Applied Soft Computing 108 (2021) 107435

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Enhancing instance-level constrained clustering through differential evolution

Germán González-Almagro [a,*], Julián Luengo [a], José-Ramón Cano [b], Salvador García [a]

[a] DaSCI Andalusian Institute of Data Science and Computational Intelligence, University of Granada, Spain
[b] Dept. of Computer Science, EPS of Linares, University of Jaén, Campus Científico Tecnológico de Linares, Cinturón Sur S/N, Linares 23700, Jaén, Spain

ARTICLE INFO

Article history:
Received 7 June 2019
Received in revised form 22 February 2021
Accepted 20 April 2021
Available online xxxx

Keywords:
Constrained clustering
Instance-level
Must-link
Cannot-link
Differential evolution

ABSTRACT

Clustering has always been a powerful tool in knowledge discovery. Traditionally unsupervised, it received renewed attention when it was shown to produce better results when provided with new types of information, thus leading to a new kind of semi-supervised learning: constrained clustering. This technique is a generalization of traditional clustering that considers additional information encoded by constraints. Constraints can be given in the form of instance-level must-link and cannot-link constraints, which this paper focuses on. We propose the first application of Differential Evolution to the constrained clustering problem, which has proven to produce a better exploration–exploitation trade-off when comparing with previous approaches. We will compare the results obtained by this proposal to those obtained by previous nature-inspired techniques and by some of the state-of-the-art algorithms on 25 datasets with incremental levels of constraint-based information, supporting our conclusions with the aid of Bayesian statistical tests.

1. Introduction

One of the most widely known and studied data analysis problems is clustering. It is one of the most successful methods in the field of unsupervised learning, where there is no supervision on how the information should be handled. Clustering has been able to provide solutions in a large number of knowledge fields, such as, marketing, banking, psychology, psychiatry, astronomy, archaeology, genetics,[1] cartographic labeling [2] and image segmentation in various fields, ranging from general applications [3] to very specific ones such as in medicine [4,5].

In the literature, clustering methods are often divided into two subsets: partitional clustering and hierarchical clustering. In hierarchical clustering the result is not a partition of the data with a certain number of clusters – as in partitional clustering –, but instead a dendrogram in which there are partitions that include from the whole dataset to particular individuals [1]. A representative example of partitional clustering is the widely studied and well-known algorithm K-means [6], while for hierarchical clustering the CURE method should be highlighted [7]. In this paper we will focus on partitional clustering.

We can define partitional clustering as the task of grouping the instances of a dataset into k clusters, so that new information can be extracted from them. A dataset X is composed of n instances, each one of them described by u features. More formally, $X = \{x_1, \ldots, x_n\}$, with the ith instance noted as $x_i = (x_{[i,1]}, \ldots, x_{[i,u]})$. A typical clustering algorithm assigns a class label l_i to each instance $x_i \in X$. As a result, we obtain the set of labels $L = \{l_1, \ldots, l_n\}$, with $l_i \in \{1, \ldots, k\}$, that effectively splits X into k non-overlapping clusters c_i to form a partition called C. The criterion used to assign an instance to a given cluster is the similarity to the rest of elements in that cluster, and the dissimilarity to the rest of instances of the dataset, which can be obtained with some kind of distance measurement $d(\cdot, \cdot)$ [8].

Semi-supervised learning (SSL) is a machine learning paradigm that arises from adding incomplete information to unsupervised learning. We can divide SSL methods into two broad categories according to their objective: semi-supervised classification and semi-supervised clustering [9]. The first method has partial information about the labels, so it tries to minimize the error based on them while taking into account the distribution of unlabeled instances. The second tries to obtain better defined clusters incorporating background information to the clustering process, this is known as constrained clustering and is the main subject of the study presented in this paper [10].

Constrained clustering is an SSL learning method whose goal is to find a partition of the dataset that meets the proper characteristics of a clustering method result, in addition to satisfying a certain constraint set. It has been successfully applied in many knowledge fields, among which it is worth mentioning: advanced robotics applications [11,12], applied marketing [13], obstructive

* Corresponding author.
E-mail address: germangalmagro@ugr.es (G. González-Almagro).

https://doi.org/10.1016/j.asoc.2021.107435
1568-4946/

sleep apnea analysis [14], handwritten digits classification [15], Internet traffic classification [16], electoral district designing, [17], and lane finding in GPS data [18] among others.

Constraints can be understood in different ways, resulting in three main types of constrained clustering: cluster-level [19], instance-level [20] and feature-level constrained clustering [21]. Moreover, hybrid approaches which try to integrate different types of constraints have also been proposed [22].

In particular, we can find in the literature two types of instance-level constraints: pairwise constraints and distance-based constraints. In one hand, pairwise constraints tell us if two specific instances of a dataset must be placed in the same or in different clusters, resulting in Must-link (ML) and Cannot-link (CL) constraints respectively. On the other hand, distance-based constraints do not involve specific instances, but tell us if instances must be placed in the same or in different clusters based on a given distance measure [20]. This paper focuses on pairwise instance-level constraints (ML and CL), which will be discussed later in Section 2.1.

Regarding the degree to which the constraints have to be satisfied, we can make a distinction between the concepts of hard [18] and soft [23] constraints. Hard constraints must necessarily be satisfied in the output partition of any algorithm that makes use of them, while soft constraints are taken as a strong guide for the algorithm that uses them but can be partially satisfied in the output partition [13]. For the purposes of this paper, we will employ the latter.

Finding the optimal partition in a dataset, with respect to any kind of reasonable criteria, is known to be a **NP**-hard problem. Therefore, the incorporation of constraints may modify the complexity of the clustering problem, depending on the type of constraints used. As we will study in more depth in Section 2.2, the use of ML and CL constraints makes the constrained clustering problem **NP**-complete [11].

The constrained clustering problem can be formulated in terms of optimization, so that we can apply various optimization techniques to solve it. As mentioned earlier, it is a difficult problem to solve, so nature-inspired techniques are presented as a promising option to find quality approximate solutions. Nature is the best example of adaptive problem solving since it can apply an optimal strategy suited for each natural phenomenon [24]. Nature-inspired algorithms are designed to emulate natural optimization phenomena, such as evolution, collective behavior of animals, physics laws or even human being-related processes. There have been attempts to solve the constrained clustering problem with nature-inspired algorithms, such as the adaptation of the Biased Random-key Genetic Algorithm (BRKGA) presented in [25]. Swarm-based methods have also been applied to constrained clustering, such as the one presented in [26].

Differential Evolution (DE) is an evolution-based algorithm that has proven to be excellent in real-domain problem solving [27]. In this paper we propose the first DE application to find quality solutions for the constrained clustering problem. In particular, we will take as basis for our proposal one of the DE variants which showed excellent behavior in worldwide competitions, known as SHADE [28]. We will make use of the Random-key concept from BRKGA, along with proposing a new fitness function, to build the already mentioned DE variant. We will properly compare the results obtained by our new method with existing nature-inspired algorithms, as well as with the constrained clustering state-of-the-art.

Regarding the organization of this paper, Section 2 reviews the existing knowledge concerning constrained clustering and DE, the BRKGA algorithm and its adaptation for constrained clustering will be briefly reviewed. Afterward, the SHADE variant of DE algorithm will be presented in Section 3. In Section 4 the scheme that allow us to apply SHADE to constrained clustering, including the proposal of a new fitness function, will be presented. Sections from 5 and 6 present the experimental setup, results and their analysis respectively. Finally, in Section 7 conclusions and future work are discussed.

2. Background

In this section we present the background knowledge concerning constrained clustering (Section 2.1), its computational complexity (Section 2.2) and a brief description of some of the state-of-the-art methods for constrained clustering (Section 2.3). We will also present the basis of the DE optimization method (Section 2.4).

2.1. Constrained clustering

In most clustering applications it is common to have some kind of information about the dataset to be analyzed. In pairwise instance-level constrained clustering this information is given in the form of pairs of instances. A constraint states whether the instances which it refers to must, or must not, be assigned to the same cluster. It is possible to obtain a better result by using this type of information than by using completely unsupervised clustering algorithms. We can now formalize the two type of constraints mentioned:

- Must-link constraints $C_{=}(x_i, x_j)$: instances x_i and x_j from X must be placed in the same cluster.
- Cannot-link constraints $C_{\neq}(x_i, x_j)$: instances x_i and x_j from X cannot be assigned to the same cluster.

The goal of constrained clustering is to find a partition (or clustering) of k clusters $C = \{c_1, \ldots, c_k\}$ of the dataset X that ideally satisfies all constraints in the constraint set. As in the original clustering problem, it must be fulfilled that the sum of instances in each cluster c_i is equal to the number of instances in X, which we have defined as $n = |X| = \sum_{i=1}^{k} |c_i|$.

Knowing how a constraint is defined, ML constraints are an example of an equivalence relation; therefore, ML constraints are reflexive, transitive and symmetric. This way, given constraints $C_{=}(x_a, x_b)$ and $C_{=}(x_b, x_c)$ then $C_{=}(x_a, x_c)$ is verified. In addition, if $x_a \in c_i$ and $x_b \in c_j$ are related by $C_{=}(x_a, x_b)$, then $C_{=}(x_c, x_d)$ is verified for any $x_c \in c_i$ and $x_d \in c_j$ [20].

It can also be proven that CL constraints do not constitute an equivalence relation. However, analogously, given $x_a \in c_i$ and $x_b \in c_j$, and the constraint $C_{\neq}(x_a, x_b)$, then it is also true that $C_{\neq}(x_c, x_d)$ for any $x_c \in c_i$ and $x_d \in c_j$ [20].

2.2. The feasibility problem

Given that constrained clustering adds a new element to the clustering problem, we must consider how this element affects the complexity of the problem. The feasibility problem for non-hierarchical instance-level constrained clustering was defined in [11] as in Definition 1.

Definition 1 (*Feasibility Problem*)**.** Given a dataset X, a constraint set CS, and the bounds on the number of clusters $k_l \leq k \leq k_u$, does there exist a partition C of X with k clusters such that all constraints in CS are satisfied? [11]

In [11] it is proven that, when $k_l = 1$ and $k_u \geq 3$, the feasibility problem for constrained clustering is **NP**-complete, by reducing it from the Graph K-Colorability problem (it is also proven that it is not harder, so both have the same complexity). Table 1 shows the complexity of the feasibility for different types of constraints.

Table 1
Feasibility problem complexity [11].

Constraints	Complexity
Must-Link	**P**
Cannot-Link	**NP**-complete
ML and CL	**NP**-complete

These complexity results show that the feasibility problem with CL constraints is intractable and hence constrained clustering is intractable too. For more details on the complexity of constrained clustering see [11].

Intractable problems are hard to solve with deterministic and exact methods. That is the reason why nature-inspired algorithms constitute good approaches to find quality solutions to the constrained clustering problem.

2.3. State-of-the-art methods

The first adaptation of a classic clustering method for constrained clustering was proposed in [18]. It involved modifying the widely studied K-means algorithm to take into account instance-level constraints: the already known ML and CL. This method was named COP-kmeans, it introduces a modification to the assignation rule of instances to clusters of the K-means algorithm so that an instance can be assigned to a cluster only if the assignment does not violate any constraint.

In [29] Constrained Evidential c-means (CECM), a variant of the Evidential c-means (ECM [30]) algorithm is proposed, within the fuzzy clustering family of methods. The particularity of this algorithm is that the membership of instances to a cluster is defined by a probabilistic belief function. This method redefines constraints from the point of view of belief functions and includes them in the cost function.

A modification of the Constrained Vector Quantization Error algorithm (CVQE [11]) is proposed in [31]. The CVQE algorithm proved to produce high quality results, at the cost of a very high computational complexity. Linear CVQE (LCVQE) introduces a modification of the cost function of CVQE to make it more intuitive and less computationally complex. The experimentation resulted in a dramatic improvement of clustering quality over both noisy and clean constraint sets.

Two Views Clustering (TVClust) and Relation Dirichlet Process-Means (RDPM) were proposed in [32]. TVClust is able to incorporate the constraints into the clustering problem by making a soft interpretation of them. The authors model the dataset and constraints in different ways, perform clustering methods on them and try to find a consensus between both interpretations. Using this model as a basis, the authors derive the deterministic algorithm RDP-means. This method can be viewed as an extension of K-means that includes side information (constraints) and has the property that the number of clusters (k) does not need to be specified.

2.4. Differential evolution

Evolution-based methods comprise a group of algorithms developed on the basis of the natural laws of evolution. In this type of techniques a population of individuals – representing potential solutions to the problem – is usually employed, which compete and combine across a certain number of generations so that the population evolves in such a way that only the best individuals, and thus the best solutions, remain at the end. This type of techniques involves the development of a series of operators that allow us to simulate the evolution process, such as crossover, mutation and selection operators [24].

DE, which emerged in [33], is a good example of evolution-based methods. From that point on, the reputation of DE was consolidated in conferences and competitions where it was able to obtain competitive results that rivaled those achieved by the state-of-the-art. In particular, DE excelled in real-coded optimization problems. A highly detailed analysis of DE and its variants can be found in [27].

DE arose as a variant of population-based evolutionary algorithms, its goal being to perform an intelligent search in the solution space of the problem in the most optimized and effective way possible.

Like most evolutionary algorithms, DE uses a population of individuals P, where each individual p_i is a vector of real values $p_i = \{p_{[i,1]}, \ldots, p_{[i,v]}\}$, with v being the number of features of each individual. Each p_i serves as parameter for the function to be optimized. These individuals are considered solutions to the problem.

To guide the optimization process, we use a fitness function. The task of DE is to seek the parameter vector p_i^* that minimizes a function such that $f(p_i)(f : \Omega \subseteq \Re^v \rightarrow \Re)$, where $f(p_i^*) < f(p_i)$ for all $p_i \in \Omega$ with Ω a nonempty finite set that is the search domain.

