series Dxx and Sxxxx.

been carried out by varying the parameter values one-at-a-time. For each parameter setting, the execution was repeated 10 times and the average and standard deviations of the two measures that reflect the relevance of the subspace ($\overline{RNIA}$ and CE) were computed. As in Section 3.3, we consider the three representative datasets *shape*, *pendigits* and *D20* to carry out this sensitivity analysis.

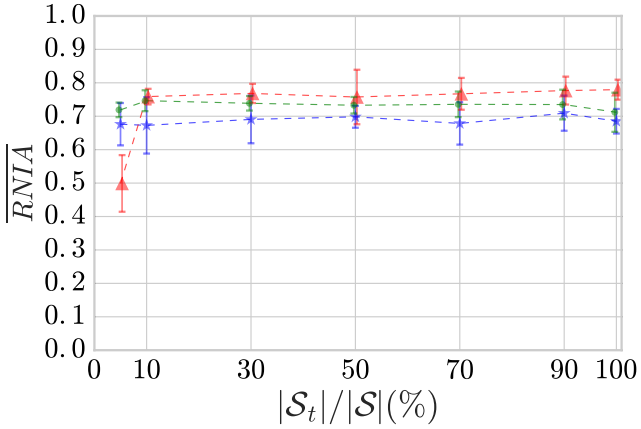
A parameter setting was used as a reference: size ω of the sliding sample $\mathcal{S}^t$ set to 10% of the dataset size, selection pressure $s = 0.5$, initial genome size $|\Gamma_{init}| = 200$ elements, mutation rate $u_m = 0.00142$, population size $N = 300$ individuals, number of generations set to 5000, and maximal number of subspace clusters $c_{max} = 20$. This corresponds to the default values specified in Section 3.3. For each parameter, the effects of changing its value were observed and are discussed in the following.

Sliding sample 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 8a and Figure 8b. 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. However, for reasonable sample sizes, the samples are representative enough of the whole dataset and good quality clusterings are obtained, as shown Figure 8. Disjoint, but more representative, samples still create a small instability in the fitness landscape, and even if an elitist selection strategy is used, as described Section 2.8, this can lead to local decreases of the fitness of the best individuals as can be observed on Figure 4a. Despite of this instability, this figure also shows that there is still a global improvement and convergence of the fitness (over the samples), and the same holds for the clustering quality (over the whole dataset) as depicted Figure 13.

Selection pressure Figure 9a and Figure 9b 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 very low ($s > 0.9$), since 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. This is consistent with the degradation of the clustering quality observed in the figures 9a and 9b. When the selection pressure is very high ($s < 0.1$), almost the complete future generation comes from the best individual of the current generation (individual having a very high reproduction probability). In this case, the genetic variability within the new generation is likely to be reduced, and Figure 9 shows a decrease of the cluster quality measures. For the smallest dataset, *shape*, the current default sample is also small, and thus is likely to be not very representative. Then, generating offspring using



(a) CE vs. dataset sample size.



(b) RNIA vs. dataset sample size.

Fig. 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 dataset sample size relative to the dataset size $\frac{|\mathcal{S}_t|}{|\mathcal{S}|}$ (percentage of the dataset size).

only the individual that has the best fitness on this sample could be the cause of the important quality degradation observed for *shape* when s is below 0.1.

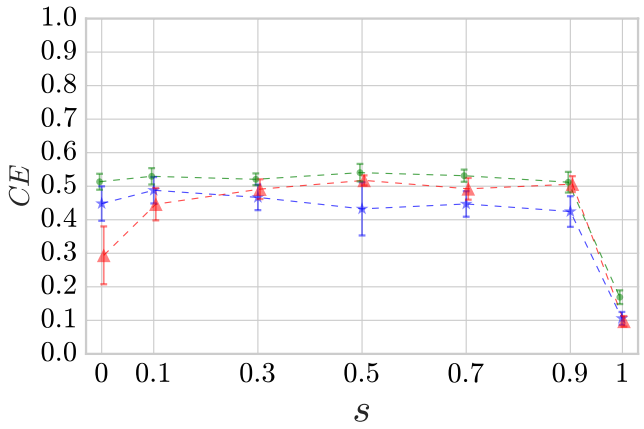
Initial genome length Set of parameter values: 10, 50, 100, 200, 300, 400, 500. As illustrated in Figure 10a and Figure 10b, the impact of the initial genome size is minor when the initial size is at least equal to 50. Indeed, ChameleoClust$^+$ genome size is evolvable and can be modified by large deletions and large duplications, consequently the initial size does not have a considerable impact on the algorithm quality. However, very small initial genomes have a high probability to stay unchanged (no mutation) as shown Figure 2, and consequently evolution tends to be slower and results tend to be poorer (for the same total number of generations).

