Computers and Operations Research 121 (2020) 104979

Contents lists available at ScienceDirect

Computers and Operations Research

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

DILS: Constrained clustering through dual iterative local search

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 29 July 2019
Revised 20 March 2020
Accepted 22 April 2020
Available online 27 April 2020

Keywords:
Constrained clustering
Instance-level
Must-link
Cannot-link
Dual iterative local search

ABSTRACT

Clustering has always been a powerful tool in knowledge discovery. Traditionally unsupervised, it has received renewed attention recently as it has 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 is the focus of this paper. We propose a new metaheuristic algorithm, the Dual Iterative Local Search, and prove its ability to produce quality results for the constrained clustering problem. We compare the results obtained by this proposal to those obtained by 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

Clustering is one of the most well-known and extensively studied data analysis problems. It constitutes a key research area in the field of unsupervised learning, where there is no supervision on how the information should be handled. 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, and each instance is 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, and can be obtained with some kind of distance measurement (Jain et al., 1999).

Semi-supervised learning (SSL) is a machine learning paradigm that arises from adding incomplete information to unsupervised learning (Chapelle et al., 2010). Following this paradigm we can incorporate background information to the clustering process, resulting in constrained clustering, which is the main subject of the study presented in this paper (Triguero et al., 2015). The objective of constrained clustering 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 the following are worth mentioning: advanced robotics applications (Semnani et al., 2016), hyperspectral image classification (Wu and Prasad, 2017), applied marketing (Seret et al., 2014), obstructive sleep apnea analysis (Mai et al., 2018), terrorist sub-communities detection (Saidi et al., 2018), vocabulary maintenance policy for case-based reasoning systems (Ayed et al., 2019), electoral district designing, Brieden et al. (2017), and lane finding in GPS data (Wagstaff et al., 2001) among others.

Constraints can be understood in different ways, resulting in three main types of constrained clustering: cluster-level (Bradley et al., 2000), instance-level (Davidson and Basu, 2007) and feature-level constrained clustering (Schmidt et al., 2011). Moreover, hybrid approaches which try to integrate different types of constraints have also been proposed (Wang et al., 2010). In particular, we can find two types of instance-level constraints in the literature: pairwise constraints and distance-based constraints. Specifically, 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. This paper focuses on these types of 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 (Wagstaff et al., 2001) and soft (Law et al., 2004) constraints. Hard

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

https://doi.org/10.1016/j.cor.2020.104979
0305-0548/

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 (Seret et al., 2014). For the purpose of this paper, we will employ soft constraints, given their practical applicability to real-world problems.

Finding the optimal partition in a dataset, with respect to any kind of reasonable criteria, is known as an **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 (Davidson and Ravi, 2005).

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, its resolution poses a tough challenge, and metaheuristics are presented as a promising option to find quality approximate solutions. Metaheuristic algorithms are designed to explore the solution space of a problem with the guidance of a fitness (heuristic) function (Gendreau and Potvin, 2010). In this type of approach the key is to find a good exploration-exploitation trade-off. For this reason the Iterated Local Search (ILS) metaheuristic algorithm, derived from Local Search (LS), introduces periodic perturbations in the LS exploration of the solution space process to escape local optima (Lourenço et al., 2010). ILS—or variants of it—has been applied to a wide variety of problems, including the traveling salesman (Archetti et al., 2018), the quadratic multiple knapsack (Avci and Topaloglu, 2017) or the vehicle routing problem (Chentli et al., 2018; Estrada-Moreno et al., 2019). ILS is also very commonly used in the design of hybrid methods with good exploration-exploitation tradeoff, such as its combination with expediting mechanisms (Zohali et al., 2019) or with the success-history based differential evolution algorithm (Zhao et al., 2019).

Metaheuristics methods have been successfully applied to the clustering problem (Hruschka et al., 2009; José-García and Gómez-Flores, 2016; Nanda and Panda, 2014); although very little work has been done in investigating their suitability for the constrained clustering problem, particularly in highly constrained environments. Encouraged by this success, we propose a new ILS variant, which we call Dual Iterative Local Search (DILS) and which constitutes the main contribution of this paper. DILS seeks to explore the solution space by combining the exploitation capability of ILS and classic diversity-introducing techniques used in the metaheuristics field such as recombination and mutation operators. DILS is also able to adapt to the problem at hand by implementing a restarting mechanism to avoid local optima. To carry out these tasks DILS optimizes two individuals at the same time, which allows it to recombine and compare them in order to manage the exploration of the solution space. We have built the application of DILS to constrained clustering—which we call $DILS_{CC}$—with the aid of an integer-based problem representation and a penalty-style fitness function. We have also proved that it constitutes a competitive approach to obtain quality results for the constrained clustering problem. Particularly, we have shown how the DILS approach is able to scale the quality of the results as the number of constraints increases, making it suitable for highly constrained problems. This is due to the exploitation capability DILS exhibits when exploring the solution space.

Regarding the organization of this paper, Section 2 reviews the existing knowledge concerning constrained clustering and the state-of-the-art. In Sections 3 and 4 the DILS algorithm and its formulation for constrained clustering $DILS_{CC}$ will be reviewed. Sections 5 to 7 present the experimental setup, the results and their analysis, respectively. Finally, in Section 9 the conclusions 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).

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, 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 $x_d \in c_j$ (Davidson and Basu, 2007).

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 $x_d \in c_j$ (Davidson and Basu, 2007).

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 instance-level constrained clustering was defined in (Davidson and Ravi, 2005) as can be seen in Definition 1.

Definition 1Feasibility Problem. : given a dataset X, a constraint set CS, and the bounds on the number of clusters $k_l \leq k \leq k_u$, is there a partition C of X with k clusters such that all constraints in CS are satisfied? (Davidson and Ravi, 2005)

In Davidson and Ravi (2005) 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.

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 Davidson and Ravi (2005).