The optimization process of DE the works through a simple cycle of stages, presented in Fig. 1, that we can summarize as follows:

- **Initialization of Vectors**: DE begins with a randomly initialized population of a given number of v-dimensional arrays of parameters. These vectors constitute candidate solutions for the problem to be solved.
- **Difference-vector-based mutation**: DE introduces diversity into the population via the mutation operator. This operator is applied to each individual of the population, known in this context as the parent vector. It combines the parent vector with two randomly selected vectors from the population by applying an arithmetic operator to them. The result of this process is a mutant vector.
- **Crossover/Recombination**: The mutant vector exchanges some components with its associated parent to form a trial vector, which will be the candidate to replace the parent vector. Several strategies exist to generate this trial vector, which we will not delve into; however, they can be found in [27].
- **Selection**: The newly generated trial vector is compared with its associated parent using the fitness function. If the trial vector is better than the parent, then it will replace the parent in the next generation. It is worth noting that, following this strategy, the population never degenerates but only gets better or remains the same.

Even though DE was a major breakthrough in terms of real optimization, many variants have been developed since its invention that provide better results. We will focus on the SHADE variant, which adaptively optimizes some of the parameters of DE. To carry out this task SHADE uses a historic record of values for these parameters that allows it to set them in favor of the optimization process, based on how good the algorithm behaved with certain parameter values in the past (see Section 3).

2.5. Brief review of the BRKGA algorithm

Genetic Algorithms (GAs) emerged as one of the first metaheuristics inspired by the concept of natural selection and evolution. It is one of the most studied and successful evolutionary algorithms, thanks in part to its simplicity and ease of implementation [24,34].

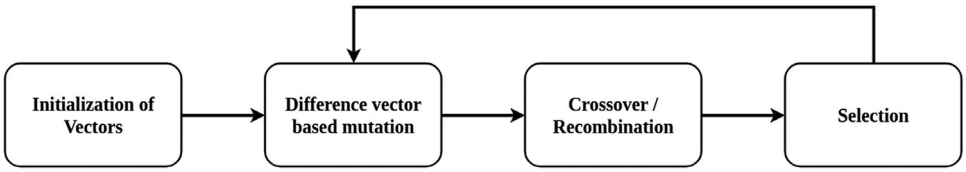


Fig. 1. The optimization process of DE [27].

In a GA, a population P of a given number of solutions $p_i = \{p_{[i,1]}, \ldots, p_{[i,v]}\}$ is first initialized; these solutions are also called chromosomes or individuals. Each element of a chromosome is called gene and the value it takes is known as allele. To simulate the natural processes of evolution and natural selection, the set of evolutionary operators that GAs make use of is similar to that used by DE: crossover, mutation and selection. The crossover operator consists in combining two different chromosomes (parents) from the population and obtaining a new chromosome (offspring) which inherits alleles for its genes from both parents. The mutation operator is used by GAs to avoid local optima, and it consists in introducing random changes into the population to favor diversity. The selection operator is used to determine whether a chromosome will survive in the next generation, and to accomplish this task it employs a fitness function that assigns a value to each chromosome to assess its quality as a solution for the problem. GAs apply these operators through a series of generations to hopefully obtain a solution for the problem to solve that is close to the optimal [24].

The Random-Key Genetic Algorithm (RKGA) emerged in [35] as a solid approach to high constrained problems. It uses vectors of random numbers to represent solutions to the problem (individuals), which are used as keys to obtain the actual solutions. Random-key vectors are sampled from the space $[0, 1)^v$, which is used as a surrogate space for the literal solution space. Points in the random-key space are transferred to the literal solution space via a decoder for fitness evaluation. A decoder is a deterministic procedure that allows us to map random-key vectors to the literal solution space. We can build a decoder that always provides as a result a feasible solution to the problem we want to solve, thus avoiding the feasibility problem that we found in classic GAs.

The Biased Random-key Genetic Algorithm (BRKGA) was first proposed in [36] as a variant of the RKGA. In the BRKGA optimization process, first, the population P of $|P|$ vectors of random-keys is initialized; this will be the initial generation. Each random-key vector p_i has v random-keys in it. The population is sorted by the fitness value f_i of each p_i to select the first P_e individuals, this is, the elite of the population. The elite will be preserved in the next generation without modification. To introduce diversity, a number P_m of new random-keys vectors are also included in the next generation. The remaining individuals of the new generation ($|P| - P_e - P_m$) are obtained through crossovers between elite and non-elite parents.

It should be noted that in BRKGA there is a clear differentiation between problem-dependent and problem-independent elements. The independent part of the algorithm has no knowledge of the problem it solves, it is limited to performing the operations related to the population optimization process. The dependent part of the problem is composed of the decoder and the fitness function, which allow us to obtain actual solutions for the problem and its evaluation. Therefore, to specify a BRKGA optimization scheme it is only necessary to define the decoder and the fitness function [36].

2.6. BRKGA optimization scheme for constrained clustering

An adaptation of BRKGA for constrained clustering was proposed in [25], we will refer to it as BRKGA$_{CC}$.

Table 2
Random-key decodification example [25].

Index (instance)	1	2	3	4	5	6	7	8	9	10
Random-key	0.12	0.37	0.66	0.56	0.00	0.97	0.23	0.25	0.15	1.00
Cluster (label)	1	2	2	2	1	3	1	3	1	3

In this scheme each random-key vector has n random-keys in it ($v = n$). The decoder divides the interval $[0, 1]$ in k intervals, so there exists a correspondence between each random-key (instance x_i) and the integer (label l_i) corresponding to the interval which it is found in. Table 2 shows an example of random-key decoding for a dataset with 10 instances ($n = 10$) and $k = 3$. Note that extreme values 0 and 1 can also appear in a random-key vector.

Once the decoded solution has been obtained, the fitness value of the solution is computed as in Eq. (1).

$$f_i = z_i + \overbrace{(\mu * n * \text{infeasibility}_i)}^{\text{penalty}}, \tag{1}$$

where μ is a high value, infeasibility$_i$ is the number of non-satisfied constraints and z_i is the within-cluster-sum-of-squares which can be computed as in Eq. (2).

$$z_i = \sum_{c_j \in C_i} \left[\frac{\sum_{x_a, x_b \in c_j} d^2(x_a, x_b)}{|c_j|} \right], \tag{2}$$

where C_i is the partition of dataset X defined by solution (individual) p_i, and (x_a, x_b) are instances in the cluster c_j such that $a \neq b$ and the Euclidean distance ($d^2(\cdot, \cdot)$) between each pair of instances in C_i is included in the sum only once.

In [25], the authors also added a new element to BRKGA, a local search procedure. This local optimization method is applied to each decoded individual in each generation, but its results are not transferred to the individual, so that diversity is maintained. Thus, the individuals produced by the local search are only used to update the best solution found so far if needed. For more details on BRKGA$_{CC}$ see [25].

3. The SHADE optimization method

Success History Based Adaptive Differential Evolution (SHADE) is an optimization method based on differential evolution. Proposed in [37], it was an improvement for the JADE model, presented in [38].

Unlike JADE, which only considers the parameters used in the previous generation, SHADE uses a historical record of successful parameters as a mechanism for adapting parameters involved in the process of creating new generations.

Since SHADE is an evolution-based algorithm, it shares elements with BRKGA (Section 2.5), so we will use a similar notation to define its elements. We will note the population as P_G, formed by $|P|$ individuals with the form $p_{[i,G]} = \{p_{[i,G,1]}, \ldots, p_{[i,G,v]}\}$, with G indicating the generation to which p_i belongs.

Table 3
Historical memory M_{CR}, M_F used by SHADE [37].

Index	1	2	···	$\|M\|-1$	$\|M\|$
M_{CR}	$M_{[CR,1]}$	$M_{[CR,1]}$	···	$M_{[CR,\|M\|-1]}$	$M_{[CR,\|M\|]}$
M_F	$M_{[F,1]}$	$M_{[F,1]}$	···	$M_{[F,\|M\|-1]}$	$M_{[F,\|M\|]}$

3.1. SHADE operators

As in differential evolution, SHADE employs mutant generation, crossover and selection operators to explore the space of solutions and bring in new generations of individuals.

The mutation strategy used by SHADE is known as current-to-*p*best/1, and the expression that generates new individuals is described in Eq. (3).

$$m_{[i,G]} = p_{[i,G]} + F_i * (p_{[p\text{best},G]} - p_{[i,G]}) + F_i * (p_{[r1,G]} - p_{[r2,G]}), \quad (3)$$

where $m_{[i,G]}$ is the mutant vector, which is generated with an individual $p_{[i,G]}$ from the population serving as a starting point. Indices $r1$ and $r2$ are random values in the range $[0, |P|]$, different from i and from each other. The individual $p_{[p\text{best},G]}$ is randomly selected from among the best $|P| \times p\text{best} \mid p\text{best} \in [0, 1]$ individuals in the population. This way, the parameter $p$best controls the greediness of the mutation strategy. The parameter F_i defines the magnitude of the operator.

After generating the mutant vector $m_{[i,G]}$, it is combined with the parent $p_{[i,G]}$ vector by means of a crossover operator; the result is the trial vector $t_{[i,G]} = \{t_{[i,G,1]}, \ldots, t_{[i,G,v]}\}$. SHADE uses the Binomial crossover operator, which is defined by Eq. (4).

$$t_{[i,G,j]} = \begin{cases} m_{[i,G,j]} & \text{if } \text{rand}[0,1) \leq CR_i \text{ or } j = j_{rand} \\ p_{[i,G,j]} & \text{otherwise} \end{cases}, \quad (4)$$

where rand$[0, 1)$ is a random number selected from a normal distribution in the range $[0, 1)$, j_{rand} is a random integer selected from the range $[1, v]$, and $CR \in [0, 1]$ is the crossover ratio.

The SHADE selection operator determines which individuals will survive in the next generation. It compares each parent $x_{[i,G]}$ with the trial vector $t_{[i,G,j]}$, generated based on it. The best one of these two individuals will survive for the next generation. Eq. (5) shows this concept.

$$p_{[i,G+1]} = \begin{cases} t_{[i,G]} & \text{if } f(t_{[i,G]}) \leq f(p_{[i,G]}) \\ p_{[i,G]} & \text{otherwise} \end{cases}. \quad (5)$$

To maintain diversity, SHADE can make use of an external archive A, in which parents $p_{[i,G]}$ who were replaced by their associated trial vector $t_{[i,G]}$ are saved. To make use of the archive, we will consider that the individual $p_{[r2,G]}$ from Eq. (3) is selected from $P \cup A$. The archive size is the same as the population size. When the size of A exceeds the size of P, random individuals are selected for removal to make room for new ones.

The parameters $p$best, F_i and CR_i are difficult to adjust and their selection is not trivial, as they largely determine the success of the optimization process. These are the parameters that SHADE adaptively optimizes using the mentioned history record.

The SHADE method stores in memory a structure M with $|M|$ entries for parameters F_i and CR_i, as shown in Table 3. This structure is initialized with the value 0.5 for all entries.

In each generation, parameters CR_i and F_i are calculated on the basis of the existing history by applying Eqs. (6) and (7):

$$CR_i = \text{randn}_i(M_{[CR,r_i]}, 0.1), \quad (6)$$

$$F_i = \text{randc}_i(M_{[F,r_i]}, 0.1), \quad (7)$$

where randn(μ, σ^2) and randc(μ, σ^2) are random values from normal and Cauchy distributions respectively, with mean μ and variance σ^2. When a value for CR_i outside of the range $[0, 1]$ is generated, it is replaced by the corresponding limit value. If $F_i > 1$, then it is truncated to 1; conversely, if $F_i < 0$, then Eq. (7) is applied as many times as needed to obtain a legal value.

SHADE keeps two auxiliary sets S_{CR} and S_F to store the CR_i and F_i values that were successfully used to generate a trial vector that replaced the parent. At the end of each generation, the content of the memory M is updated following Eqs. (8) and (9).

$$M_{[CR,h,G+1]} = \begin{cases} \text{mean}_{WA}(S_{CR}) & \text{if } S_{CR} \neq \emptyset \\ M_{[CR,h,G]} & \text{otherwise} \end{cases}, \quad (8)$$

$$M_{[F,h,G+1]} = \begin{cases} \text{mean}_{WL}(S_F) & \text{if } S_F \neq \emptyset \\ M_{[F,h,G]} & \text{otherwise} \end{cases}. \quad (9)$$

The h $(0 \leq h \leq |M|)$ index specifies the position of the memory M to be updated. This index is initialized to 0 at the beginning of the optimization process and increased by one after every generation. When $h \geq |M|$ then h is reset to 1. It is worth noting that, when there is a generation with no trial vectors successfully replacing their parents, no update of M is done.

In Eq. (8), the term $\text{mean}_{WA}(S_{CR})$ is the weighted mean, which is computed following Eqs. (10) and (11), proposed in [39] to prevent CR from converging to small values.

$$\text{mean}_{WA}(S_{CR}) = \sum_{i=1}^{|S_{RC}|} \omega_i * S_{[RC,i]}, \quad (10)$$

$$\omega_i = \frac{\Delta f_i}{\sum_{i=1}^{|S_{RC}|} \Delta f_i}, \quad (11)$$

where $\Delta f_i = |f(t_{[i,G]}) - f(p_{[i,G]})|$, which is the amount of improvement obtained from the trial vector with respect to the parent.

In Eq. (9) the term $\text{mean}_{WL}(S_F)$ refers to the weighted Lehmer mean, which is computed as in Eq. (12) (proposed in [37]).

$$\text{mean}_{WL}(S_F) = \frac{\sum_{i=1}^{|S_F|} \omega_i * S_{[F,i]}^2}{\sum_{i=1}^{|S_F|} \omega_i * S_{[F,i]}}. \quad (12)$$

Unlike parameters S_{CR} and S_F, the parameter $p$best from Eq. (3), used to set the greediness of the mutation strategy, is not included in the adaptive optimization process. However, it is not static: it is calculated for every individual p_i of the population following Eq. (13).

$$p\text{best}_i = \text{randn}[2/N, 0.2], \quad (13)$$

such that there is always at least two individuals to choose between.