Population size Set of parameter values: 10, 50, 100, 300, 500, 1000. As illustrated in Figure 11a and Figure 11b the larger the population the better the results. Indeed smaller populations may only explore a small portion of the solution space at each generation and tend also to have a smaller genetic variability. This leads to a slower evolution and poorer results. At the other end of the parameter range, the gain induced by having more individuals tends to become smaller as the population size increases.

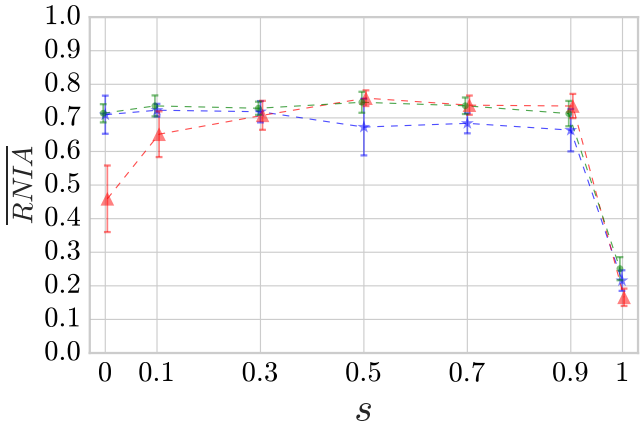
Mutation rate Set of parameter values: 0.0001, 0.00034, 0.00142, 0.00249, 0.01. We decided to test the mutation rates delimiting the suitable mutation rate range defined in Section 3.3 ($u_m = 0.00034$ and $u_m = 0.00249$), the default mutation rate ($u_m = 0.00142$) and two values outside the suitable 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 cluster quality, as showed in Figure 12. If we choose a mutation rate far outside the boundary, the subspace cluster quality decreases. This is coherent with Figure 2, the mutation rate is too low, and then the evolution process becomes very slow as most of the individuals do not mutate. While, when the mutation rate is too high, it becomes harder for the organisms to converge towards a suitable subspace clustering.

Number of generations We ran ChameleoClust$^+$ 10 times for each chosen dataset over 120000 generations. The different evaluation measures were computed each 100 generations. As illustrated in Figures 13a and 13b, 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. As discussed in the paragraph related to the number of generations in Section 3.3, the early 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 to get good results.

In this section, the impact of the choice of the parameter values, on the cluster quality, has been discussed. It should be noticed that similar effects can be observed on the fitness itself, as shown for the population size in Figure 3

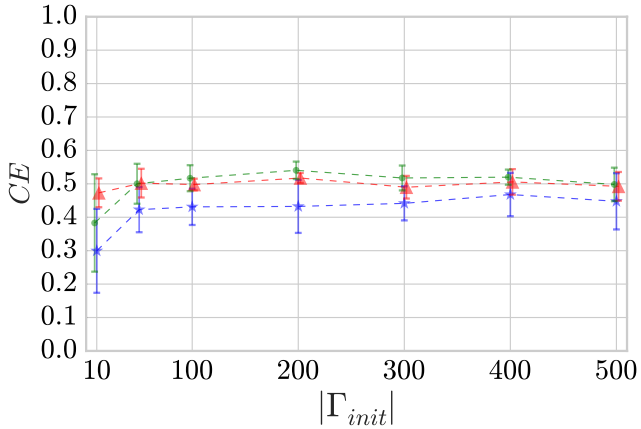


(a) CE vs. selection pressure.

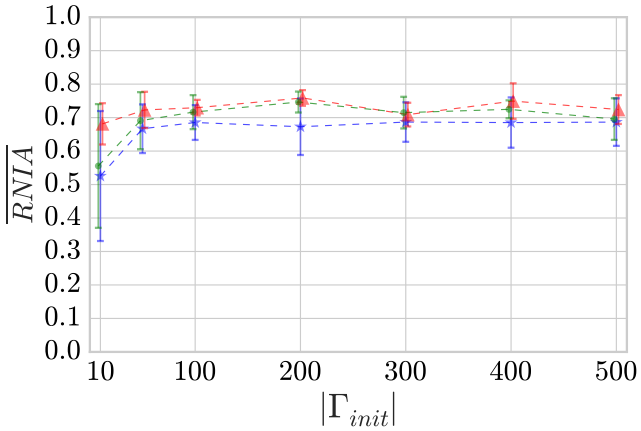


(b) RNIA vs. selection pressure

Fig. 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 selection pressure parameter s.

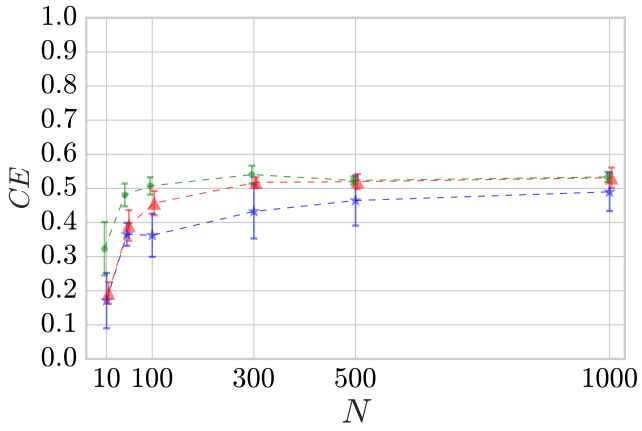


(a) CE vs. initial genome size.