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

Table 1
Feasibility problem complexity (Davidson and Ravi, 2005).

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

2.3. Constrained clustering state-of-the-art methods

The first adaptation of a classic clustering method for constrained clustering was proposed in (Wagstaff et al., 2001). It involved modifying the widely studied K-means algorithm to take into account instance-level constraints: the already-known ML and CL. This method, named COP-kmeans, introduces a modification to the assignment 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 constraints.

Within the fuzzy clustering family of methods, Constrained Evidential c-means (CECM), a variant of the Evidential c-means (ECM Masson and Denoeux, 2008) algorithm, is proposed in (Antoine et al., 2012). 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 Davidson and Ravi, 2005) is proposed in Pelleg and Baras (2007). 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 Khashabi et al. (2015). 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 is characterized by the fact that the number of clusters (k) does not need to be specified.

An adaptation of the Biased Random-Key Genetic Algorithm (BRKGA) (Gonçalves and Resende, 2011)—a variant of the Random-Key genetic algorithm (Bean, 1994)—for the constrained clustering problem, the BRKGA+LS method, is proposed in (de Oliveira et al., 2017). It uses a genetic algorithm, combined with a LS procedure and guided by a penalty-style fitness function, to obtain a partition of the input dataset. It constitutes one of the few approaches to constrained clustering from the point of view of evolutionary computing. An adaptive version of the BRKGA+LS algorithm (A-BRKGA) can also be found in the literature (Chaves et al., 2018). Even if it has not yet been applied to constrained clustering, the reader may consider this method for future comparisons.

3. The dual iterative local search method

The Dual Iterative Local Search (DILS) is a new variant of the classic ILS method (Lourenço et al., 2010). Its goal is to perform a search in the solution space to find the solution with the best fitness value (given by fitness function f) by introducing diversity in an adaptive and guided way. While ILS works with a single individual, DILS keeps two of them $\{m_b, m_w\}$ in memory at all times, which allows it to guide diversity-inducing methods and to avoid local optima. The individual m_b provides the best fitness value, whereas m_w provides the worst fitness value at the end of each stage of the optimization process. These stages involve generating, evaluating and optimizing new individuals, and we refer to them as generations. Successive generations will eventually lead to finding better individuals (in terms of f).

Recombination and mutation DILS builds a new individual in each generation G of the optimization process by applying a recombination operator to m_b and m_w. After that, it applies a strong mutation operator to the newly generated individual. This is the individual DILS applies the LS procedure to in order to improve its fitness value, resulting in the trial individual m_t. The trial individual m_t replaces the worst of its predecessors, m_w, if it achieves a better fitness value; this operation represents the acceptance criterion for DILS. Eq. (1) displays the expression for this criterion in the case of a minimization problem.

$$m_{w,G} = \begin{cases} m_t & \text{if } f(m_t) < f(m_w) \\ m_{w,G} & \text{otherwise} \end{cases} \tag{1}$$

Reinitialization method The process described so far will most likely fall into a local optimum, effectively stopping the exploration of the solution space. To avoid it, DILS implements a reinitialization method for m_w based on the differences between the two individuals it keeps in memory ($\{m_b, m_w\}$). Eq. (2) shows the reinitialization criterion for a minimization problem. RandInit() is a function that returns a randomly initialized individual and $\xi \in [-1, 1]$ is the parameter that controls the tolerance of the reinitialization method.

$$m_{w,G+1} = \begin{cases} \text{RandInit}() & \text{if } f(m_b) - f(m_w) > f(m_b) * \xi \\ m_{w,G} & \text{if } f(m_b) - f(m_w) \leq f(m_b) * \xi \wedge f(m_b) < f(m_w) \\ m_{b,G} & \text{if } f(m_b) - f(m_w) \leq f(m_b) * \xi \wedge f(m_b) > f(m_w) \end{cases}. \tag{2}$$

Maintaining consistency In order to maintain consistency between generations we also need to reassign m_b to the true best individual once the reinitialization criterion has been tested, as shown in Eq. (3). As a result, the best individual is always preserved and therefore always takes part in the process of generating new individuals (via the recombination operator) as the best predecessor.

$$m_{b,G+1} = \begin{cases} m_{b,G} & \text{if } f(m_b) < f(m_w) \\ m_{w,G} & \text{if } f(m_b) > f(m_w) \end{cases} \tag{3}$$

Fig. 1 summarizes the DILS optimization process. Arrows in red indicate stages of the algorithm where m_b and m_w are reassigned to the true best and worst individual respectively. Consequently, at the point where the reinitialization criterion is tested, it is possible for $f(m_w)$ to yield a smaller value than $f(m_b)$, so that $f(m_b) - f(m_w)$ can be either positive or negative.

On the influence of ξ Let us consider a minimization problem again. On the one hand, if $\xi > 0$, then the true worst individual is only reinitialized when m_t replaces m_w and is also better than m_b by a margin. On the other hand, if $\xi < 0$, we allow DILS to reinitialize the true worst individual even when m_t is not better than m_b. It becomes clear that when $\xi = 0$ the worst individual is restarted if $f(m_b) = f(m_w)$. Note that ξ could take any real value, but it is not realistic to set it outside of the range $[-1, 1]$, and we recommend $[-0.5, 0.5]$ to preserve a good exploration-exploitation tradeoff. Otherwise ξ would bias the search towards a random-restart LS in the case of $\xi < -0.5$, or towards a simple-path optimization process applied to two individuals in the case of $\xi > 0.5$. To summarize, the closer ξ is to 1 the more restrictive the reinitialization criterion is—in the minimization case. It is also worth noting that in early stages of the optimization process