4. SHADE optimization scheme for constrained clustering

In this section, we present an optimization scheme for the SHADE algorithm when applied to constrained clustering. We will call this new approach SHADE$_{CC}$. To begin with, we will use random-keys to create the individuals of the population; this way, each individual is defined as a vector of n random-keys ($v = n$). Additionally, we will also make use of the initialization method and the decoder found in Section 2.6. However, we propose a new fitness function to evaluate the quality of the individuals of the population, Eq. (14), whose element z_i is obtained as in Eq. (2).

$$f_i = z_i * (\text{infeasibility}_i + 1). \quad (14)$$

Note that in Eq. (14) no parameter is involved that cannot be calculated from the dataset or constraints, whereas in Eq. (1)

the μ parameter must be calculated and optimized for each individual problem. Therefore with Eq. (14) a better generalization capability is obtained.

We found that with the fitness function defined in (1) there is no competition between the penalty term and the within-cluster-sum-of-squares term, because the penalty is always significantly larger than it or zero. This can bias the exploration of the solution space, restricting it in practice to those that satisfy all the constraints, even if the within-cluster-sum-of-squares is still improvable by moving two instances involved in a constraint to different clusters without violating that constraint.

With Eq. (14) we try to find a trade-off between the within-cluster-sum-of-squares and the penalty term, allowing solutions that violate a certain number of constraints to compete with those who violate a smaller number of them but score a better within-cluster-sum-of-squares.

To make these concepts clear we consider a toy dataset and three partitions over it. The dataset from Fig. 2 contains 100 instances and a single ML constraint between two instances from the same class. Partitions C_0 – displaying the true labels – and C_1 do not violate the constraint (infeasibility $= 0$), whereas C_2 does violate it (infeasibility $= 1$). However, C_2 shows a possibility, for later iterations, of adding both instances to their nearest cluster while also satisfying the ML constraint.

Table 4 shows the fitness values for each partition from Fig. 2. We refer to the function presented in Eq. (1) as f_1 and to the function presented in Eq. (14) as f_2.

Results in Table 4 are a clear example of the differences between the two fitness functions discussed. In the general case, by using f_2 the penalty for violating constraints is proportional to the within-cluster-sum-of-squares—the lower it is, the lower the penalty. This is not the case with f_1, whose penalty term is independent of the within-cluster-sum-of-squares, which can result in a difference of several orders of magnitude between partitions satisfying different amounts of constraints.

As in [25] we apply a local optimization procedure to the individuals of the population, but without transferring its results to the original individual in order to maintain diversity. The difference is that we do not apply it to the whole population but only to a number P_e of individuals considered as the elite of the population, with the aim of preserving a good exploration–exploitation trade-off. The percentage of individuals considered as elite must be determined for each case.

Algorithm 1 describes the overall SHADE$_{CC}$ optimization process, including the local search procedure presented in Algorithm 2. The goal of this procedure is to locally improve solutions (individuals from P) in a non-exhaustive way. To do so, it randomly chooses an instance from the dataset and iteratively assigns it to different clusters. When improvement in the fitness function is detected, the change is transferred to the solution, when there is no possible improvement the local optimization process stops.

Algorithm 1: SHADE$_{CC}$

Input: Dataset X, constraint sets $C_=$ and $C_{\neq}$, population size p_{size}, elite size P_e, number of clusters k.

`// Initialization phase`

[1] $G \leftarrow 0$

[2] Initialize population $P_0 \leftarrow \{p_{[1,0]}, \cdots, p_{[p_{size},0]}\}$

[3] Initialize all values in M_{CR}, M_F to 0.5

[4] $A \leftarrow \emptyset$; $h \leftarrow 1$

`// Main loop`

[5] **while** *Termination criteria are not met* **do**

[6] $S_{CR} \leftarrow S_F \leftarrow \emptyset$

[7] $P_G \leftarrow$ `Sort`(P_G) `// Sort the population`

`// Apply an LS procedure to the elite of the population`

[8] `LocalSearch`$(\{p_{[1,G]}, \cdots, p_{[p_e,G]}\})$

[9] **for** $i \in [1, p_{size}]$ **do**

[10] $r_i \leftarrow$ randInt$[1, |M|]$

[11] $CR_{[i,G]} \leftarrow$ randn$_i(M_{[CR,r_i]}, 0.1)$

[12] $F_{[i,G]} \leftarrow$ randc$_i(M_{[F,r_i]}, 0.1)$

[13] $p_{[i,G]} \leftarrow$ rand$[N/2, 0.2]$

[14] Generate trial vector $t_{[i,G]}$ by current-to-$p$best/1/bin

[15] **if** $f(t_{[i,G]}) \leq f(p_{[i,G]})$ **then**

[16] $p_{[i,G+1]} \leftarrow t_{[i,G]}$

[17] **else**

[18] $p_{[i,G+1]} \leftarrow p_{[i,G]}$

[19] **end**

[20] **if** $f(t_{[i,G]}) < f(p_{[i,G]})$ **then**

[21] $p_{[i,G]} \rightarrow A$; $CR_{[i,G]} \rightarrow S_{CR}$; $F_{[i,G]} \rightarrow S_F$

[22] **end**

[23] **end**

`/* Whenever` $|A| > |P|$`, randomly select an individual from` A `to be deleted so that` $|A| \leq |P|$ `*/`

[24] **if** $S_{CR} \neq \emptyset$ ***and*** $S_F \neq \emptyset$ **then**

[25] Update $M_{[CR,h]}$ and $M_{[F,h]}$ based on S_{CR} and S_F

[26] $h \leftarrow (h + 1) \mod |M|$

[27] **end**

[28] $G \leftarrow G + 1$

[29] **end**

5. Experimental setup

For our experiments we will compare the results obtained by BRKGA$_{CC}$ and SHADE$_{CC}$ over 25 datasets and 3 constraint sets for each one of them. Most of these datasets can be found at the Keel-dataset repository[1] [40], though some of them have been obtained via scikit-learn python package[2] [41]. We also include 3 artificial datasets in our analysis, namely: *Circles*, *Moons* and *Spiral*, which can be found at GitHub.[3] Table 5 displays a summary of every dataset.

Classification datasets are commonly used in the literature to test constrained clustering algorithms; they enable us to generate constraints with respect to the true labels (see Section 5.1). They also facilitate an easy evaluation of the quality of the algorithm by means of measures like the Adjusted Rand Index (see Section 5.2).

We have selected datasets to cover a wide range of the three main features a dataset can present. The number of instances covers a range from 47 to 990, the number of features does so for the range 2 to 90 and finally the number of classes goes from 2 to 15. It is worth pointing out the addition of non-convex artificial datasets (the already mentioned Circles, Moons and Spiral). Because the fitness function proposed in Eq. (14) is not able to naturally identify non-convex shapes, these three datasets let us examine the capability of SHADE$_{CC}$ to incorporate constraints into

[1] https://sci2s.ugr.es/keel/category.php?cat=clas.

[2] https://scikit-learn.org/stable/datasets/index.html.

[3] https://github.com/GermangUgr/SHADE_CC.

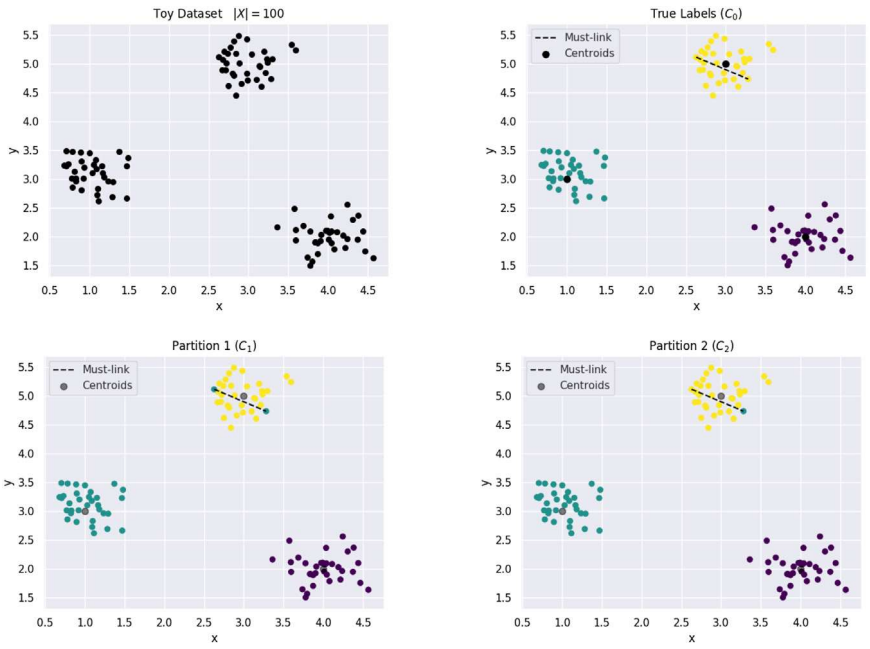


Fig. 2. Three partitions over a toy dataset.

Table 4
Expression and value of fitness functions over three partitions ($\mu = 10$).

Partition	Expression		Value	
	f_1	f_2	f_1	f_2
C_0	z_0	z_0	0.457	0.457
C_1	z_1	z_1	0.540	0.540
C_2	$z_2 + (\mu * N *$ infsblt)	$z_2 * ($infsblt $+ 1)$	$0.503 + 1000 = 1000.503$	$0.503 * 2 = 1.005$

Algorithm 2: Local Search

Input: Dataset X, constraint sets $C_=$ and $C_{\neq}$, decoded random-key vector (solution) S, number of clusters k.

```
while improvement do
    improvement ← false
    s_i ← Select random object from S
    // Random shuffle labels set
    RSL ← RandomShuffle({1, ···, K})
    for l ∈ RSL do
        S' ← S
        // Move object s_i to the cluster associated
           with label l
        S'[s_i] ← l
        if f(S') < f(S) then
            S ← S'
            improvement ← true
        end
    end
end
return S
```

the clustering process and find a good partition of the dataset mostly based on the constraint set.

Table 5
Summary of datasets used for the experiments.

Name	No. instances	No. classes	No. features
Appendicitis	106	2	7
Breast Cancer	569	2	30
Bupa	345	2	6
Circles	300	2	2
Ecoli	336	8	7
Glass	214	6	9
Haberman	306	2	3
Hayesroth	160	3	4
Ionosphere	351	2	33
Iris	150	3	4
Led7Digit	500	10	7
Monk2	432	2	6
Moons	300	2	2
Movement Libras	360	15	90
Pima	768	2	8
Saheart	462	2	9
Sonar	208	2	60
Soybean	47	4	35
Spectfheart	267	2	44
Spiral	300	2	2
Tae	151	3	5
Vehicle	846	4	18
Vowel	990	11	13
Wdbc	569	2	30
Zoo	101	7	16

5.1. Constraint generation

Since we have the true labels associated with each dataset, we will use the method proposed in [18] to generate artificial constraint sets. This method consists of randomly selecting two instances of a dataset, then comparing its labels, and finally

Table 6
Number of constraints used in experiments.

Dataset	CS_{10}		CS_{15}		CS_{20}	
	ML	CL	ML	CL	ML	CL
Appendicitis	39	16	71	49	164	67
Breast Cancer	876	720	1965	1690	3487	2954
Bupa	323	272	699	627	1201	1145
Circles	208	227	502	488	853	917
Ecoli	163	398	357	918	609	1669
Glass	52	179	139	389	259	644
Haberman	304	161	634	401	1135	756
Hayesroth	39	81	102	174	177	319
Ionosphere	330	300	732	646	1299	1186
Iris	26	79	82	171	136	299
Led7Digit	126	1099	267	2508	460	4490
Monk2	473	473	979	1101	1917	1824
Moons	200	235	494	496	900	870
Movement Libras	27	603	112	1319	158	2398
Pima	1604	1322	3595	3075	6452	5329
Saheart	595	486	1292	1123	2330	1948
Sonar	100	110	245	251	436	425
Soybean	4	6	6	22	12	33
Spectfheart	233	118	543	277	965	466
Spiral	224	211	487	503	918	852
Tae	40	80	82	171	151	314
Vehicle	874	2696	1955	6046	3589	10 776
Vowel	445	4406	1026	10 000	1705	17 798
Wdbc	840	756	1925	1730	3472	2969
Zoo	21	34	29	91	41	169

setting an ML or CL constraint depending on whether the labels are the same or different.

We will generate, for each dataset, three different sets of constraints—CS_{10}, CS_{15} and CS_{20}—that will be associated with three small percentages of the size of the dataset: 10%, 15% and 20%. With n_f being the fraction of the size of the dataset associated with each of these percentages, the formula $\frac{n_f(n_f-1)}{2}$ tells us how many artificial constraints will be created for each constraint set; this number is equivalent to how many edges a complete graph with n_f vertices would have.

The random allocation of constraints has a potential advantage over simply using the constraints contained in an n_f-vertex complete graph: there is a lower probability of biasing the constraint set towards having classes with poor representation in it. Table 6 shows the number of constraints of each type obtained for each dataset.

Note that the greater the number of classes in the dataset, the fewer ML constraints obtained with the method proposed in [18]. This is because the probability of randomly choosing two individuals from the same class decreases as the number of classes present in the dataset increases.

5.2. Evaluation method

Since we have the true labels associated to each of the datasets, we can use them to evaluate the results provided by each method. We will use the Adjusted Rand Index (ARI) to measure the accuracy of the predictions resulting from each method we test [42]. The basic Rand Index computes the degree of agreement between two partitions C_1 and C_2 of a given dataset X. C_1 and C_2 are viewed as collections of $n(n-1)/2$ pairwise decisions [43].