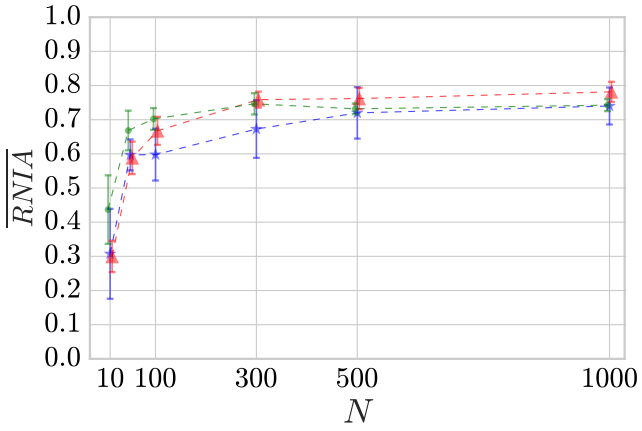


(b) RNIA vs. initial genome size.

Fig. 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 initial genome size.

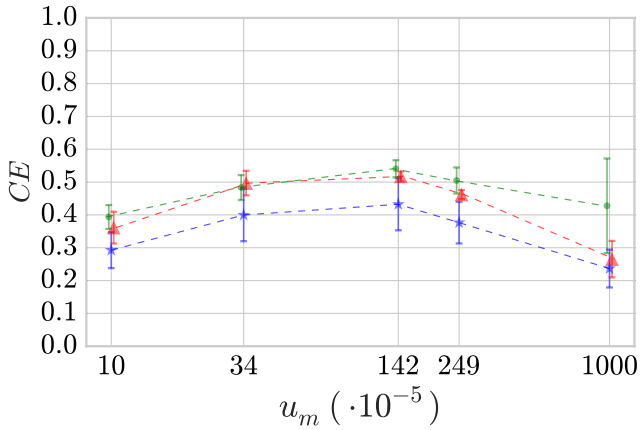


(a) CE vs. population size.

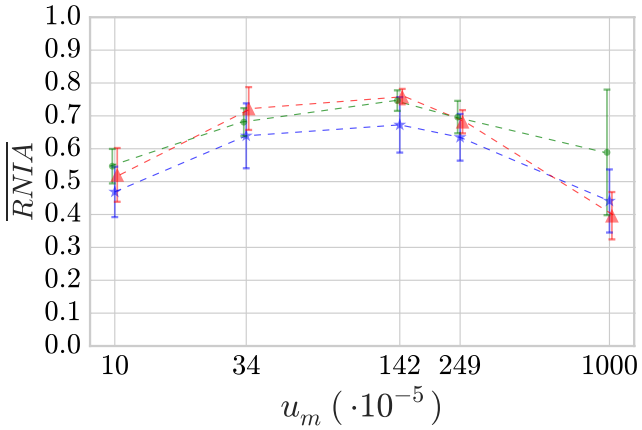


(b) RNIA vs. population size.

Fig. 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 population size N.



(a) CE vs. mutation rate.



(b) RNIA vs. mutation rate.

Fig. 12: 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.

and for the number of generations in Figure 4a. For the sake of completeness, the fitness curves obtained when modifying the other parameters are given in Appendix 7.2 (Figures 17a, 17b, 17c and 17d).

4.4 Possible alternative models

Aside from its evolvable genome size driven by large duplications and deletions, the ChameleoClust$^+$ approach relies on two other choices: an elitist reproduction method and the presence of non-functional elements. In this section, their effects on the quality measures $\overline{RNIA}$ and CE are reported using the datasets *pendigits*, *shape* and *D20* (similar trends were observed on the fitness values, and the corresponding figures are given in Appendix 7.1).

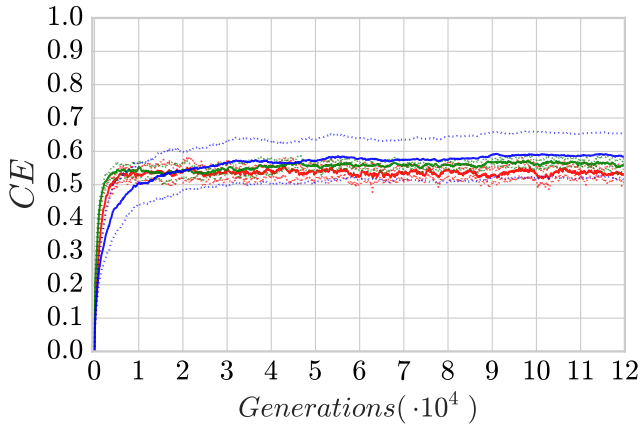
For the synthetic dataset *D20* the parameter c_{max} was set to the true number of groups (i.e., $c_{max} = 10$). For the real datasets this parameter was set to the number of classes ($c_{max} = 9$ for *shape* and $c_{max} = 10$ for *pendigits*) and other runs were performed using twice the number of classes ($c_{max} = 18$ for *shape* and $c_{max} = 20$ for *pendigits*), since the real number of groups is not necessarily equal to the number of classes.