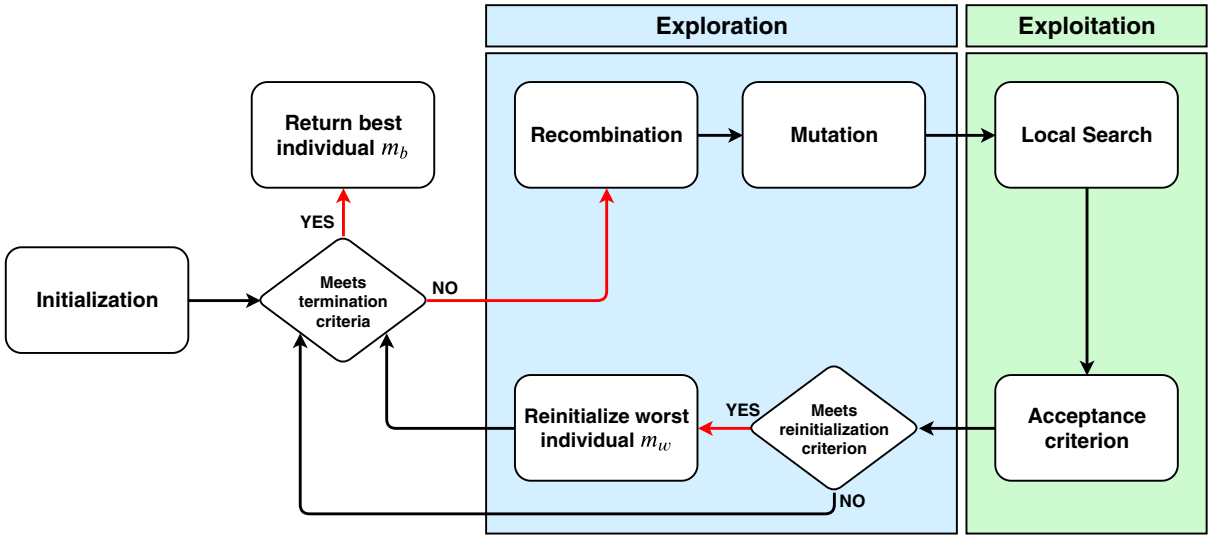


Fig. 1. Diagram summarizing the DILS optimization process.

the reinitialization criterion tends to be met more often than in later stages. The reason for this is that the fitness value of the best individual $f(m_b)$ never worsens ($f(m_{b,G}) \geq f(m_{b,G+1})$), so it becomes increasingly harder for m_w to achieve similar or better scores. Algorithm 1 summarizes the overall DILS optimization process.

Algorithm 1: DILS.

Input: Probability for recombination operator p_r, segment size for mutation operator p_s, reinitialization method tolerance ξ.

```
// Initialization phase
G ← 0
m_b ← RandInit(); m_w ← RandInit()
// Main loop
while Termination criteria are not met do
    // Find best and worst individual
    m_b ← Best({m_b, m_w}); m_w ← Worst({m_b, m_w})
    // Generate new individual
    m_t ← Recombination(m_b, m_w)
    m_t ← Mutation(m_t)
    // Improve new individual
    m_t ← LocalSearch(m_t)
    // Apply replacement operator
    if f(m_t) < f(m_w) then
        m_w ← m_t
    end
    // Check restart criterion
    if f(m_b) − f(m_w) > f(m_b) × ξ then
        m_b ← Best({m_b, m_w})
        m_w ← RandInit()
    end
    G ← G + 1
end
m_b ← Best({m_b, m_w})
return m_b
```

cess.

For the recombination and mutation operators that appear in Algorithm 1, we use the uniform recombination and the segment mutation operator respectively. The uniform recombination consists in selecting features from the best individual m_b based on a given probability p_r to introduce them in the resulting individual, so the rest of the features are taken from m_w (Spears and De Jong, 1995). The segment mutation operator consists in replacing a fixed-size segment from the features of the individual with randomly generated features. The size of the segment p_s must be fixed at the beginning of the optimization process. The starting point of the segment to replace is chosen randomly. Although other (problem-specific) recombination and mutation operators could be used in DILS, we highly recommend the operators described above. The uniform recombination operator provides a non-biased way to include the features of the best solutions found in the search process so far, while the segment mutation operator is able to produce diversity to widen the search space.

It should be noted that the optimization process described so far is applicable to any minimization problem, although it is clear that there is a direct adaptation of DILS for maximization problems. In addition, the LS procedure used to obtain the trial individual m_t must be specified for each problem, as well as numerical representation details and termination criteria.

On the complexity of DILS Regarding the algorithmic complexity of the proposed DILS method, the mutation and recombination operators are the only complex algorithmic operations that have been explicitly defined. Both of them can be performed in linear time $\mathcal{O}(n)$, with n being the length of the vector representing the solution to the problem. However, the fitness function f and the LS procedure need to be defined for each problem and will likely represent the algorithmic bottleneck for the optimization process, as most hard problems (suitable to be addressed with DILS) will make use of complex evaluation and local optimization methods.

4. DILS Application scheme for constrained clustering

In this section, we build the application scheme of DILS for constrained clustering. We will call this new approach DILS$_{CC}$. We will use an integer-based representation for all the individuals DILS$_{CC}$ works with; this way, each individual m is defined as a vector of n integers—with n being the number of instances of the dataset. Each integer $m_i \in [0, k)$ in m represents the label of the cluster that instance x_i is assigned to, and therefore each individual m is a solution to the clustering problem. We have chosen a label-based representation because of its straightforward application to DILS operators such as recombination and mutation operators. Also notice that the neighborhood-generation operator from the LS procedure benefits from a label-based representation by simplifying the

tracking of the already generated neighbors. Additionally, in order not to bias the exploration of the solution space, we will use random initialization to set m_b and m_w; this procedure will also be used in the reinitialization of m_w.

When it comes to the diversity-introducing operators, the recommended uniform recombination and segment mutation operators are used. Please note that no repairing procedure is applied to the newly generated solutions (using the mentioned operators), so the resulting solutions will most likely violate an unspecified number of constraint. Experimental results show that there is no need for such a repairing procedure, as it would highly bias the exploration of the solution space.