For each pair of instances x_i and x_j in X, a partition assigns them to the same cluster or to different clusters. We take a as the number of pairings where x_i is in the same cluster as x_j in both C_1 and C_2, and b as the opposite event (x_i and x_j are in different clusters in C_1 and C_2). Then, the degree of similarity between C_1 and C_2 is calculated as in Eq. (15).

$$\text{Rand}(C_1, C_2) = \frac{a+b}{n(n-1)/2} \tag{15}$$

The ARI is a corrected-for-chance version of the Rand Index. This correction uses the expected similarity of all comparisons between clusterings specified by a random model to set up a baseline. The ARI is computed as in Eq. (16).

$$\text{ARI}(C_1, C_2) = \frac{\text{Rand}(C_1, C_2) - \text{Expected Index}}{\text{Maximum Index} - \text{Expected Index}}, \tag{16}$$

where Maximum Index is expected to be 1 and Expected Index is the already mentioned expected degree of similarity with a random model. It is easy to see that $\text{ARI}(C_1, C_2) \in [-1, 1]$, such that an ARI value close to 1 means a high degree of agreement between C_1 and C_2, a positive value close to 0 means no agreement and a value smaller that 0 means that the $\text{Rand}(C_1, C_2)$ is less than expected when comparing with random partitions. To summarize, the higher the ARI, the greater the degree of similarity between C_1 and C_2. For more details on ARI see [42]. Our objective is to quantify the quality of the solutions obtained as a result of the methods presented in this paper. To accomplish this task we just set one of the two partitions given to compute ARI as the ground truth labels.

In order to get a better insight of the proposed method and to be able to achieve a more reliable comparison with other methods we also include the *Unsat* measure as a benchmark. It refers to the percentage of unsatisfied constraints in the partition produced by a method as result. This measure lets us examine the capability of a given method to integrate constraints into the clustering process.

5.3. Validation of results

In order to validate the results which will be presented in Section 6 we will use Bayesian statistical tests, instead of the classic Null Hypothesis Statistical Tests (NHST). In [44] we find an in-depth analysis of the disadvantages of NHST, and a new model is proposed for carrying out comparisons researchers are interested in. *"In a nutshell: NHST do not answer the question we ask"*. To put it clear, the disadvantages of the NHST that the authors highlight in [44] are based on the trap of black-and-white thinking, this is: to reject, or not to reject?

To start with, NHST do not provide us with the probabilities associated to the analyzed hypotheses, and therefore it is not possible to answer the question: what is the probability that two methods are different? Another pitfall of NHST is that, with a sufficiently large number of observations, it is possible to reject almost any hypothesis. This is because the p-value does not allow us to separate between the effective size and the sample size, which is established by the researcher.

Also, NHST do not provide information about the magnitude of the effects and the uncertainty of its estimate. As a consequence, NHST may reject hypotheses despite very small effects, or even if there is significant uncertainty in the magnitude of the effects.

Furthermore, and this is a situation that all researchers have faced, NHST do not provide any information about the null hypothesis! That is: What can we conclude when NHST do not reject the null hypothesis? We cannot infer anything since NHST cannot provide evidence in favor of the said hypothesis.

Finally, there are two other problems that researchers face when performing NHST. The first one is the choice of the significance level α, for which there are no objective guidelines despite being critical to the test results. The second one is the need to previously formalize the intentions of the sampling of the results, which are usually fixed a posteriori; this could lead to a misreading of said results.

As shown in [44], most of these problems can be avoided by using Bayesian tests instead of NHST. In particular we will use the Bayesian sign test, which is the Bayesian version of the

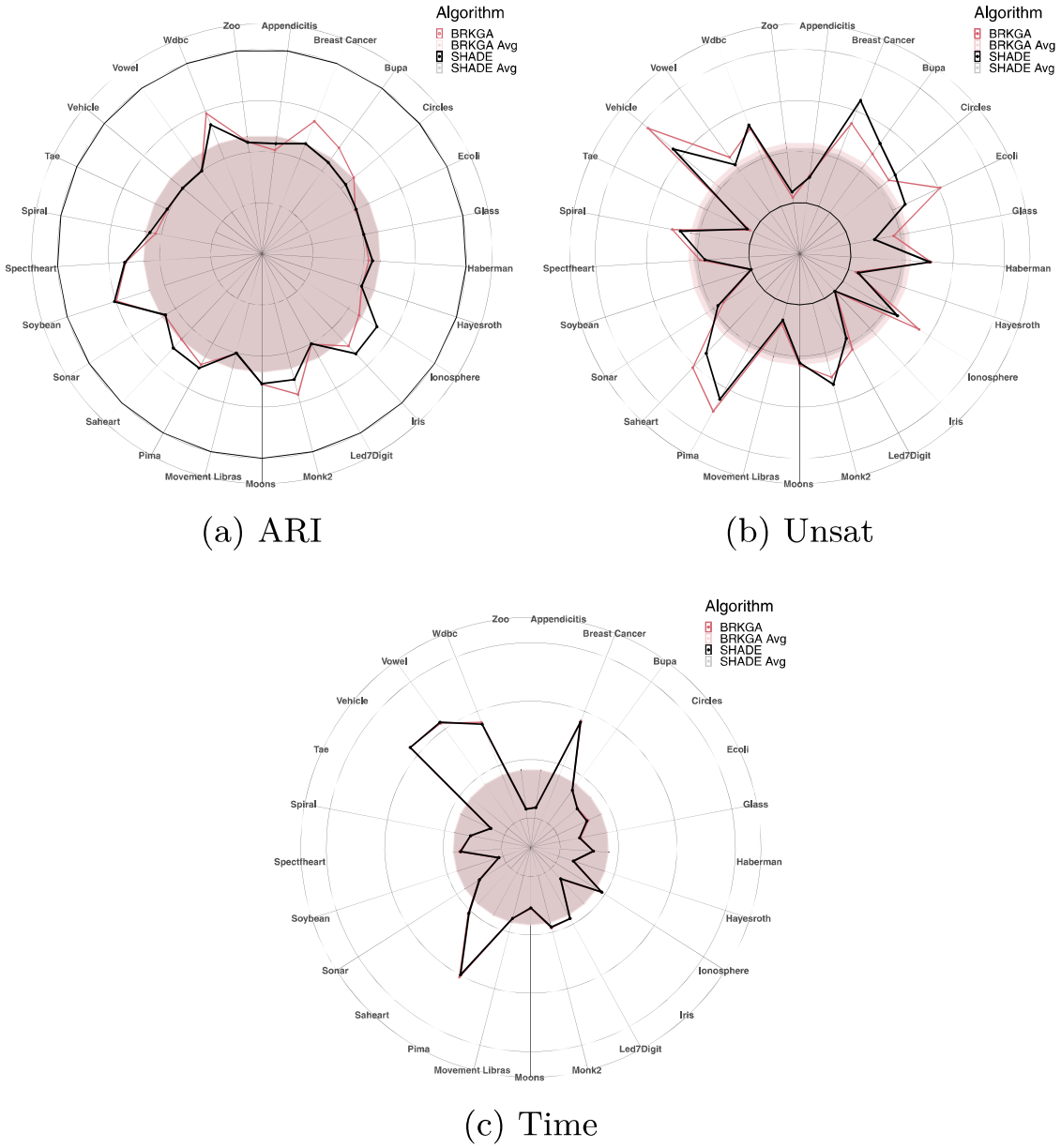


Fig. 3. Radarplots for the comparison of $SHADE_{CC}$ with $BRKGA_{CC}$ at CS_{10}.

frequentist non-parametric sign test. To make use of it we will employ the R package `rNPBST`, whose documentation and guide can be found in [45].

We will use the Bayesian sign test to validate our results, which is based on obtaining the statistical distribution of a certain parameter ρ according to the difference between the results, under the assumption that said distribution is a Dirichlet distribution. To get the distribution of ρ we count the number of times that $A-B < 0$, the number of times where there are no significant differences, and the number of times that $A - B > 0$. In order to identify cases where there are no significant differences, we define the region of practical equivalence (rope) $[r_{\min}, r_{\max}]$, so that $P(A \approx B) = P(\rho \in \text{rope})$. Using these results we calculate the weights of the Dirichlet distribution and sample it to get a set of triplets with the following form:

$$[P(\rho < r_{\min}) = P(A - B < 0), \quad P(\rho \in \text{rope}), \quad P(\rho > r_{\max}) = P(A - B > 0)]$$

Table 7
Parameters setup used for $BRKGA_{CC}$ and $SHADE_{CC}$.

Parameter	Meaning	$BRKGA_{CC}$	$SHADE_{CC}$
$\|P\|$	Population size	100	100
Evals	Fitness function evaluations	300 000	300 000
P_e	Size of the elite set in population	$0.2 * \|P\|$	$0.25 * \|P\|$
P_m	Number of mutants to be introduced in the population in each generation	$0.2 * \|P\|$	–
p_{inherit}	Probability that a feature is inherited from an elite parent	50%	–
k	Output partition number of clusters	No. Classes (Table 5)	

5.4. Calibration

Table 7 shows a summary of the parameter setup used for both $BRKGA_{CC}$ and $SHADE_{CC}$ algorithms.

In both cases – $BRKGA_{CC}$ and $SHADE_{CC}$ – the population size will be 100 individuals, and the stop criterion is given by the

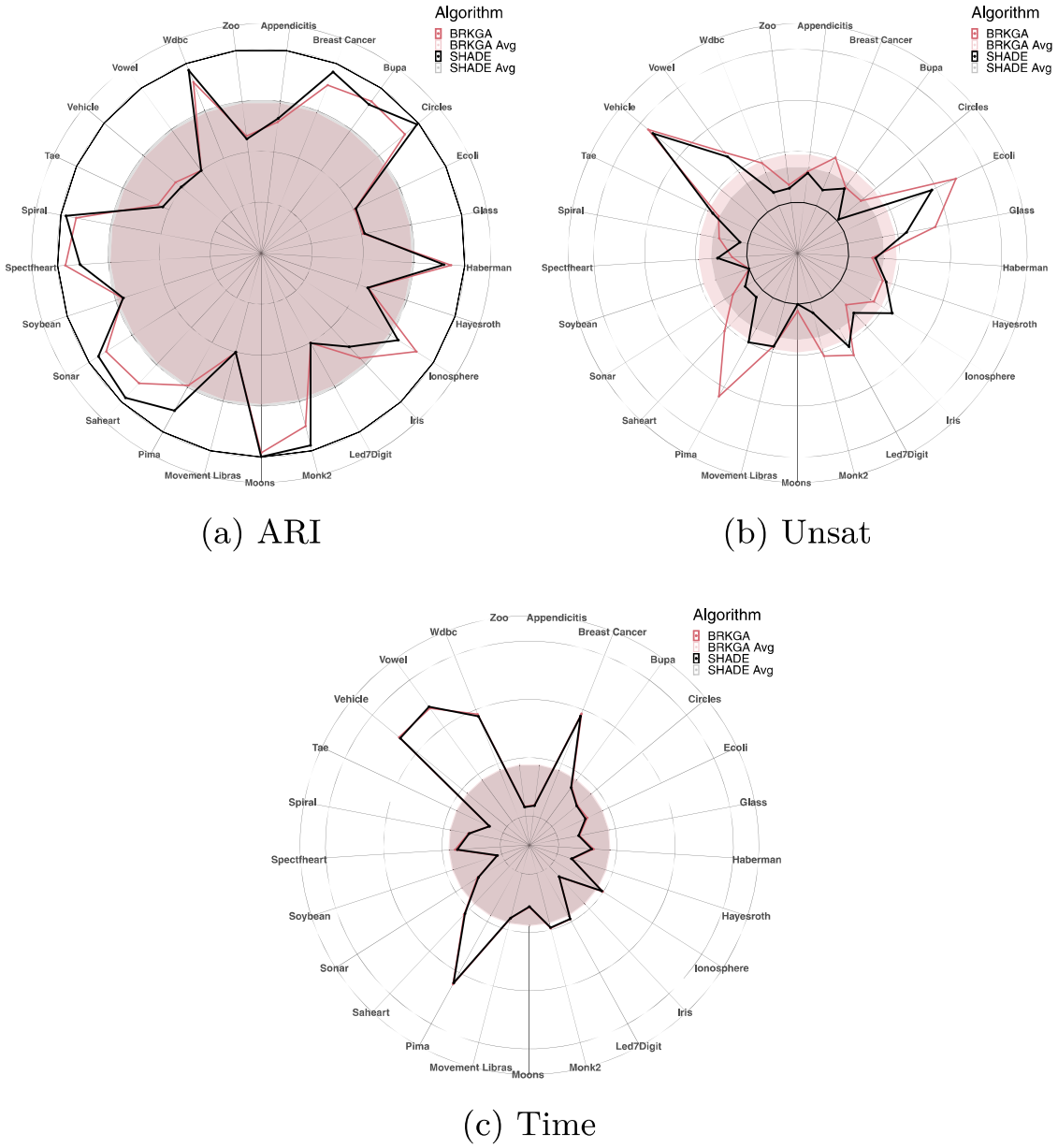


Fig. 4. Radarplots for the comparison of SHADE_{CC} with BRKGA_{CC} at CS_{15}.

number of evaluations of the fitness function, which at most will be 300 000. A Python implementation for both algorithms can be found at GitHub[4]

For our experimentation SHADE_{CC} parameters have been set up trying to preserve its general applicability. However, when it comes to particular applications, we recommend that an in-depth analysis of all parameters using the appropriate optimization process be carried out. Also notice that there are general rules to help bias the parameter optimization process towards the correct direction. In order not to compromise the exploration capability of the method we consider that the probability of a feature being inherited from an elite parent must lie within the range $p_{\text{inherit}} \in [0.4, 0.7]$. Something similar can be said for other exploration-restricting parameters such as $P_e \in [0.1 \times P, 0.4 \times P]$ and $P_m \in [0.0.5 \times P, 0.3 \times P]$. Regarding the size of the population and the number of fitness function evaluations, we find that these are very problem-dependent parameters and the general advice is to make sure that a reasonably-sized population can converge.

To compare with the state-of-the-art methods mentioned in Section 2.3 we will use the parameters setup shown in Table 8 for the implementation that can be found at GitHub.[5]

Parameter values have been assigned following the guidelines of the original creators of the different proposals. Since the evaluation in the experimental stage makes use of a high number of datasets, tuning each parameter specifically for each dataset is not feasible. Indeed, our goal is not optimization in a case-by-case basis but instead a comparison made in the most general scenario possible. Therefore, given that the purpose of this work is to draw a fair comparison between the algorithms and assess their robustness in a common environment with multiple datasets, we have not included a tuning step to maximize any particular performance metric.

4 https://github.com/GermangUgr/SHADE_CC.