ChameleoClust$^+$ was executed 10 times for each dataset and each value of c_{max}, using the setting described Section 3.3 for the other parameters. Figures 14a and 14b show the impact of elitism on CE and $\overline{RNIA}$. In these experiments, elitism does not seem to have a significant positive or negative effect. However, it is still a way to avoid the possible lost of a good current solution during the search. Since, according to the complexity given Section 2.9, it does not increase the cost of the generation of a new population, then there is no advantage to remove it from ChameleoClust$^+$. To test an alternative model without non-functional elements, all elements in the initial genomes were set to be functional, and the point mutations that could transform them, during evolution, into non-functional elements were simply discarded. Figures 15a and 15b report the impact of these non-functional elements on the quality measures, showing a positive effect that turns out to be significant for *pendigits* and *D20*.

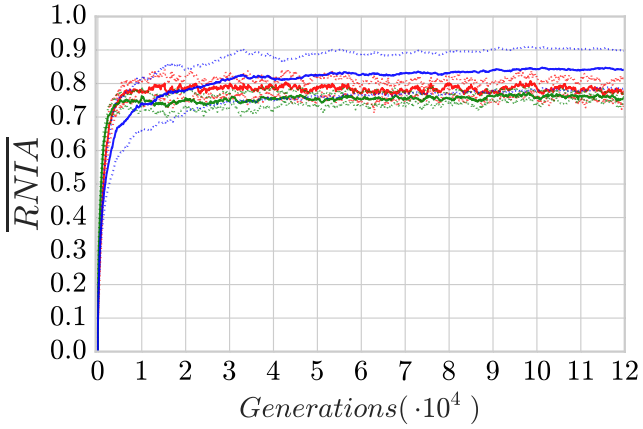
5 Related work

Many approaches have been investigated for subspace clustering in the literature using various clustering paradigms. The reader is referred for instance to (Kriegel et al., 2009), (Müller et al., 2009), and (Parsons et al., 2004) for detailed reviews and comparisons of the best methods and main categories:

– The *cell-based* approach, that defines clusters as hyper-rectangles laying in specific subspaces and containing more than a given number of objects. Clusters are usually constructed by discretizing the data space into axis-parallel cells and then aggregating promising cells. These selected cells are commonly the ones containing more objects than a threshold given as parameter. Other typical parameters are the number or the size of the cells.

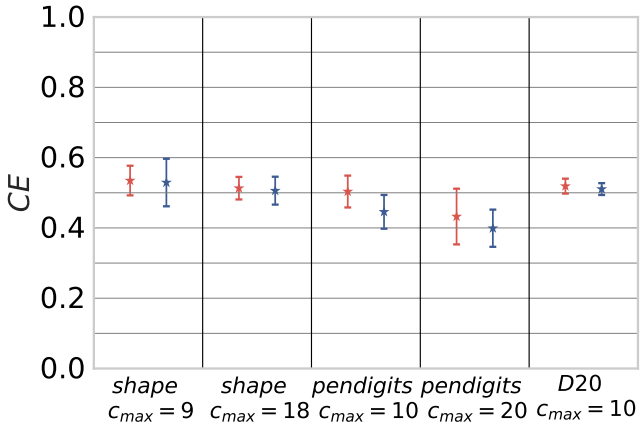


(a) Evolution of CE quality measure.

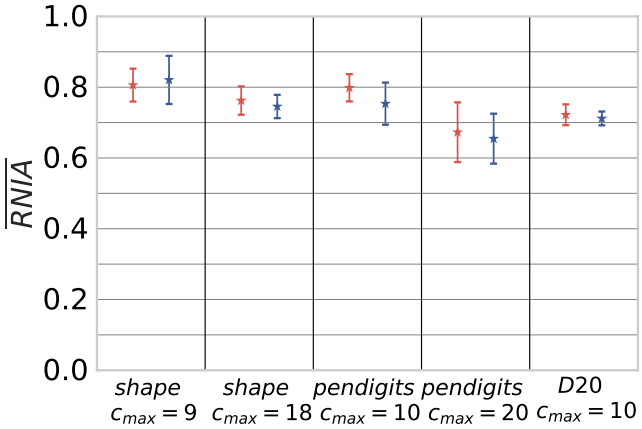


(b) Evolution of RNIA quality measure.