Regarding the fitness function f, for each individual m its fitness value f_m can be calculated as seen in Eq. (4).

$$f_m = z_m * \overbrace{(\text{infeasibility}_m + 1)}^{penalty}, \tag{4}$$

where infeasibility_m is the number of unsatisfied constraints and z_m is the within-cluster-sum-of-squares that can be computed as shown in Eq. (5). As infeasibility is used as a scale factor, this fitness function allows DILS to clearly identify solutions satisfying different numbers of constraints. Let us remember that, even though we make a soft interpretation of the constraints, they must be used as a strong guide for the optimization process. The fitness function is designed to favor the exploration of the solution space from the point of view of the constraint set, so that in Fig. 1 the exploration module optimizes the penalty term in Eq. (5) while the exploitation module optimizes z_m. That said, both the exploration and the exploitation module share the overall fitness optimization workload without being completely independent from each other.

$$z_m = \sum_{c_i \in C_m} \left[\frac{\sum_{x_a, x_b \in c_i} d^2(x_a, x_b)}{|c_i|} \right]. \tag{5}$$

We also need to specify a LS procedure for DILS to be able to improve the mutant individual to obtain the trial vector m_t. Algorithm 2 shows the LS used in DILS_{CC}. In each LS iteration a random index $i \in [1, n]$ (instance x_i) is chosen to explore its neighborhood, which is generated by changing its label l_i (moving it from one cluster to another) in a random fashion, in order not to bias the search. When a neighbor is found to be better ($f(m') < f(m)$) it replaces the current solution m and neighborhood generation stops. If no improvement is found then a new random index that has not already been explored is selected. If all indices have been explored the explored set is set to empty. Note that, in order not to bias the search, indices from vector m and labels from the set of labels are randomly chosen to generate the neighborhood of m. We also establish the maximum number of neighbors p_m that can be generated in each call; this is done to ensure a better exploration-exploitation tradeoff.

On the complexity of DILS_{CC} When it comes to the algorithmic complexity of the DILS optimization scheme for constrained clustering, we need to first focus con the defined specific LS procedure. The maximum number of iterations is given by the p_m parameter, and in each iteration the entire set of labels could be analyzed ($p_m{}^*k$). Additionally we need to study the complexity of the fitness function f, which is also specific to the constrained clustering problem. Constraints can be given in the form of lists or matrices. Computing the infeasibility of a newly generated partition of a given dataset require $n^2/2 - 1$ comparisons when the constraints are given in matrix form (as the matrix must be symmetrical), and $|C_=| + |C_{\neq}|$ when they are given in the form of list. Note that, unless the complete graph of constraints is available, $|C_=| + |C_{\neq}|$ will always be smaller than $n^2/2 - 1$, so we choose the list representation of constraint for the implementation of DILS_{CC}. The real bottleneck of DILS_{CC} is found in the computation of the within-cluster-sum-of-squares (z_m). This can be done by first computing the centroid of each cluster, which involve iterating over the dataset, and then computing the distance of each instance the de centroid of the cluster it has been assigned to, which again involves iterating over the entire dataset. With this, the computation of the within-cluster-sum-of-squares can be done in 2^*n^*u operations. Bearing this in mind, we can write the algorithmic complexity of DILS_{CC} as in Expression 6.

Algorithm 2: Local search.

Input: Dataset X, constraint sets $C_=$ and $C_{\neq}$, individual (solution) m, number of clusters k, maximum number of neighbors that can be generated p_m.

```
[1]  explored ← ∅
[2]  while improvement and generated < p_m do
[3]      improvement ← false
         // Select random index (object) from m
[4]      i ← Rand({1, ..., n} − explored)
[5]      explored ← explored ∪ {i}
         // Random shuffle labels set
[6]      RSL ← RandomShuffle({1, ..., k})
[7]      for l ∈ RSL and while not improvement do
[8]          m' ← m
             /* Move object i from m to the cluster associated
                with label l                                  */
[9]          m'_i ← l
[10]         if f(m') < f(m) then
[11]             m ← m'
[12]             improvement ← true
[13]         end
[14]         generated ← generated + 1
[15]     end
[16]     if |explored| == n then
[17]         explored ← ∅
[18]     end
[19] end
[20] return m
```

$$\mathcal{O}(p_m \times k \times (\overbrace{\underbrace{|C_=| + |C_{\neq}|}_{\text{Infeasibility}} \times \underbrace{n \times u}_{z_m}}^{\text{Fitness Function } f})) \tag{6}$$

5. Experimental setup

For our experiments we will compare the results obtained by DILS_{CC} and state-of-the-art methods 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] (Triguero et al., 2017), although some of them have been obtained via ⟨12:monospace ⟩ scikit-learn⟨/12:monospace⟩pythonpackage [2] (Pedregosa et al., 2011). We also include 3 artificial datasets in our analysis, namely: *Circles*, *Moons* and *Spiral*, which can be found at GitHub [3]. Table 2 displays a summary of every dataset.

Classification datasets are commonly used in the literature to test constrained clustering algorithms; the reason behind this is that they enable us to generate constraints with respect to the true labels (see Section 5.1). They also facilitate an easy evaluation of

[1] https://sci2s.ugr.es/keel/category.php?cat=clas
[2] https://scikit-learn.org/stable/datasets/index.html
[3] https://github.com/GermangUgr/DILS_CC

Table 2
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
Heart	270	2	13
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
Newthyroid	215	3	5
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
Wine	178	3	13
Zoo	101	7	16

Table 3
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
Heart	178	173	396	424	744	687
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
Newthyroid	108	123	270	258	449	454
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	10776
Wine	49	104	121	230	217	413
Zoo	21	34	29	91	41	169

the quality of the algorithm by means of measures like the Adjusted Rand Index (see Section 5.2).