5 https://github.com/GermangUgr/TFG/tree/master/Software.

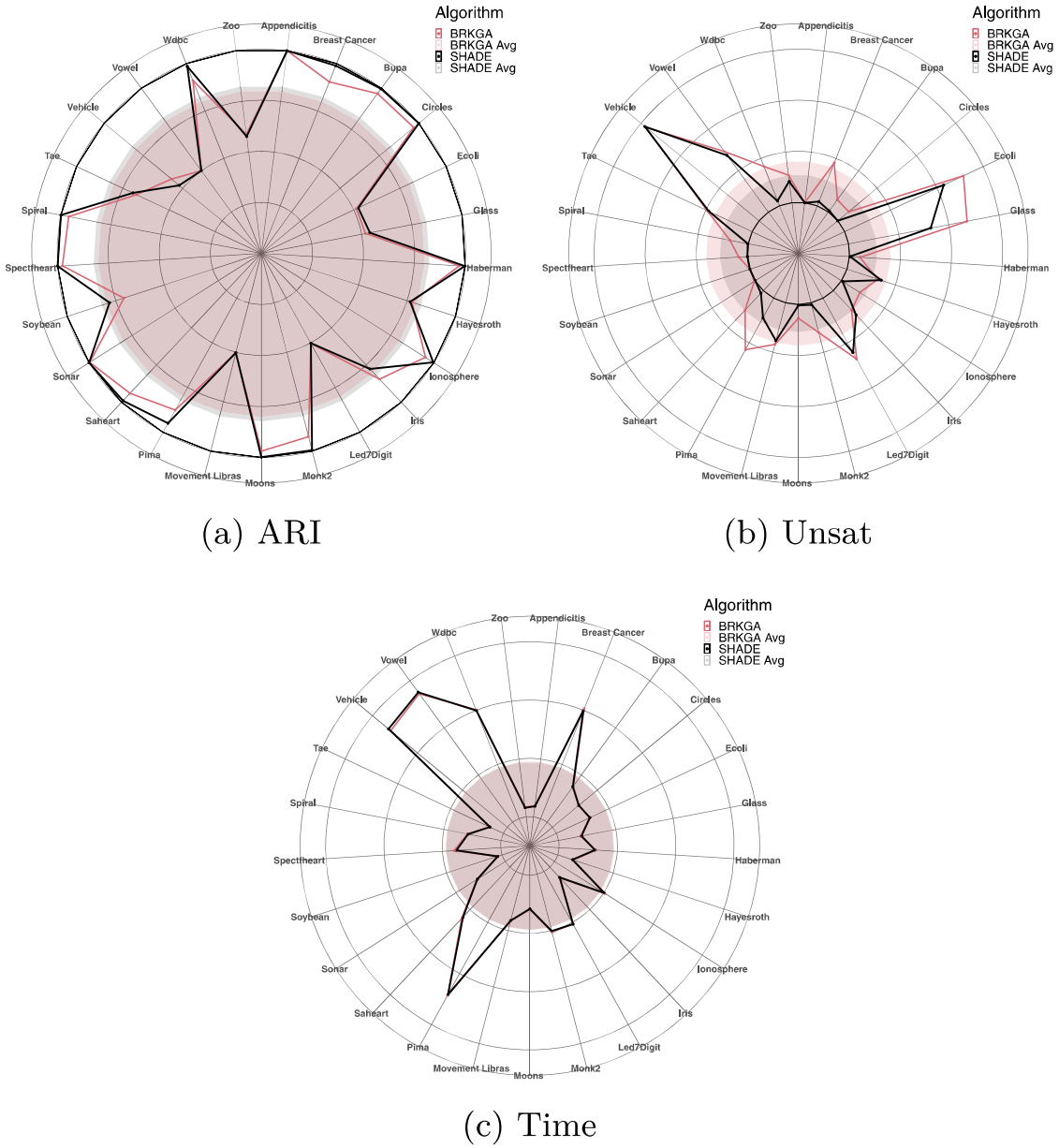


Fig. 5. Radarplots for the comparison of SHADE_{CC} with BRKGA_{CC} at CS_{20}.

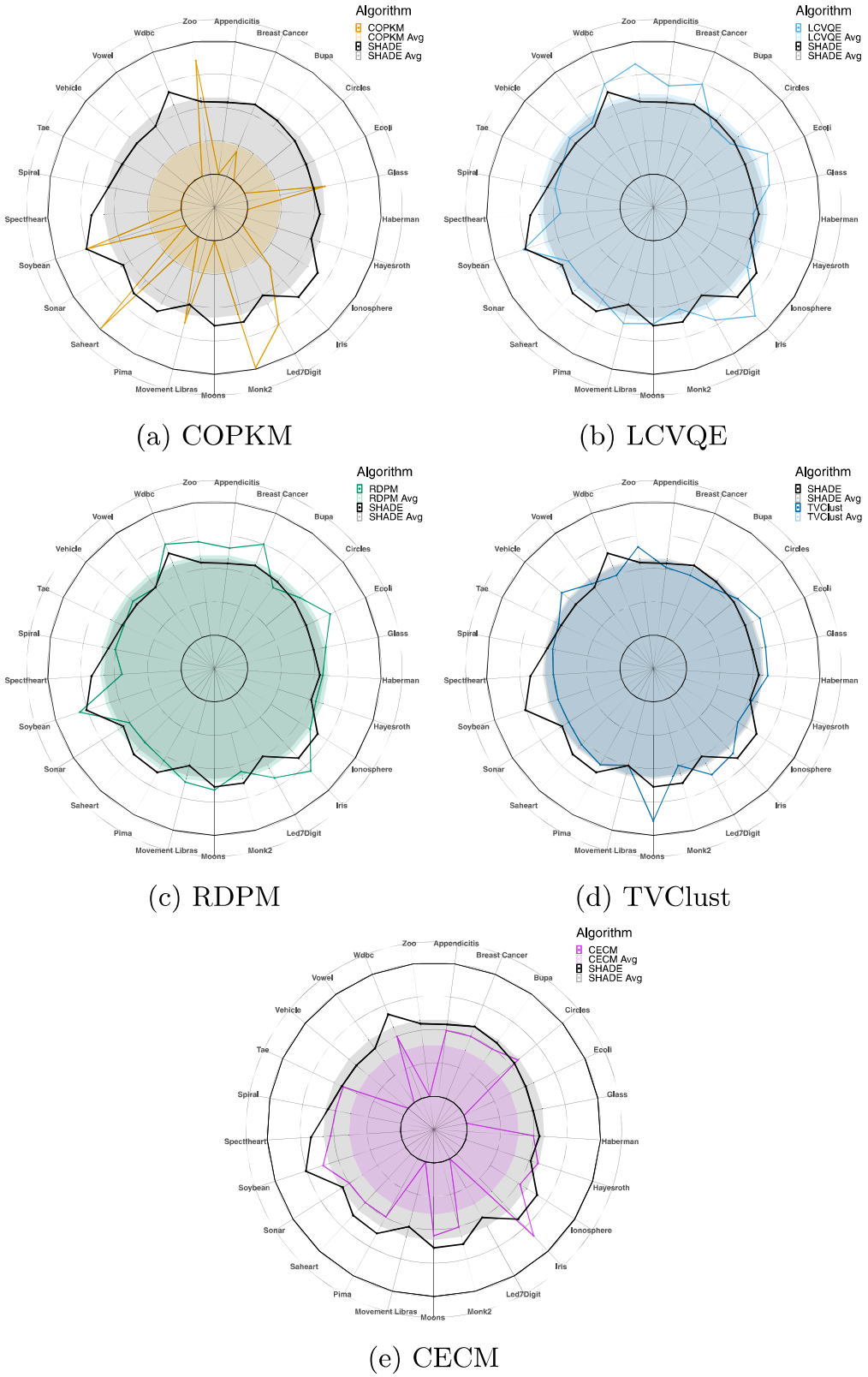
Table 8
Parameters setup used for the state-of-the-art algorithms.

Common parameters	Meaning	Value
`max_iter`	Maximum number of iterations	300
k	Output partition number of clusters	No. Classes (Table 5)
Specific Parameters	**Name and Value**	
COPKM	`tolerance` $= 1 * 10^{-4}$; `init_mode` $=$ ‘‘`rand`’’	
CECM	$\alpha = 1$, $\rho = 100$, $\xi = 0.5$, `stop_threshold` $= 1 * 10^{-3}$, `init_mode` $=$ ‘‘`rand`’’	
LCVQE	`initial_centroids` $= \emptyset$	
RDPM	$\xi_0 = 0.1$, $\xi_{\text{rate}} = 1$, λ is calculated on the basis of the mean distances in the dataset	
TVClust	$\alpha_0 = 1.2$, `stop_threshold` $= 5 * 10^{-4}$	

6. Experimental results

In this section we present Figures from 3 to 8, which display the results obtained by the methods to be compared for each dataset and constraint set.

Since the methods we are comparing involve non-deterministic procedures, the results may vary from one run to another. To lessen the effect this may have on the results, we will apply each method 5 times to every dataset and constraint set, so that the measures shown in the previously mentioned tables correspond to the average of the 5 runs. Given the large amount of datasets used in our experiments, we have choosen *radarplots* as the better method for visualizing the results. Radarplots are histograms presented in polar coordinates. In each radarplot two sets or results are presented, as well as their average. However, if the reader is interested in the particular numerical results obtained in our experimentation, they can be found in Appendix.



(a) COPKM (b) LCVQE

(c) RDPM (d) TVClust

(e) CECM

Fig. 6. Radarplots for the comparison of $SHADE_{CC}$ with the state-of the-art at CS_{10}.

6.1. Comparing $BRKGA_{CC}$ with $SHADE_{CC}$

In Figures from 3 to 5 the ARI subplot shows the Adjusted Rand Index for each case, the Unsat subplot displays the percentage of violated constraints and the Time subplot shows the time taken for each algorithm to deliver results, measured in seconds. The upper limit of the ARI measure and the lower limit of the Unsat measure have been highlighted with a black line (circle in polar coordinates) for a better visualization.

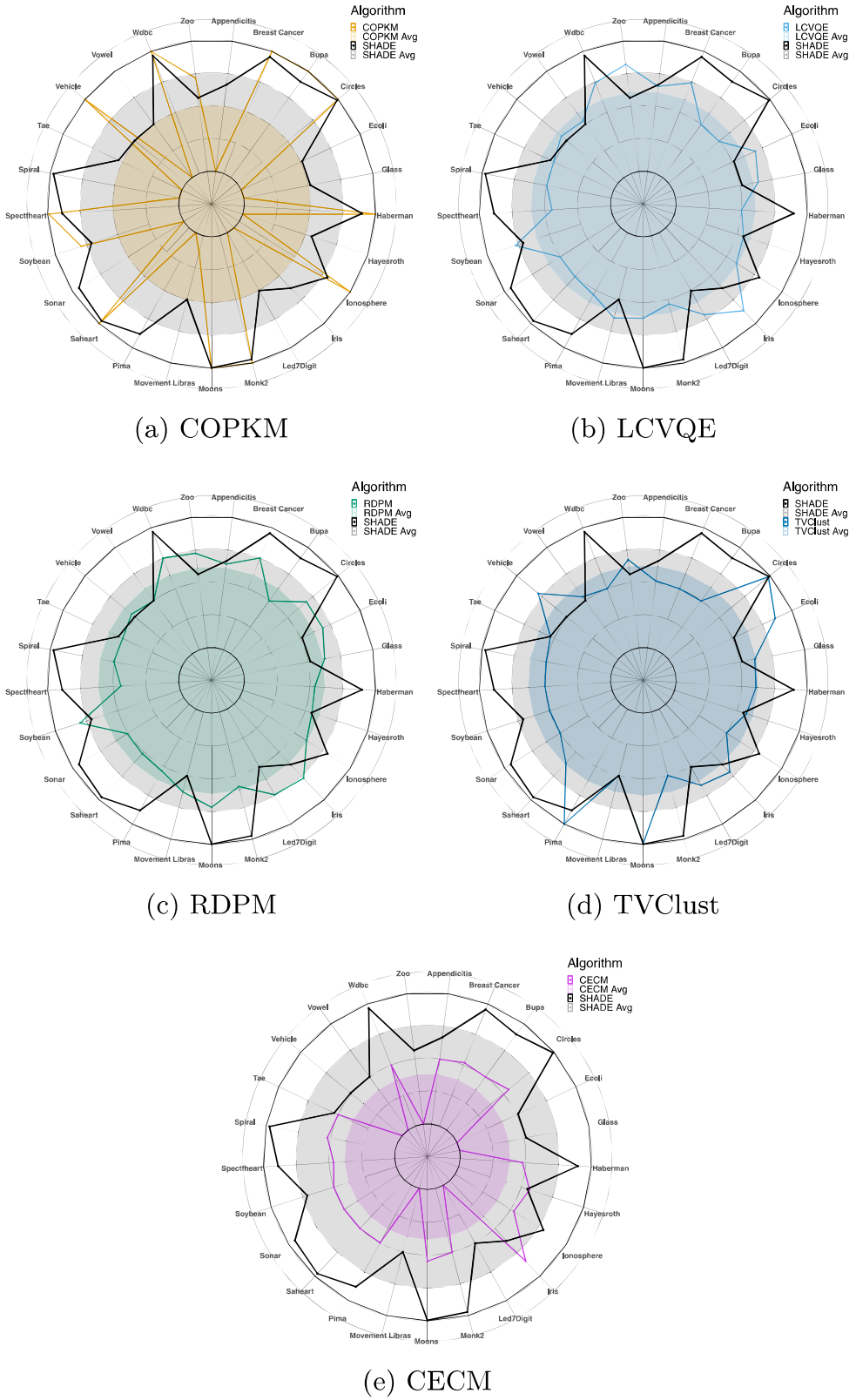
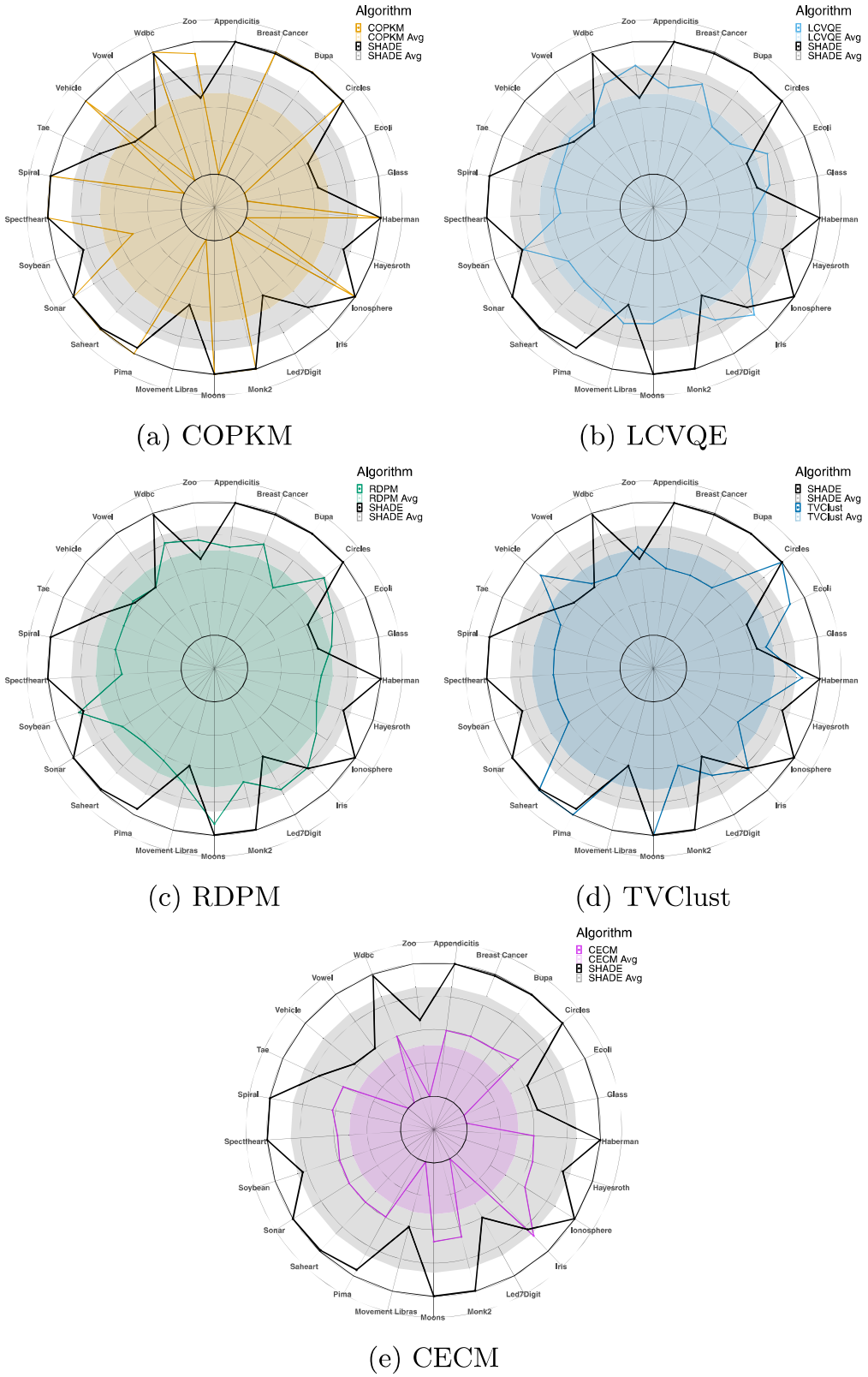