Fig. 13: Evolution of the mean ± standard deviation (dashed lines) of quality measures for the best individual of each generation over 10 runs of ChameleoClust$^+$ for *shape* (red), *pendigits* (blue) and *D20* (green).

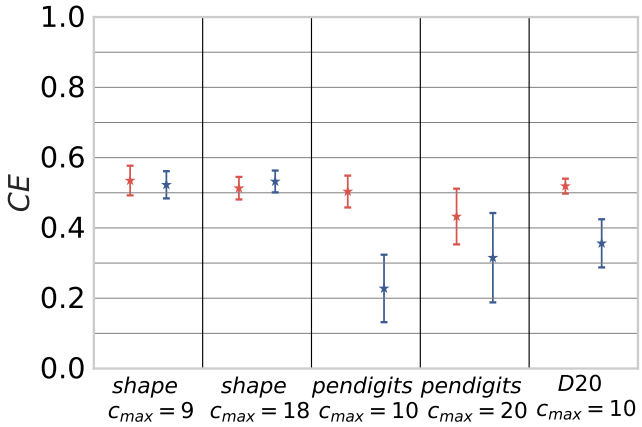


(a) Impact of elitism on CE.

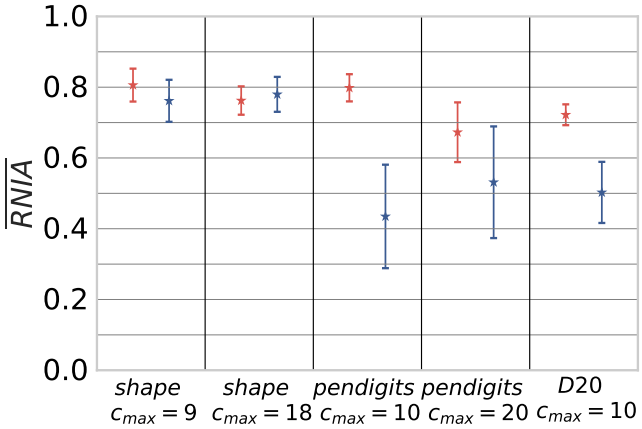


(b) Impact of elitism on RNIA.

Fig. 14: Mean ± standard deviation of quality measures for 10 runs on *shape*, *pendigits* and *D20*, with (red) and without (blue) elitism.



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



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

Fig. 15: Mean ± standard deviation of quality measures for 10 runs on *shape*, *pendigits* and *D20*, with (red) and without (blue) non-functional tuples.

– The *density-based* approach, in which clusters are dense groups of objects in space. A cluster can have an arbitrary shape, but must be separated from the other clusters by low density regions. This approach defines dense regions as regions where within a given radius a number of objects exceeding a minimum threshold can be found. Clusters are built by joining together the objects from adjacent dense regions.
– The *clustering-oriented* approach, that usually defines properties of the targeted clustering such as the expected number of clusters or the cluster average dimensionality. According to these constraints, the objects are grouped together mainly using distance-based similarity. Most of these methods tend to build hyper-spherical shaped clusters in particular subspaces.

It should be noticed that subspace clustering is also related to paradigms known as co-clustering, bi-clustering and pattern-based clustering. According to Sim et al. (2013), the main difference with subspace clustering is that these approaches consider the objects and the features of the dataset interchangeably, and cluster simultaneously objects and features exhibiting common patterns. Different survey articles have been dedicated to these paradigms, we refer the reader to Charrad and Ahmed (2011), Sim et al. (2013) and Mounir and Hamdy (2015) for a detailed presentation.

Even if many evolutionary clustering approaches exist (Hruschka et al., 2009) very few of them address the subspace clustering problem. An early approach was presented in (Sarafis et al., 2003), introducing a subspace clustering evolutionary algorithm that uses a rule-based representation to encode axis-parallel hyper-rectangular disjoint clusters. This algorithm is a member of the cell-based subspace clustering family. It uses task-specific mutation and recombination operators, and requires a non-evolutionary first stage to find promising clusters in 2D subspaces. More recently, in (Vahdat et al., 2010), a different evolutionary approach has been presented. It is also based on a first non-evolutionary clustering stage, used here to find a set of cluster candidate positions in each dimension. Next, it uses a genetic algorithm to produce subspace clusters by combining the candidate positions found at the previous step. The final stage is then to run a second genetic algorithm to find the best combination of subspace clusters to form the whole clustering of the data. This approach is related to the clustering-oriented family. The ChameleoClust$^+$ algorithm presented in this paper also falls into the clustering-oriented category, but it is a single stage and fully evolutionary approach, without any preliminary stage to identified clusters in lower dimensional spaces. In addition it relies on generic bio-like mutation operations that are not specific to the subspace clustering task. Moreover, ChameleoClust$^+$ has shown to performed well when compared to state-of-the-art subspace clustering algorithms using a reference evaluation framework.