5.1. Constraint generation

Since we have the true labels associated with each dataset, we will use the method proposed in (Wagstaff et al., 2001) to generate artificial constraint sets. This method consists of randomly selecting two instances of a dataset, then comparing its labels, and finally 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. Table 3 shows the number of constraints of each type obtained for each dataset. All constraint sets used in our experiments are available here [4]

Note that the greater the number of classes present in the dataset, the fewer ML constraints that are obtained with the method proposed in (Wagstaff et al., 2001). 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 with 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 (Hubert and Arabie, 1985). 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 (Rand, 1971).

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. (7).

$$\text{Rand}(C_1, C_2) = \frac{a+b}{n(n-1)/2} \tag{7}$$

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 seen in Eq. (8).

$$\text{ARI}(C_1, C_2) = \frac{\text{Rand}(C_1, C_2) - \text{Expected Index}}{\text{Maximum Index} - \text{Expected Index}}, \tag{8}$$

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 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 Hubert and Arabie (1985).

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.

[4] https://drive.google.com/drive/u/1/folders/1sjnPYitey8q9zrPKa_YpFS7iNKTT15QH

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 Benavoli et al. (2017) we can find an in-depth analysis of the disadvantages of NHST, and a new model is proposed for carrying out the 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 Benavoli et al. (2017) are based on the trap of black-and-white thinking, that is: to reject, or not to reject?

To start with, NHST do not provide us with the probabilities associated with 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 can not infer anything since NHST can not provide evidence in its favor.

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 Benavoli et al. (2017), 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 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 Carrasco et al. (2017).

The Bayesian sign test 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:

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

5.4. Calibration

Table 4 shows a summary of the parameter setup used for the DILS$_{CC}$.

The stop criterion is given by the *Evals* parameter, which refers to the number of evaluations of the fitness function f and which will be set at 300000. The implementation for DILS$_{CC}$ can be found at GitHub [5].

To compare with the state-of-the-art methods mentioned in Section 2.3 we will use the parameters setup shown in Table 5 for the python implementation that can be found at GitHub [6]. The implementation for the BRKGA+LS algorithm can be found in the same link as DILS$_{CC}$. In all cases the value for k is equal to the number of classes displayed in Table 2.

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 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.

As we have mentioned before, the programming language chosen to implement all methods described in this paper is Python. All our experiments have been carried out in the University of Granada Hercules Computing Server, which features 51 computing nodes with a 2.8 GHz Intel i7 processor, 24 GB of RAM and 1TB SATA2 hard drive disc in each node. Two Gigabit Ethernet internal nets are used to interconnect nodes. Ubuntu 18.04.1 LTS is installed in each node.

6. Experimental results

In this section we present Tables from 6 to 8, which display the results obtained by DILS$_{CC}$ and six state-of-the-art methods to be compared for each dataset and constraint set.

Since some of 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 30 times to each dataset and constraint set, so that the measures shown in the previously mentioned tables correspond to the average of the 30 runs. Therefore, with 25 datasets, 3 constraint sets for each one of them, 7 different methods and 30 runs, we have carried out a total of 15750 experiments for our study.

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.

Table 6 shows the results for the SC_{10} constraint set. It can be observed that DILS$_{CC}$ represents a consistent improvement as compared to all other 6 methods. This is due to the fact that the exploitation capacity of DILS$_{CC}$ allows it to take better advantage of the information contained in the constraint set than the rest of the methods, even with the lowest level of constraint-based information we work with.

The results obtained with the SC_{15} constraint set are presented in Table 7. We observe how DILS$_{CC}$ and COPKM are able to obtain ARI values equal to 1, meaning that the resulting partitions of the datasets perfectly match the true partitions. The fact that DILS$_{CC}$

[5] https://github.com/GermangUgr/DILS_CC

[6] https://github.com/GermangUgr/TFG/tree/master/Software

Table 4
Parameters setup used for DILS$_{CC}$.

Parameter	Meaning	Value
Evals	Fitness function evaluations	300000
p_s	Segment size for the mutation operator	$0.3 \times n$
p_m	Maximum number of neighbors generated in each call to the LS procedure	Evals $\times$ 0.01
p_r	Probability that a feature is selected from m_b for recombination	0.3
ξ	Reinitialization method tolerance	0.0
k	Output partition number of clusters	No. Classes (Table 2)

Table 5
Parameters setup used for the state-of-the-art algorithms.

Method name	Parameters name and values
BRKGA+LS	Evals = 300000; $\|P\| = 100$; $P_e = P_m = 0.2 * \|P\|$; $p_{\text{inherit}} = 50\%$
COPKM	`max_iter` = 300; `tolerance` = $1 * 10^{-4}$; `init_mode` = ‘‘`rand`’’
CECM	`max_iter` = 300; $\alpha = 1$, $\rho = 100$, $\xi = 0.5$, `stop_threshold` = $1 * 10^{-3}$, `init_mode` = ‘‘`rand`’’
LCVQE	`max_iter` = 300; `initial_centroids` = $\emptyset$
RDPM	`max_iter` = 300; $\xi_0 = 0.1$, $\xi_{\text{rate}} = 1$, λ is calculated on the basis of the mean distances in the dataset.
TVClust	`max_iter` = 300; $\alpha_0 = 1.2$, `stop_threshold` = $5 * 10^{-4}$

Table 6
Experimental results obtained for CS_{10} by DILS$_{CC}$ and the state-of-the-art methods.