Fig. 7. Radarplots for the comparison of SHADE$_{CC}$ with the state-of.the-art at CS_{15}.

Fig. 3 shows the results for the SC_{10} constraint set. We can note that SHADE$_{CC}$ represents a consistent improvement over BRKGA$_{CC}$ for the Unsat measure, while in terms of ARI both methods are similar. This is because the penalty applied to the fitness function of both methods has little presence when operating with a small constraint set, and since they are population-based and heuristic-guided methods, they have similar exploration and exploitation capabilities. Regarding the Time measure, both method appear to be similar in the diagram, which one could expect given that they are both population-based method. However, in Table A.11



(a) COPKM (b) LCVQE

(c) RDPM (d) TVClust

(e) CECM

Fig. 8. Radarplots for the comparison of SHADE$_{CC}$ with the state-of.the-art at CS_{20}.

we can notice a consistent difference in running time in favor of SHADE$_{CC}$, even if this difference is not big enough with respect to the scale of the results to be visualized.

Fig. 4 presents the results obtained with the CS_{15} constraint set. While conclusions for Unsat and Time remain unchanged, we see a clear difference as far as ARI is concerned. We observe that both methods already report ARI results above 0.7, which in an ARI ranking is considered to be an acceptable result. We can even observe that the SHADE$_{CC}$ method provides an ARI higher than

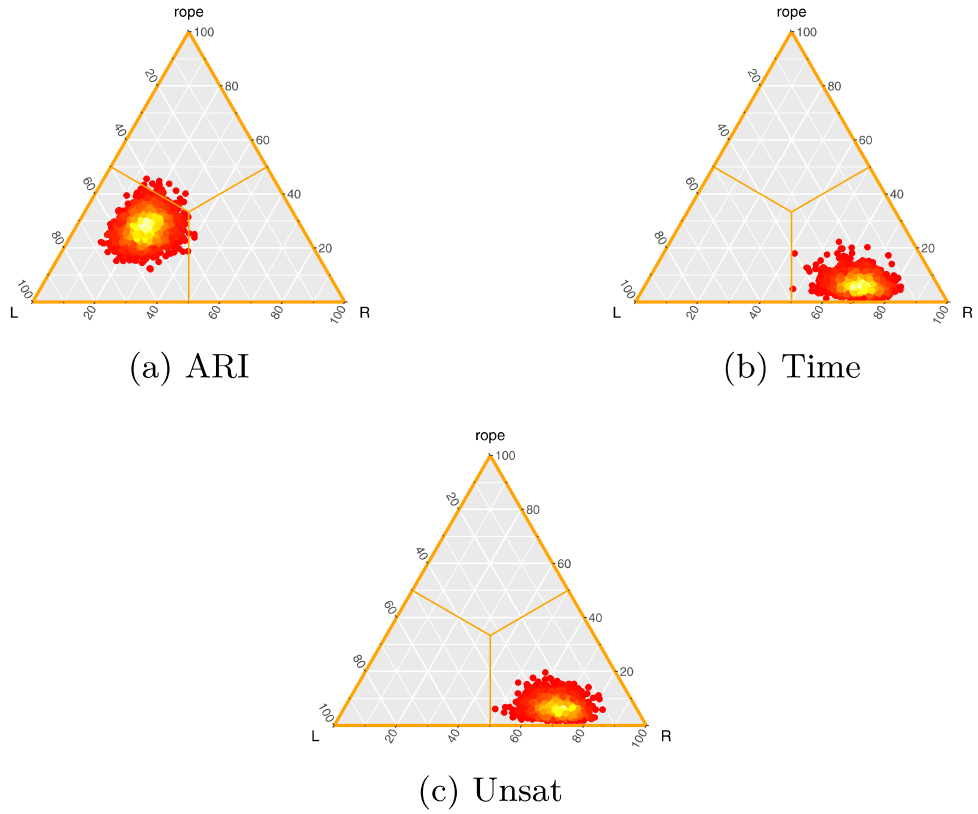


Fig. 9. Heatmaps for the comparison with $BRKGA_{CC}$.

Table 9
Average experimental results obtained by $SHADE_{CC}$ and $BRKGA_{CC}$ for all constraint sets.

Dataset	$BRKGA_{CC}$			$SHADE_{CC}$		
	ARI	Unsat (%)	Time (s)	ARI	Unsat (%)	Time (s)
CS_{10}	**0.161**	11.771	1675.618	0.156	10.828	1652.02
CS_{15}	0.483	9.456	1784.931	**0.507**	6.997	1751.368
CS_{20}	0.602	8.086	1895.671	**0.647**	5.443	1880.921

0.9 for 7 datasets, compared to $BRKGA_{CC}$ that only obtains similar results in 2 datasets.

It is in Fig. 5, which gathers results obtained for the constraint set CS_{20}, that we observe major differences between the two methods. We find that $SHADE_{CC}$ is capable of producing clusters identical to the original ones for 4 datasets, with the exception of one dataset, for which $BRKGA_{CC}$ also produces identical clusters. Let us remember that an ARI of 1 represents a perfect match between the two partitions given to compute it. This improvement in ARI is clearly reflected in the percentage of violated constraints – Unsat column –, value for which, once again, $SHADE_{CC}$ outperforms $BRKGA_{CC}$. Conclusions on execution time remain the same. Table 9 shows a summary of the results presented above, displaying average results for all experiments.

6.2. Comparing $SHADE_{CC}$ with the state-of-the-art

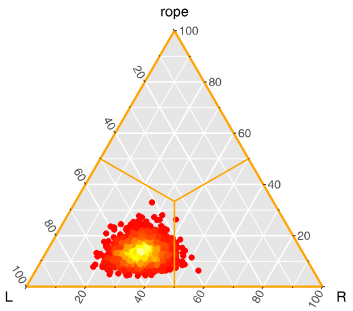
In this section we present a brief comparison between the proposed $SHADE_{CC}$ method and some of the state-of-the-art methods. Figures from 6 to 8 show the results obtained by these methods. To shorten the presented results this time we will only compare the ARI, whose limits have been highlighted with a black line (circle in polar coordinates) for a better visualization.

It should be noted that there are some missing results in these tables. In the case of the COPKM algorithm, this is due to the fact that it is highly dependent on the order in which constraints are analyzed. It is possible that COPKM cannot find a solution, even though it is always feasible, since the constraints have been generated based on the true labels. In the case of CECM, some of the results are not available because the memory structures that hold the algorithm grow non-linearly with the number of classes and the number of features of the dataset to be analyzed. We establish that in these cases the ARI value considered for the mean calculation – and the forthcoming statistical analysis of results – is −1.000, which is the worst possible ARI value.

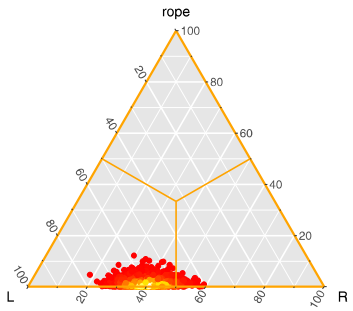
Fig. 6 presents the results obtained for the CS_{10} constraint set. LCVQE, RDPM and TVClust methods (Figs. 6b, 6c and 6d respectively) stand out, achieving results as competitive as $SHADE_{CC}$ with a low number of constraints.

In one hand, Fig. 7, which shows the results for the CS_{15} constraint set, we observe that there is a significant improvement in precision in the results obtained by $SHADE_{CC}$. On the other hand, methods that obtained better results for smaller constraint sets do not present an improvement comparable to that presented by $SHADE_{CC}$, with the exception of COPKM (Fig. 7a), which is able to reach the optimum for some of the cases in which it finds a solution.

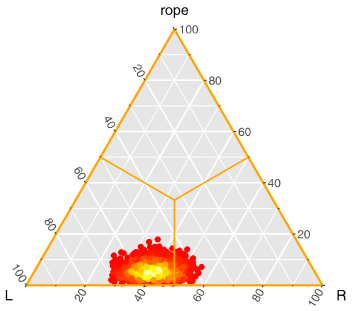
Fig. 8 presents the results obtained for CS_{20} and we continue to observe the progression in accuracy of $SHADE_{CC}$. As far as the state-of-the-art methods are concerned, we find a general improvement in ARI, except for CECM. Even with this, it is $SHADE_{CC}$ and COPKM (Fig. 8a) that obtain the highest quality results, with the difference that $SHADE_{CC}$ is able to produce results in all cases. The TVClust (Fig. 7d) method should also be highlighted, since it is able to find the optimal solution in 4 cases, which implies a significant improvement over its performance on previous constraint sets. Table 10 shows a summary of the results presented above, displaying average results for all experiments.



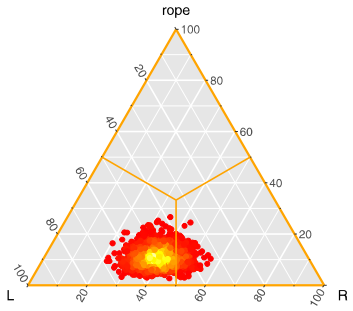
(a) SHADE$_{CC}$ vs COPKM



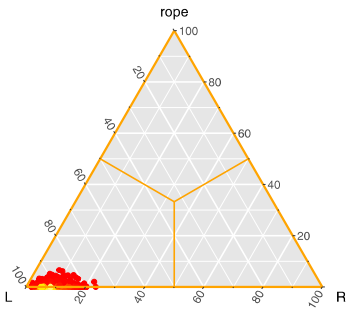
(b) SHADE$_{CC}$ vs LCVQE



(c) SHADE$_{CC}$ vs RDPM



(d) SHADE$_{CC}$ vs TVClust



(e) SHADE$_{CC}$ vs CECM

Fig. 10. Heat diagrams for the comparison with the state-of-the-art.

Table 10
Average experimental results obtained by SHADE$_{CC}$ and the five state-of-the-art methods for all constraint sets.

Dataset	SHADE$_{CC}$	COPKM	LCVQE	RDPM	TVClust	CECM
CS_{10}	0.156	−0.51	**0.215**	0.211	0.13	−0.226
CS_{15}	**0.507**	0.01	0.213	0.229	0.253	−0.238
CS_{20}	**0.647**	0.224	0.212	0.28	0.319	−0.228

6.3. Statistical analysis of results

We can perform an empirical analysis of all methods with the results obtained for a total of 75 different datasets—the 25 datasets in combination with the 3 constraint sets for each one of them. This way we can statistically determine whether SHADE$_{CC}$ represents a significant improvement over previous proposals.