More recently different extensions/variants of the subspace clustering problem have been investigated. For instance, in (Aksehirli et al., 2013) and (Aksehirli et al., 2015), the authors have introduced a grouping of objects based on the sharing of similar neighborhoods, instead of relying on traditional distance measures. The handling of noise has also received an increasing attention, as

in (Wang and Xu, 2016), (Vidal and Favaro, 2014) and (Soltanolkotabi et al., 2014), that tackled subspace clustering in the context of very noisy datasets. Another useful aspect, in a clustering process, is the integration of user knowledge by means of constraints. In (Hu et al., 2015), the authors have proposed such a constraint-based subspace clustering method, to guide the search for the cluster content and their subspaces.

6 Conclusion

In this paper, we presented ChameleoClust$^+$, an evolutionary algorithm for subspace clustering. Its key underlying principle is to use an evolvable genome structure to find various numbers of clusters in subspaces of different dimensionality. The genome undergoes local point mutations and is shaped by two kinds of global rearrangements: large deletions and large duplications. Beyond cluster locations, this enable to evolve the number of clusters and the number of dimensions used by each cluster.

ChameleoClust$^+$ was shown to be very competitive with respect to state-of-the-art algorithms using an evaluation framework of reference, that includes both real and synthetic datasets (varying size, number of dimensions and proportion of noise). A parameter setting method has been described, and was effective for all the datasets of the framework. In addition, a sensitivity analysis showed that the impact of the parameters related to the evolution strategy (population size, mutation rate, ...) is low for a large portion of the parameter space. The only parameter not related to evolution is the maximum number of desired clusters. To set its value, a simple procedure was given and adopted for all the datasets used in the evaluation.

Directions for future work include to investigate the impact of more complex transfers, like crossover in vertical transfer or bio-inspired horizontal transfer operations, in this evolutionary subspace clustering approach. Another promising direction of work is to extend ChameleoClust$^+$ to handle data incrementally (e.g., streaming data), a context that requires to adapt the standardization of the data. It could be handled by recomputing periodically the needed statistics over a recent part of the data, and then modifying the current core point locations by taking into account the shift and scaling induced by the new standardization parameters.

Acknowledgements

This research has been supported by EU-FET grant EvoEvo (ICT-610427). Other support: Christophe Rigotti is a member of LabEx IMU (ANR-10-LABX-0088).

Bibliography

Aggarwal, Charu C., Alexander Hinneburg, and Daniel A. Keim. 2001. On the surprising behavior of distance metrics in high dimensional space. *In* Proc. of the 8th Int. Conf. on Database Theory, Springer, pp. 420–434.

Aggarwal, Charu C., Joel L. Wolf, Philip S. Yu, Cecilia Procopiuc, and Jong Soo Park. 1999. Fast algorithms for projected clustering. *In* Proc. of the 1999 ACM SIGMOD Int. Conf. on Management of Data. ISBN 1-58113-084-8. pp. 61–72. 10.1145/304182.304188.

Aksehirli, Emin, Bart Goethals, and Emmanuel Müller. 2015. Efficient cluster detection by ordered neighborhoods. *In* International Conference on Big Data Analytics and Knowledge Discovery, Springer, pp. 15–27.

Aksehirli, Emin, Bart Goethals, Emmanuel Muller, and Jilles Vreeken. 2013. Cartification: A neighborhood preserving transformation for mining high dimensional data. *In* Data Mining (ICDM), 2013 IEEE 13th International Conference on, IEEE, pp. 937–942.

Bache, K., and M. Lichman. 2013. UCI machine learning repository.

Banzhaf, Wolfgang, Guillaume Beslon, Stephen Christensen, A. James, François Képès, Virginie Lefort, F. Julian, Miroslav Radman, and Jeremy J. Ramsden. 2006. Guidelines: From artificial evolution to computational evolution: a research agenda. Nature Reviews Genetics, **7**(9):729–735.

Beyer, Kevin, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. 1999. When is "nearest neighbor" meaningful? *In* Proc. of the 7th Int. Conf. on Database Theory. ISBN 3-540-65452-6. pp. 217–235.

Blickle, Tobias, and Lothar Thiele. 1996. A comparison of selection schemes used in evolutionary algorithms. Evolutionary Computation, **4**(4):361–394. ISSN 1063-6560. 10.1162/evco.1996.4.4.361.

Charrad, Malika, and Mohamed Ben Ahmed. 2011. Simultaneous clustering: A survey. *In* International Conference on Pattern Recognition and Machine Intelligence, Springer, pp. 370–375.

Crombach, Anton, and Paulien Hogeweg. 2007. Chromosome rearrangements and the evolution of genome structuring and adaptability. Molecular Biology and Evolution, **24**(5):1130–9. ISSN 0737-4038.

Hindré, Thomas, Carole Knibbe, Guillaume Beslon, and Dominique Schneider. 2012. New insights into bacterial adaptation through in vivo and in silico experimental evolution. Nature Reviews Microbiology, **10**:352–365.