Dataset	DILS$_{CC}$	BRKGA+LS	COPKM	LCVQE	RDPM	TVClust	CECM
Appendicitis	0.611 ± 0.000	0.118 ± 0.078	-	0.450 ± 0.000	0.267 ± 0.081	0.211 ± 0.184	0.005 ± 0.001
Breast Cancer	0.755 ± 0.040	0.173 ± 0.198	−0.604 ± 0.792	0.917 ± 0.000	0.502 ± 0.000	0.016 ± 0.048	0.005 ± 0.013
Bupa	0.889 ± 0.022	0.355 ± 0.233	-	0.149 ± 0.009	−0.006 ± 0.003	−0.009 ± 0.002	−0.005 ± 0.002
Circles	0.781 ± 0.014	0.104 ± 0.055	-	0.006 ± 0.008	0.243 ± 0.280	0.404 ± 0.377	0.072 ± 0.049
Ecoli	0.039 ± 0.008	0.018 ± 0.008	-	0.404 ± 0.066	0.333 ± 0.057	0.432 ± 0.223	-
Glass	0.008 ± 0.011	0.014 ± 0.012	0.212 ± 0.027	0.229 ± 0.007	0.202 ± 0.077	0.231 ± 0.040	-
Haberman	0.802 ± 0.029	0.020 ± 0.023	−0.807 ± 0.580	0.019 ± 0.003	0.088 ± 0.057	0.332 ± 0.043	0.004 ± 0.010
Hayesroth	0.057 ± 0.033	0.015 ± 0.011	-	0.073 ± 0.046	0.103 ± 0.034	0.064 ± 0.076	0.061 ± 0.013
Heart	0.846 ± 0.046	0.131 ± 0.179	-	0.006 ± 0.007	0.032 ± 0.004	0.223 ± 0.213	0.000 ± 0.008
Ionosphere	0.809 ± 0.040	0.018 ± 0.022	-	0.060 ± 0.000	0.202 ± 0.047	0.004 ± 0.011	0.112 ± 0.094
Iris	0.550 ± 0.065	0.311 ± 0.083	−0.105 ± 0.895	0.769 ± 0.000	0.567 ± 0.064	0.511 ± 0.074	0.405 ± 0.283
Led7Digit	0.013 ± 0.006	0.005 ± 0.003	0.525 ± 0.036	0.503 ± 0.024	0.359 ± 0.031	0.254 ± 0.055	-
Monk2	0.823 ± 0.032	0.404 ± 0.130	0.982 ± 0.000	0.575 ± 0.000	0.092 ± 0.031	0.096 ± 0.292	0.009 ± 0.025
Moons	0.963 ± 0.022	0.227 ± 0.124	-	0.319 ± 0.000	0.310 ± 0.028	0.962 ± 0.027	0.267 ± 0.161
Movement Libras	0.019 ± 0.008	−0.000 ± 0.004	0.283 ± 0.013	0.322 ± 0.017	0.249 ± 0.025	0.000 ± 0.000	-
Newthyroid	0.040 ± 0.053	0.013 ± 0.019	-	0.791 ± 0.009	0.370 ± 0.150	0.695 ± 0.206	−0.004 ± 0.009
Saheart	0.788 ± 0.025	0.163 ± 0.092	0.974 ± 0.000	0.020 ± 0.011	0.028 ± 0.023	0.367 ± 0.372	0.007 ± 0.003
Sonar	0.710 ± 0.068	0.023 ± 0.020	-	0.109 ± 0.000	0.007 ± 0.007	0.000 ± 0.000	−0.003 ± 0.007
Soybean	0.289 ± 0.062	0.342 ± 0.165	0.613 ± 0.152	0.551 ± 0.007	0.635 ± 0.034	0.000 ± 0.000	0.076 ± 0.087
Spectfheart	0.895 ± 0.039	0.257 ± 0.083	-	0.014 ± 0.000	−0.118 ± 0.008	0.000 ± 0.000	−0.024 ± 0.032
Spiral	0.849 ± 0.042	0.386 ± 0.310	-	0.042 ± 0.000	0.015 ± 0.009	0.049 ± 0.041	0.036 ± 0.006
Tae	0.028 ± 0.019	0.022 ± 0.022	-	0.015 ± 0.000	−0.002 ± 0.004	0.052 ± 0.022	−0.000 ± 0.000
Vehicle	0.023 ± 0.009	0.006 ± 0.005	-	0.078 ± 0.001	0.081 ± 0.000	0.250 ± 0.125	0.020 ± 0.013
Wine	0.326 ± 0.051	0.152 ± 0.051	-	0.397 ± 0.000	0.368 ± 0.002	0.376 ± 0.097	0.010 ± 0.008
Zoo	0.221 ± 0.031	0.111 ± 0.064	0.684 ± 0.099	0.710 ± 0.048	0.438 ± 0.110	0.399 ± 0.116	-
Mean	**0.485**	0.136	−0.490	0.301	0.214	0.236	−0-158

is always able to output a partition of the datasets is a noteworthy advantage over COPKM. Even in the cases where COPKM is unable to produce a partition, DILS often finds the optimal or a near-optimal partition.

In Table 8, which shows the results obtained with the SC_{20} constraint set, we can see how the quality of the results of DILS$_{CC}$ scales with the number of constraints, as well as that of methods such as BRKGA+LS or COPKM (the latter being more limited in other aspects). The TVClust method should also be highlighted, as it is able to find the optimal solution in 2 cases, which implies a significant improvement over its performance on previous constraint sets. It is also worth noting that DILS$_{CC}$ provides the best average ARI for each of the three constraint sets analyzed in this paper.

Considering that BRKGA and DILS$_{CC}$ are the two metaheuristic approaches to the constrained clustering problem presented in this paper, we want to highlight the clear improvement that the latter constitutes over BRKGA. Both methods are able to scale the quality of the solutions with the amount of available constraint-based information, although the DILS$_{CC}$ exhibits a better behavior from the smallest constraint set CS_{10}, where we also find the largest average difference between these to methods. The aim of the DILS$_{CC}$ method is to preserve a good exploration-exploitation tradeoff while maintaining an exploitation-oriented general scheme (ILS). This gives DILS$_{CC}$ the ability to deeply exploit certain regions of the solutions space when needed (which we have found to be beneficial to the constrained clustering problem), whereas BRKGA does not includes mechanisms to explicitly exploit a particular local-optima region when it is likely to contain the general optimal solution. Notice that BRKGA does include a local search optimization procedure, making it a memetic algorithm, although it does not transfer the results of this procedure to the population itself, it is only used to update the best solution found so far if required.