Having in mind the notation introduced in Section 5.3, and as far as the comparison of population-based methods is concerned, we will refer to the results obtained by BRKGA$_{CC}$ as sample *A*, and to the results obtained with SHADE$_{CC}$ as sample *B*.

One of the major advantages of the Bayesian sign test is that we can obtain a very illustrative visual representation of its results. We can produce a representation of the triplet set in the form of a heatmap where each triplet constitutes one point whose location is given by barycentric coordinates. With this in mind, we will associate each of the triplet values with each of the three vertices of an equilateral triangle. In order to find out where a certain triplet will be placed within the triangle, we will take each of its three values and draw a parallel line to the opposing side of the corresponding vertex; the separation between a triangle side and its parallel line will be proportional to the associated triplet value, so that the higher the value, the closer the line will be to the vertex. The location where the three lines intersect is where we draw a point. Since the values of every triplet describe a probability distribution and therefore they must add up to one, we can be sure that all triplets will lie in some point within the triangle. The color indicates the density of points in a given region, with yellow representing a high density and red a low density.

In one hand, Fig. 9a shows the heatmap associated with the ARI measurement. The diagram suggests that the Bayesian sign

Table A.11
Experimental results obtained for CS_{10} comparing $SHADE_{CC}$ and $BRKGA_{CC}$.

Dataset	$BRKGA_{CC}$			$SHADE_{CC}$		
	ARI	Unsat (%)	Time (s)	ARI	Unsat (%)	Time (s)
Appendicitis	0.022	5.454	383.642	**0.086**	5.091	366.491
Breast Cancer	**0.395**	17.444	3651.263	0.159	22.256	3602.665
Bupa	**0.281**	14.151	1420.117	0.104	16.672	1409.487
Circles	**0.166**	12.598	1077.238	0.063	14.207	1060.787
Ecoli	0.009	20.321	1169.798	**0.020**	12.763	1112.981
Glass	**0.016**	8.658	735.507	0.013	4.848	694.962
Haberman	0.047	15.699	1155.985	**0.084**	15.441	1146.023
Hayesroth	**0.027**	1.500	550.117	0.025	2.000	529.424
Ionosphere	0.125	17.587	1894.390	**0.332**	12.603	1872.308
Iris	0.238	0.190	505.112	**0.342**	0.000	483.584
Led7Digit	**0.009**	11.347	1786.943	0.004	8.866	1784.708
Monk2	**0.421**	14.926	1848.543	0.270	16.385	1821.208
Moons	**0.277**	11.816	1098.089	0.272	11.356	1075.089
Movement Libras	**0.004**	4.159	1508.279	0.003	3.396	1519.185
Pima	0.236	25.195	4072.087	**0.275**	22.468	3990.383
Saheart	0.147	20.555	2135.624	**0.265**	16.725	2095.554
Sonar	**0.124**	7.619	1106.039	0.116	8.952	1081.994
Soybean	0.496	0.000	176.333	**0.515**	0.000	151.163
Spectfheart	0.334	9.573	1451.235	**0.343**	8.604	1405.243
Spiral	0.058	15.402	1098.622	**0.114**	13.793	1099.747
Tae	0.018	0.833	520.855	**0.024**	1.333	508.988
Vehicle	0.005	28.515	4366.104	**0.006**	22.151	4352.663
Vowel	**0.003**	13.330	4245.672	0.002	11.523	4290.154
Wdbc	**0.478**	16.316	3596.517	0.357	17.093	3528.029
Zoo	**0.105**	1.091	336.345	0.098	2.182	317.674
Mean	**0.161**	11.771	1675.618	0.156	10.828	1652.02

Table A.12
Experimental results obtained for CS_{15} comparing $SHADE_{CC}$ and $BRKGA_{CC}$.

Dataset	$BRKGA_{CC}$			$SHADE_{CC}$		
	ARI	Unsat (%)	Time (s)	ARI	Unsat (%)	Time (s)
Appendicitis	0.296	6.167	381.625	**0.332**	5.833	362.106
Breast Cancer	0.772	10.172	3854.911	**0.911**	3.310	3766.402
Bupa	**0.840**	5.973	1463.152	0.801	5.641	1437.639
Circles	0.828	6.101	1136.484	**0.982**	0.404	1108.928
Ecoli	0.015	24.314	1200.463	**0.025**	19.090	1127.568
Glass	0.017	17.500	745.987	**0.031**	11.743	722.009
Haberman	**0.866**	4.677	1209.471	0.796	5.314	1147.479
Hayesroth	**0.107**	7.536	545.861	0.099	8.261	520.702
Ionosphere	**0.803**	7.722	1986.543	0.591	11.974	1944.085
Iris	**0.412**	3.873	506.548	0.260	6.087	486.723
Led7Digit	0.004	12.836	1902.272	**0.007**	10.919	1897.511
Monk2	0.751	10.827	1949.305	**0.945**	2.067	1920.361
Moons	0.958	1.334	1133.858	**0.995**	0.020	1113.687
Movement Libras	**0.006**	8.679	1584.028	0.001	8.889	1590.321
Pima	0.481	22.051	4451.193	**0.762**	9.928	4402.693
Saheart	0.749	10.940	2281.992	**0.945**	1.839	2225.210
Sonar	0.800	5.040	1124.700	**0.890**	2.177	1069.862
Soybean	**0.428**	0.000	170.674	0.422	0.000	155.948
Spectfheart	**0.924**	2.854	1561.040	0.778	5.732	1485.260
Spiral	0.847	5.677	1133.962	**0.949**	1.495	1102.920
Tae	**0.119**	7.115	522.088	0.066	8.380	513.289
Vehicle	**0.089**	28.044	4806.559	0.018	26.856	4749.812
Vowel	**0.002**	14.486	4807.165	0.001	13.341	4867.849
Wdbc	0.805	8.985	3829.407	**0.930**	2.801	3750.711
Zoo	**0.157**	3.500	333.984	0.128	2.834	315.131
Mean	0.483	9.456	1784.931	**0.507**	6.997	1751.368

Table A.13
Experimental results obtained for CS_{20} comparing $SHADE_{CC}$ and $BRKGA_{CC}$.

Dataset	$BRKGA_{CC}$			$SHADE_{CC}$		
	ARI	Unsat (%)	Time (s)	ARI	Unsat (%)	Time (s)
Appendicitis	1.000	0.000	389.905	1.000	0.000	371.579
Breast Cancer	0.807	9.086	4061.162	**0.977**	0.891	3984.370
Bupa	0.936	2.975	1529.151	**0.993**	0.281	1508.772
Circles	0.935	2.814	1180.268	**1.000**	0.000	1170.452
Ecoli	0.028	25.716	1267.419	**0.047**	21.466	1277.903
Glass	0.038	23.610	766.861	**0.082**	16.412	796.162
Haberman	0.943	2.073	1266.131	**0.997**	0.138	1235.441
Hayesroth	**0.548**	6.331	554.339	0.531	7.056	540.585
Ionosphere	0.902	4.314	2051.251	**0.993**	0.258	2019.558
Iris	**0.691**	5.241	513.675	0.553	6.528	488.262
Led7Digit	**0.007**	13.653	2053.264	0.005	12.170	2066.223
Monk2	0.855	6.025	2055.425	**0.991**	0.379	2013.468
Moons	0.937	2.689	1177.848	**0.995**	0.158	1157.222
Movement Libras	**0.006**	8.381	1654.159	0.004	7.676	1645.174
Pima	0.753	11.544	4855.446	**0.899**	4.419	4820.684
Saheart	0.881	5.250	2402.513	**0.981**	0.706	2353.931
Sonar	0.992	0.186	1138.769	**1.000**	0.000	1127.020
Soybean	0.416	0.000	171.281	**0.563**	0.000	153.406
Spectfheart	0.953	1.677	1602.229	**1.000**	0.000	1526.925
Spiral	0.921	3.604	1178.580	**0.998**	0.091	1155.194
Tae	0.351	9.807	527.292	**0.392**	9.419	508.610
Vehicle	**0.144**	28.781	5189.066	0.044	29.008	5284.288
Vowel	0.002	14.682	5450.331	**0.003**	13.766	5514.052
Wdbc	0.823	8.281	4019.683	**0.972**	1.068	3983.564
Zoo	**0.176**	5.429	335.715	0.153	4.191	320.186
Mean	0.602	8.086	1895.671	**0.647**	5.443	1880.921

test assigns a high probability to $A - B < 0$, since most triplets are represented in the lower left third of the triangle. It also indicates that in some cases there are no significant differences between A and B, since we also find triplets in the rope area. Bearing this in mind, and considering that ARI is a measure to maximize, we can say that, for cases where there are significant differences, $SHADE_{CC}$ performs better than $BRKGA_{CC}$. On the other hand, Figs. 9b and 9c which show the heatmaps obtained for Time and Unsat measurements, are very similar. In both, all triplets are represented in the lower right third of the triangle, indicating that the Bayesian sign test assigns a high probability to $A-B > 0$. With this, and bearing in mind that both measures must be minimized, we conclude that $SHADE_{CC}$ represents a solid improvement over $BRKGA_{CC}$ as far as this measures are concerned.

Regarding the comparison of $SHADE_{CC}$ with the state-of-the-art, we will keep the results obtained by $SHADE_{CC}$ as B, and note the results obtained by the state-of-the-art methods as A.

This way, we apply the process described earlier in this section to obtain the heatmaps corresponding to the comparison of $SHADE_{CC}$ with each of the 5 state-of-the-art methods. This time the only measure we compare is ARI. Figs. 10a to 10e show said comparison.

We see very similar diagrams for COPKM (10a), LVCQE (10b), RDPM (10c) and TVClust (10d). In all of them most of the triplets are represented in the lower middle region of the diagram, with a slight shift of the point cloud to the left. That is indicative that there are indeed significant differences between the two methods – since there are no triplets with representation in the rope area –, but their performance differs depending on the case, with $SHADE_{CC}$ outperforming the other methods in most cases. However, this does not apply in the case of CECM (10e), where $SHADE_{CC}$ indisputably gets the best results.

7. Conclusions and future work

In this paper we proposed $SHADE_{CC}$, the first application of a DE-based algorithm to the SSL constrained clustering problem. Focusing on the instance-level ML and CL constraints, $SHADE_{CC}$ has proven that nature-inspired evolution-based techniques constitute a good approach to constrained clustering, being able to scale the quality of the results in a way directly proportional to the number of constraints.

Supported by the Bayesian statistical tests, we were able to objectively prove that the $SHADE_{CC}$ approach is significantly better than previous evolution-based proposals. As far as the state-of-the-art is concerned, the $SHADE_{CC}$ approach has proven to be

Table A.14
Experimental results obtained for CS_{10} comparing $SHADE_{CC}$ and the state-of-the-art.

Dataset	$SHADE_{CC}$	COPKM	LCVQE	RDPM	TVClust	CECM
Appendicitis	0.086	–	**0.335**	0.316	0.025	−0.005
Breast Cancer	0.159	−0.604	0.486	**0.502**	0.000	0.000
Bupa	**0.104**	–	−0.005	−0.005	−0.004	−0.011
Circles	0.063	–	−0.003	**0.162**	0.137	0.133
Ecoli	0.020	–	0.387	**0.417**	0.265	–
Glass	0.013	0.184	**0.268**	0.197	0.211	–
Haberman	0.084	–	−0.002	0.127	**0.218**	−0.004
Hayesroth	0.025	–	0.106	0.097	0.054	**0.139**
Ionosphere	**0.332**	–	0.168	0.197	0.000	0.030
Iris	0.342	−0.285	**0.730**	0.607	0.244	0.684
Led7Digit	0.004	**0.497**	0.425	0.369	0.316	–
Monk2	0.270	**0.982**	0.072	0.094	−0.002	0.007
Moons	0.272	–	0.241	0.319	**0.785**	0.092
Movement Libras	0.003	0.285	**0.293**	0.256	0.000	–
Pima	**0.275**	–	0.076	0.075	0.145	−0.008
Saheart	0.265	**0.974**	0.018	0.020	0.068	0.000
Sonar	**0.116**	–	0.004	0.013	0.000	0.000
Soybean	0.515	0.503	0.545	**0.621**	0.000	0.244
Spectfheart	**0.343**	–	−0.107	−0.114	0.000	0.050
Spiral	**0.114**	–	−0.003	0.012	0.034	−0.002
Tae	0.024	–	0.009	−0.000	**0.067**	0.000
Vehicle	0.006	–	0.121	0.081	**0.280**	–
Vowel	0.002	–	0.063	−0.003	**0.067**	–
Wdbc	0.357	–	0.486	**0.502**	0.000	0.000
Zoo	0.098	**0.715**	0.666	0.412	0.335	–
Mean	0.156	−0.51	**0.215**	0.211	0.13	−0.226

Table A.15
Experimental results obtained for CS_{15} comparing $SHADE_{CC}$ and the state-of-the-art.

Dataset	$SHADE_{CC}$	COPKM	LCVQE	RDPM	TVClust	CECM
Appendicitis	**0.332**	–	0.305	0.284	0.025	−0.006
Breast Cancer	0.911	**1.000**	0.486	0.502	0.000	0.037
Bupa	0.801	**1.000**	−0.005	−0.007	−0.007	0.001
Circles	0.982	**1.000**	−0.003	0.375	0.973	0.105
Ecoli	0.025	–	0.387	0.372	**0.719**	–
Glass	0.031	–	0.280	0.253	0.229	–
Haberman	0.796	**1.000**	−0.002	0.075	0.219	−0.050
Hayesroth	0.099	–	0.106	0.107	**0.150**	0.135
Ionosphere	0.591	**1.000**	0.178	0.212	0.000	0.057
Iris	0.260	–	**0.730**	0.547	0.421	0.684
Led7Digit	0.007	–	0.425	**0.490**	0.324	–
Monk2	0.945	**1.000**	0.072	0.170	−0.002	0.007
Moons	0.995	**1.000**	0.241	0.436	0.987	0.095
Movement Libras	0.001	−0.742	**0.293**	0.255	0.000	–
Pima	**0.762**	–	0.076	0.075	1.000	0.000
Saheart	0.945	**1.000**	0.020	0.037	0.221	0.000
Sonar	**0.890**	–	0.004	0.019	0.000	0.000
Soybean	0.422	0.584	0.545	**0.605**	0.000	0.000
Spectfheart	0.778	**0.983**	−0.107	−0.117	0.000	−0.070
Spiral	**0.949**	–	−0.003	0.014	0.006	0.051
Tae	**0.066**	–	0.008	−0.004	0.062	0.000
Vehicle	0.018	**1.000**	0.121	0.081	0.576	–
Vowel	0.001	–	0.063	−0.003	**0.073**	–
Wdbc	0.930	**1.000**	0.486	0.502	0.000	0.000
Zoo	0.128	0.435	**0.642**	0.450	0.353	–
Mean	**0.507**	0.01	0.213	0.229	0.253	−0.238

Table A.16
Experimental results obtained for CS_{20} comparing $SHADE_{CC}$ and the state-of-the-art.