Hruschka, Eduardo R., Ricardo José Gabrielli Barreto Campello, Alex Alves Freitas, and André Carlos Ponce Leon Ferreira de Carvalho. 2009. A survey of evolutionary algorithms for clustering. IEEE Transactions on Systems, Man, and Cybernetics, **39**(2):133–155. 10.1109/TSMCC.2008.2007252.

Hu, Han, Jianjiang Feng, and Jie Zhou. 2015. Exploiting unsupervised and supervised constraints for subspace clustering. IEEE transactions on pattern analysis and machine intelligence, **37**(8):1542–1557.

Knibbe, Carole, Antoine Coulon, Olivier Mazet, Jean-Michel Fayard, and Guillaume Beslon. 2007. A Long-Term Evolutionary Pressure on the Amount of Noncoding DNA. Molecular Biology and Evolution, **24**(10):2344–2353. 10.1093/molbev/msm165.

Kriegel, Hans-Peter, Peer Kröger, and Arthur Zimek. 2009. Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. ACM Transactions on Knowledge Discovery from Data, **3**(1):1–58. ISSN 1556-4681. 10.1145/1497577.1497578.

Mounir, Mahmoud, and Mohamed Hamdy. 2015. On biclustering of gene expression data. *In* Intelligent Computing and Information Systems (ICICIS), 2015 IEEE Seventh International Conference on, IEEE, pp. 641–648.

Müller, Emmanuel, Stephan Günnemann, Ira Assent, and Thomas Seidl. 2009. Evaluating clustering in subspace projections of high dimensional data. *In* Proc. 35th Int. Conf. on Very Large Data Bases (VLDB 2009), pp. 1270–1281.

Parsons, Lance, Ehtesham Haque, and Huan Liu. 2004. Subspace clustering for high dimensional data: A review. SIGKDD Explorations Newsletter, **6**(1):90–105.

Patrikainen, Anne, and Marina Meila. 2006. Comparing subspace clusterings. IEEE Transactions on Knowledge and Data Engineering, **18**(16):902–916.

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

Sim, Kelvin, Vivekanand Gopalkrishnan, Arthur Zimek, and Gao Cong. 2013. A survey on enhanced subspace clustering. Data Mining and Knowledge Discovery, **26**(2):332–397.

Soltanolkotabi, Mahdi, Ehsan Elhamifar, and Emmanuel J Candes. 2014. Robust subspace clustering. The Annals of Statistics, **42**(2):669–699.

Vahdat, Ali, Malcolm I. Heywood, and A. Nur Zincir-Heywood. 2010. Bottom-up evolutionary subspace clustering. *In* Proc. of the IEEE Congress on Evolutionary Computation (CEC 2010), pp. 1–8.

Vidal, René, and Paolo Favaro. 2014. Low rank subspace clustering (lrsc). Pattern Recognition Letters, **43**:47–61.

Wang, Yu-Xiang, and Huan Xu. 2016. Noisy sparse subspace clustering. The Journal of Machine Learning Research, **17**(1):320–360.

7 Appendix

7.1 Impact on fitness of the alternative models

The figures 16a and 16b show the effect of the elitist reproduction and of the presence of non-functional elements on the fitness values.

7.2 Sensitivity analysis complement

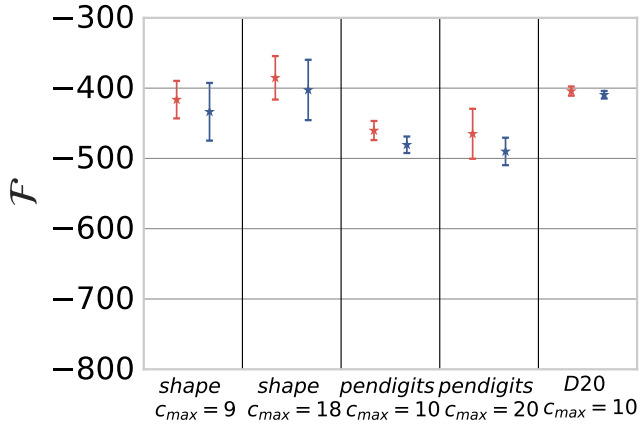
The fitness values obtained when modifying the parameters sample size, selection pressure, initial genome size and mutation rate are given Figure 17.