7. Statistical analysis of results

With the results obtained by all methods for a total of 75 different datasets—the 25 datasets in combination with the 3 constraint

Table 7
Experimental results obtained for CS_{15} by $DILS_{CC}$ and the state-of-the-art methods.

Dataset	$DILS_{CC}$	BRKGA+LS	COPKM	LCVQE	RDPM	TVClust	CECM
Appendicitis	0.957 ± 0.000	0.418 ± 0.368	-	0.379 ± 0.018	0.335 ± 0.084	0.261 ± 0.201	−0.005 ± 0.003
Breast Cancer	0.792 ± 0.016	0.915 ± 0.031	1.000 ± 0.000	0.979 ± 0.000	0.502 ± 0.000	0.096 ± 0.285	−0.006 ± 0.008
Bupa	0.993 ± 0.006	0.994 ± 0.006	1.000 ± 0.000	−0.002 ± 0.000	−0.006 ± 0.003	0.092 ± 0.303	0.000 ± 0.000
Circles	1.000 ± 0.000	0.987 ± 0.013	1.000 ± 0.000	0.011 ± 0.010	0.408 ± 0.292	0.910 ± 0.270	0.066 ± 0.049
Ecoli	0.091 ± 0.026	0.024 ± 0.012	-	0.513 ± 0.058	0.402 ± 0.103	0.564 ± 0.241	-
Glass	0.076 ± 0.042	0.025 ± 0.009	-	0.286 ± 0.028	0.248 ± 0.016	0.244 ± 0.066	-
Haberman	1.000 ± 0.000	0.490 ± 0.392	1.000 ± 0.000	0.001 ± 0.001	0.079 ± 0.048	0.973 ± 0.000	0.006 ± 0.009
Hayesroth	0.478 ± 0.066	0.131 ± 0.057	-	0.087 ± 0.055	0.111 ± 0.021	0.097 ± 0.082	0.002 ± 0.013
Heart	1.000 ± 0.000	0.980 ± 0.020	1.000 ± 0.000	0.053 ± 0.000	0.030 ± 0.012	0.435 ± 0.433	−0.000 ± 0.001
Ionosphere	0.973 ± 0.009	0.981 ± 0.020	1.000 ± 0.000	0.004 ± 0.000	0.261 ± 0.046	0.004 ± 0.011	0.129 ± 0.094
Iris	0.832 ± 0.051	0.240 ± 0.136	-	0.941 ± 0.000	0.543 ± 0.006	0.524 ± 0.127	0.228 ± 0.323
Led7Digit	0.012 ± 0.002	0.003 ± 0.003	-	0.557 ± 0.029	0.454 ± 0.033	0.261 ± 0.051	-
Monk2	0.899 ± 0.015	0.951 ± 0.031	1.000 ± 0.000	0.671 ± 0.000	0.144 ± 0.028	0.098 ± 0.301	0.100 ± 0.025
Moons	1.000 ± 0.000	0.982 ± 0.018	1.000 ± 0.000	0.639 ± 0.000	0.470 ± 0.065	1.000 ± 0.000	0.299 ± 0.161
Movement Libras	0.018 ± 0.006	0.002 ± 0.002	−0.742 ± 0.516	0.334 ± 0.025	0.255 ± 0.032	0.000 ± 0.000	-
Newthyroid	0.390 ± 0.113	0.104 ± 0.076	-	0.835 ± 0.010	0.455 ± 0.156	0.848 ± 0.191	0.013 ± 0.018
Saheart	0.870 ± 0.021	0.815 ± 0.265	1.000 ± 0.000	0.017 ± 0.000	0.030 ± 0.023	0.808 ± 0.384	0.003 ± 0.003
Sonar	0.981 ± 0.000	0.947 ± 0.067	-	0.037 ± 0.013	0.014 ± 0.013	0.000 ± 0.000	−0.001 ± 0.001
Soybean	0.468 ± 0.045	0.421 ± 0.137	0.656 ± 0.200	0.550 ± 0.011	0.593 ± 0.058	0.000 ± 0.000	0.153 ± 0.232
Spectfheart	1.000 ± 0.000	0.861 ± 0.302	0.983 ± 0.000	0.167 ± 0.000	−0.116 ± 0.008	0.000 ± 0.000	0.032 ± 0.032
Spiral	1.000 ± 0.000	0.594 ± 0.333	-	0.022 ± 0.007	0.011 ± 0.009	0.411 ± 0.482	0.012 ± 0.006
Tae	0.386 ± 0.045	0.077 ± 0.077	-	0.046 ± 0.000	0.002 ± 0.007	0.053 ± 0.018	−0.000 ± 0.000
Vehicle	0.066 ± 0.014	0.023 ± 0.014	1.000 ± 0.000	0.083 ± 0.003	0.081 ± 0.000	0.454 ± 0.159	−0.006 ± 0.007
Wine	0.740 ± 0.047	0.163 ± 0.083	-	0.395 ± 0.000	0.369 ± 0.001	0.430 ± 0.129	0.001 ± 0.002
Zoo	0.193 ± 0.066	0.109 ± 0.066	0.416 ± 0.023	0.724 ± 0.059	0.434 ± 0.112	0.423 ± 0.138	-
Mean	**0.649**	0.489	0.013	0.333	0.244	0.359	−0.159

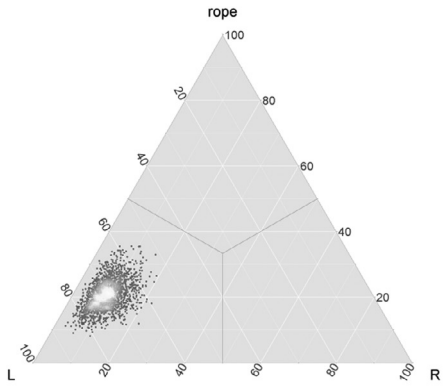
Table 8
Experimental results obtained for CS_{20} by $DILS_{CC}$ and the state-of-the-art methods.