Dataset	$SHADE_{CC}$	COPKM	LCVQE	RDPM	TVClust	CECM
Appendicitis	**1.000**	–	0.305	0.331	0.012	−0.006
Breast Cancer	0.977	**1.000**	0.486	0.502	0.000	0.000
Bupa	0.993	**1.000**	−0.005	−0.007	−0.006	0.000
Circles	**1.000**	**1.000**	−0.003	0.629	**1.000**	0.138
Ecoli	0.047	–	0.387	0.459	**0.763**	–
Glass	0.082	–	0.271	**0.287**	0.218	–
Haberman	0.997	**1.000**	−0.002	0.106	0.737	0.000
Hayesroth	**0.531**	–	0.106	0.107	0.208	0.056
Ionosphere	0.993	**1.000**	0.173	0.309	0.000	0.115
Iris	0.553	–	**0.708**	0.540	0.585	0.684
Led7Digit	0.005	–	0.425	**0.571**	0.327	–
Monk2	0.991	**1.000**	0.072	0.253	−0.002	0.160
Moons	0.995	**1.000**	0.241	0.831	**1.000**	0.180
Movement Libras	0.004	–	**0.293**	**0.293**	0.000	–
Pima	0.899	**1.000**	0.076	0.076	**1.000**	−0.005
Saheart	0.981	**1.000**	0.018	0.026	**1.000**	−0.006
Sonar	**1.000**	**1.000**	0.004	0.127	0.000	0.003
Soybean	0.563	−0.218	0.545	**0.631**	0.000	−0.016
Spectfheart	**1.000**	**1.000**	−0.107	−0.112	0.000	−0.054
Spiral	0.998	**1.000**	−0.003	0.011	0.006	0.045
Tae	**0.392**	–	0.008	0.000	0.035	0.000
Vehicle	0.044	**1.000**	0.122	0.081	0.689	–
Vowel	0.003	–	0.063	−0.003	**0.071**	–
Wdbc	0.972	**1.000**	0.486	0.522	0.000	0.000
Zoo	0.153	**0.821**	0.642	0.439	0.335	–
Mean	**0.647**	0.224	0.212	0.28	0.319	−0.228

equivalent or better when considering the quality of the solutions, especially in cases where large sets of constraints are analyzed.

Given that the constrained clustering problem is an **NP**-complete problem, we found that population-based algorithms are a promising option to solve it. This is because, apart from the benefits of their exploration capabilities (which SHADE excels at), they are highly independent of the problem at hand and the main implementation concern is the fitness function. In the case of constrained clustering, and by choosing a label-based representation, the intra-cluster similarity part of the fitness function can be factorized in order to make it more efficient. Also, it should be noted that, if constraints are given in the form of a list, the penalty term can be computed in linear time with respect to the number of constraints. With all of this in mind, and considering that population-based algorithms can be exported to the big data framework without loss of generality, we think that the results presented in our paper do support the suitability of DE-based algorithms to solve potential big data constrained clustering applications.

Regarding future research lines, and although the main purpose of this paper is to demonstrate the suitability of DE based strategies to solve the constrained clustering problem, it is also worth mentioning existing variants of it (other than SHADE) that look promising. A DE multimodal variant is proposed in [46], which can be applied to constrained clustering, not only to find a good partition, but also to find multiple and potentially different ones. Other options like multi-population variants such as EDEV [47], composite trial vectors and parameter control variants such as CoDE [48], and parameter and mutation strategies ensembles DE variants such as [49] should also be considered for future work.

CRediT authorship contribution statement

Germán González-Almagro: Methodology, Software, Implementation, Writing, Visualization. **Julián Luengo:** Supervision, Validation, Writing - review & editing. **José-Ramón Cano:** Supervision, Validation, Writing - review & editing. **Salvador García:** Conceptualization, Supervision, Validation, Writing - review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

Our work has been supported by the research projects TIN2017-89517-P and PP2019.PRI.I.06.

Appendix. Numerical results

See Tables A.11–A.16.

References

[1] B.S. Everitt, S. Landau, M. Leese, Cluster Analysis, fourth ed., Wiley Publishing, 2009.
[2] E.J. Araújo, A.A. Chaves, L.A. Lorena, Improving the clustering search heuristic: An application to cartographic labeling, Appl. Soft Comput. 77 (2019) 261–273.
[3] Z. Wang, Y. Yang, A non-iterative clustering based soft segmentation approach for a class of fuzzy images, Appl. Soft Comput. 70 (2018) 988–999.
[4] H. Verma, R. Agrawal, A. Sharan, An improved intuitionistic fuzzy c-means clustering algorithm incorporating local information for brain image segmentation, Appl. Soft Comput. 46 (2016) 543–557.
[5] J. Aparajeeta, P.K. Nanda, N. Das, Modified possibilistic fuzzy C-means algorithms for segmentation of magnetic resonance image, Appl. Soft Comput. 41 (2016) 104–119.
[6] X. Wu, V. Kumar, The Top Ten Algorithms in Data Mining, first ed., Chapman & Hall/CRC, 2009.
[7] S. Guha, R. Rastogi, K. Shim, CURE: an efficient clustering algorithm for large databases, in: ACM Sigmod Record, Vol. 27, ACM, 1998, pp. 73–84.
[8] A.K. Jain, M.N. Murty, P.J. Flynn, Data clustering: a review, ACM Comput. Surv. (CSUR) 31 (3) (1999) 264–323.
[9] O. Chapelle, B. Schlkopf, A. Zien, Semi-Supervised Learning, first ed., The MIT Press, 2010.
[10] I. Triguero, S. García, F. Herrera, Self-labeled techniques for semi-supervised learning: taxonomy, software and empirical study, Knowl. Inf. Syst. 42 (2) (2015) 245–284.
[11] I. Davidson, S. Ravi, Clustering with constraints: Feasibility issues and the k-means algorithm, in: Proceedings of the 2005 International Conference on Data Mining, SIAM, 2005, pp. 138–149.
[12] S.H. Semnani, O.A. Basir, P. Van Beek, Constrained clustering for flocking-based tracking in maneuvering target environment, Robot. Auton. Syst. 83 (2016) 243–250.
[13] A. Seret, T. Verbraken, B. Baesens, A new knowledge-based constrained clustering approach: Theory and application in direct marketing, Appl. Soft Comput. 24 (2014) 316–327.
[14] S.T. Mai, S. Amer-Yahia, S. Bailly, J.-L. Pépin, A.D. Chouakria, K.T. Nguyen, A.-D. Nguyen, Evolutionary active constrained clustering for obstructive sleep apnea analysis, Data Sci. Eng. 3 (4) (2018) 359–378.
[15] J. Li, Y. Xia, Z. Shan, Y. Liu, Scalable constrained spectral clustering, IEEE Trans. Knowl. Data Eng. 27 (2) (2015) 589–593.
[16] Y. Wang, Y. Xiang, J. Zhang, W. Zhou, G. Wei, L.T. Yang, Internet traffic classification using constrained clustering, IEEE Trans. Parallel Distrib. Syst. 25 (11) (2014) 2932–2943.
[17] A. Brieden, P. Gritzmann, F. Klemm, Constrained clustering via diagrams: A unified theory and its application to electoral district design, European J. Oper. Res. 263 (1) (2017) 18–34.
[18] K. Wagstaff, C. Cardie, S. Rogers, S. Schrödl, Constrained K-means clustering with background knowledge, in: Proceedings of the Eighteenth International Conference on Machine Learning, Morgan Kaufmann Publishers Inc., 2001, pp. 577–584.
[19] P.S. Bradley, K.P. Bennett, A. Demiriz, Constrained K-Means Clustering, Tech. rep., MSR-TR-2000-65, Microsoft Research, 2000.
[20] I. Davidson, S. Basu, A survey of clustering with instance level constraints, ACM Trans. Knowl. Discovery Data 1 (2007) 1–41.
[21] J. Schmidt, E.M. Brandle, S. Kramer, Clustering with attribute-level constraints, in: 2011 IEEE 11th International Conference on Data Mining, IEEE, 2011, pp. 1206–1211.
[22] J. Wang, S. Wu, G. Li, Clustering with instance and attribute level side information, Int. J. Comput. Intell. Syst. 3 (6) (2010) 770–785.
[23] M.H. Law, A. Topchy, A.K. Jain, Clustering with soft and group constraints, in: Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR), Springer, 2004, pp. 662–670.
[24] F. Fausto, A. Reyna-Orta, E. Cuevas, Á.G. Andrade, M. Perez-Cisneros, From ants to whales: metaheuristics for all tastes, Artif. Intell. Rev. 53 (1) (2020) 753–810.
[25] R.M. de Oliveira, A.A. Chaves, L.A.N. Lorena, A comparison of two hybrid methods for constrained clustering problems, Appl. Soft Comput. 54 (2017) 256–266.
[26] X. Xu, L. Lu, P. He, Z. Pan, L. Chen, Improving constrained clustering via swarm intelligence, Neurocomputing 116 (2013) 317–325.
[27] S. Das, P.N. Suganthan, Differential evolution: a survey of the state-of-the-art, IEEE Trans. Evol. Comput. 15 (1) (2011) 4–31.
[28] D. Molina, A. LaTorre, F. Herrera, An insight into bio-inspired and evolutionary algorithms for global optimization: review, analysis, and lessons learnt over a decade of competitions, Cogn. Comput. 10 (2018) 517–544.
[29] V. Antoine, B. Quost, M.-H. Masson, T. Denoeux, CECM: Constrained evidential C-means algorithm, Comput. Statist. Data Anal. 56 (4) (2012) 894–914.
[30] M.-H. Masson, T. Denoeux, ECM: An evidential version of the fuzzy c-means algorithm, Pattern Recognit. 41 (4) (2008) 1384–1397.
[31] D. Pelleg, D. Baras, K-means with large and noisy constraint sets, in: European Conference on Machine Learning, Springer, 2007, pp. 674–682.
[32] D. Khashabi, J. Wieting, J.Y. Liu, F. Liang, Clustering with side information: From a probabilistic model to a deterministic algorithm, 2015, arXiv preprint arXiv:1508.06235.
[33] R. Storn, K. Price, Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces, J. Glob. Optim. 11 (4) (1997) 341–359.
[34] D. Goldberg, Genetic Algorithms, Search, Optimization & Machine Learning, Addison-Wesley, 1989.
[35] J.C. Bean, Genetic algorithms and random keys for sequencing and optimization, ORSA J. Comput. 6 (2) (1994) 154–160.
[36] J.F. Gonçalves, M.G. Resende, Biased random-key genetic algorithms for combinatorial optimization, J. Heuristics 17 (5) (2011) 487–525.
[37] R. Tanabe, A. Fukunaga, Success-history based parameter adaptation for differential evolution, in: Evolutionary Computation (CEC), 2013 IEEE Congress on, IEEE, 2013, pp. 71–78.
[38] J. Zhang, A.C. Sanderson, JADE: adaptive differential evolution with optional external archive, IEEE Trans. Evol. Comput. 13 (5) (2009) 945–958.
[39] F. Peng, K. Tang, G. Chen, X. Yao, Multi-start JADE with knowledge transfer for numerical optimization, in: Evolutionary Computation, 2009. CEC'09. IEEE Congress on, IEEE, 2009, pp. 1889–1895.
[40] I. Triguero, S. González, J.M. Moyano, S. García, J. Alcalá-Fdez, J. Luengo, A. Fernández, M.J. del Jesús, L. Sánchez, F. Herrera, KEEL 3.0: an open source software for multi-stage analysis in data mining, Int. J. Comput. Intell. Syst. 10 (1) (2017) 1238–1249.
[41] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, J. Mach. Learn. Res. 12 (2011) 2825–2830.
[42] L. Hubert, P. Arabie, Comparing partitions, J. Classification 2 (1) (1985) 193–218.
[43] W.M. Rand, Objective criteria for the evaluation of clustering methods, J. Amer. Statist. Assoc. 66 (336) (1971) 846–850.
[44] A. Benavoli, G. Corani, J. Demšar, M. Zaffalon, Time for a change: a tutorial for comparing multiple classifiers through Bayesian analysis, J. Mach. Learn. Res. 18 (1) (2017) 2653–2688.
[45] J. Carrasco, S. García, M. del Mar Rueda, F. Herrera, RNPBST: An r package covering non-parametric and Bayesian statistical tests, in: International Conference on Hybrid Artificial Intelligence Systems, Springer, 2017, pp. 281–292.
[46] B.-Y. Qu, P.N. Suganthan, J.-J. Liang, Differential evolution with neighborhood mutation for multimodal optimization, IEEE Trans. Evol. Comput. 16 (5) (2012) 601–614.
[47] G. Wu, X. Shen, H. Li, H. Chen, A. Lin, P.N. Suganthan, Ensemble of differential evolution variants, Inform. Sci. 423 (2018) 172–186.
[48] Y. Wang, Z. Cai, Q. Zhang, Differential evolution with composite trial vector generation strategies and control parameters, IEEE Trans. Evol. Comput. 15 (1) (2011) 55–66.
[49] R. Mallipeddi, P.N. Suganthan, Q.-K. Pan, M.F. Tasgetiren, Differential evolution algorithm with ensemble of parameters and mutation strategies, Appl. Soft Comput. 11 (2) (2011) 1679–1696.