7.3 Results on other real datasets

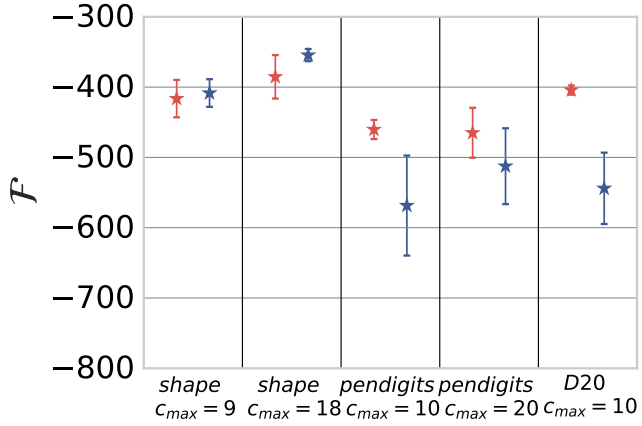
The detailed subspace clustering results on dataset *shape* are given in Section 4.1. The tables 4 to 9 given in this Appendix 7.3 report the detailed results obtained on each other real dataset of the evaluation framework. In these tables, to make the paper more self-contained for the review process, we reproduce the corresponding results from (Müller et al., 2009) and add the results obtained by ChameleoClust$^+$.

Table 4: Results on *breast* dataset: 33 dimensions, 2 classes, 198 objects, from (Müller et al., 2009) completed by results of ChameleoClust$^+$

	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 (10 runs) $\longrightarrow$	0.56		0.76		0.17		0.40		0.24		1		5.1		12.15		230	

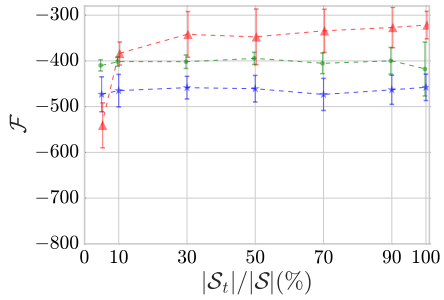


(a) Fitness with (red) and without (blue) elitism.

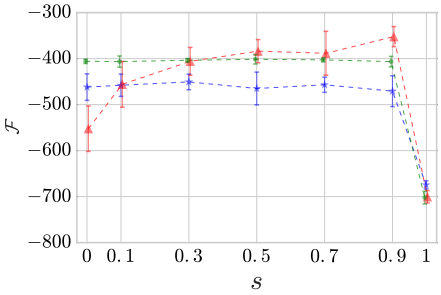


(b) Fitness with (red) and without (blue) non-functional elements.

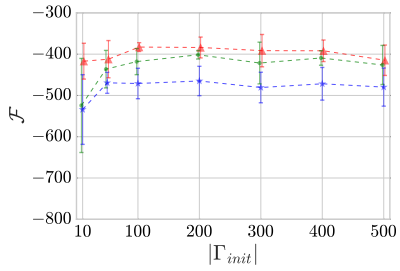
Fig. 16: Mean ± standard deviation of the fitness of the best individual of the last generation for 10 runs on *shape*, *pendigits* and *D20* under different conditions.



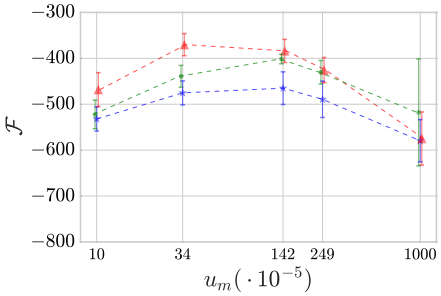
(a) Fitness vs. sample size relative to the dataset size.



(b) Fitness vs. selection pressure.



(c) Fitness vs. initial genome size.



(d) Fitness vs. mutation rate.

Fig. 17: 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.

Table 5: Results on *pendigits* dataset: 16 dimensions, 10 classes, 7494 objects, from (Müller et al., 2009) completed by results of ChameleoClust$^+$

	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	0.30	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 (10 runs) ⟶	0.64		0.68		0.43		0.67		0.63		1		11.6		10.01		4347	

Table 6: Results on *diabetes* dataset: 8 dimensions, 2 classes, 768 objects, from (Müller et al., 2009) completed by results of ChameleoClust$^+$

	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	**0.70**	**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 (10 runs) $\longrightarrow$	0.68		0.72		0.13		0.55		0.25		1		25.1		3.85		480	

Table 7: Results on *glass* dataset: 9 dimensions, 6 classes, 214 objects, from (Müller et al., 2009) completed by results of ChameleoClust$^+$

	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	**0.51**	**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 (10 runs) $\longrightarrow$	0.37		0.54		0.37		0.78		0.42		1		6.9		6.18		154	

Table 8: Results on *liver* dataset: 6 dimensions, 2 classes, 345 objects, from (Müller et al., 2009) completed by results of ChameleoClust$^+$

	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 (10 runs) $\longrightarrow$	0.62		0.64		0.14		0.47		0.11		1		24.3		2.06		179	

Table 9: Results on *vowel* dataset: 10 dimensions, 11 classes, 990 objects, from (Müller et al., 2009) completed by results of ChameleoClust$^+$

	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	0.23	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 (10 runs) $\longrightarrow$	0.39		0.40		0.15		0.60		0.42		1		28.0		5.41		910	