Dataset	$DILS_{CC}$	BRKGA+LS	COPKM	LCVQE	RDPM	TVClust	CECM
Appendicitis	1.000 ± 0.000	1.000 ± 0.000	-	0.050 ± 0.000	0.380 ± 0.184	0.419 ± 0.324	−0.007 ± 0.001
Breast Cancer	0.796 ± 0.009	0.962 ± 0.015	1.000 ± 0.000	0.944 ± 0.000	0.502 ± 0.000	0.098 ± 0.291	0.014 ± 0.013
Bupa	0.988 ± 0.007	0.996 ± 0.005	1.000 ± 0.000	−0.002 ± 0.000	−0.006 ± 0.003	0.093 ± 0.302	−0.002 ± 0.002
Circles	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	0.001 ± 0.002	0.573 ± 0.193	0.911 ± 0.266	0.066 ± 0.049
Ecoli	0.264 ± 0.126	0.039 ± 0.026	-	0.607 ± 0.072	0.427 ± 0.071	0.688 ± 0.243	-
Glass	0.258 ± 0.131	0.084 ± 0.040	-	0.216 ± 0.012	0.276 ± 0.017	0.344 ± 0.165	-
Haberman	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	0.001 ± 0.000	0.102 ± 0.053	1.000 ± 0.000	0.002 ± 0.010
Hayesroth	0.816 ± 0.044	0.395 ± 0.136	-	0.132 ± 0.048	0.114 ± 0.027	0.266 ± 0.259	0.002 ± 0.013
Heart	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	0.042 ± 0.002	0.034 ± 0.005	0.488 ± 0.468	0.003 ± 0.008
Ionosphere	0.984 ± 0.006	0.994 ± 0.006	1.000 ± 0.000	−0.002 ± 0.000	0.324 ± 0.042	0.004 ± 0.011	0.129 ± 0.094
Iris	0.953 ± 0.026	0.476 ± 0.237	-	0.941 ± 0.000	0.597 ± 0.074	0.573 ± 0.138	0.375 ± 0.283
Led7Digit	0.017 ± 0.006	0.008 ± 0.003	-	0.555 ± 0.028	0.534 ± 0.044	0.272 ± 0.057	-
Monk2	0.899 ± 0.017	0.991 ± 0.008	1.000 ± 0.000	0.883 ± 0.000	0.196 ± 0.104	0.199 ± 0.401	0.034 ± 0.025
Moons	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	0.774 ± 0.000	0.816 ± 0.095	1.000 ± 0.000	0.299 ± 0.161
Movement Libras	0.020 ± 0.009	0.003 ± 0.003	-	0.350 ± 0.024	0.292 ± 0.029	0.000 ± 0.000	-
Newthyroid	0.845 ± 0.016	0.719 ± 0.107	-	0.801 ± 0.001	0.445 ± 0.151	0.924 ± 0.138	0.000 ± 0.009
Saheart	0.867 ± 0.006	0.981 ± 0.006	1.000 ± 0.000	0.009 ± 0.001	0.030 ± 0.022	0.808 ± 0.384	0.003 ± 0.003
Sonar	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000	−0.004 ± 0.000	0.043 ± 0.042	0.000 ± 0.000	0.006 ± 0.007
Soybean	0.629 ± 0.057	0.592 ± 0.230	−0.609 ± 0.783	0.560 ± 0.000	0.641 ± 0.050	0.000 ± 0.000	0.123 ± 0.087
Spectfheart	1.000 ± 0.000	0.997 ± 0.006	1.000 ± 0.000	−0.004 ± 0.000	−0.123 ± 0.008	0.000 ± 0.000	0.032 ± 0.032
Spiral	1.000 ± 0.000	1.000 ± 0.000	0.400 ± 0.917	−0.003 ± 0.000	0.010 ± 0.006	0.601 ± 0.488	0.012 ± 0.006
Tae	0.846 ± 0.031	0.247 ± 0.128	-	0.021 ± 0.000	0.000 ± 0.006	0.051 ± 0.015	−0.000 ± 0.000
Vehicle	0.171 ± 0.049	0.043 ± 0.024	1.000 ± 0.000	0.070 ± 0.000	0.081 ± 0.000	0.089 ± 0.026	−0.006 ± 0.007
Wine	0.898 ± 0.024	0.383 ± 0.183	-	0.625 ± 0.000	0.369 ± 0.002	0.488 ± 0.165	0.011 ± 0.008
Zoo	0.250 ± 0.060	0.115 ± 0.073	0.751 ± 0.117	0.728 ± 0.069	0.455 ± 0.111	0.443 ± 0.131	-
Mean	**0.740**	0.641	0.102	0.332	0.284	0.390	−0.156

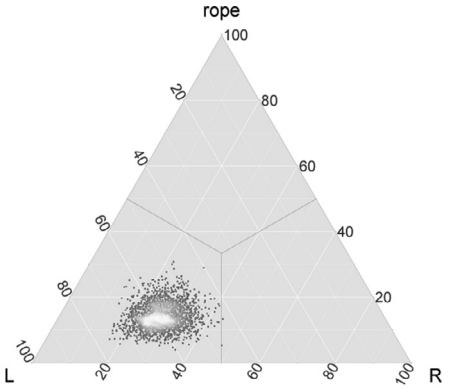
sets for each one—we can perform an empirical analysis. This way we can statistically determine whether $DILS_{CC}$ represents a significant improvement over previous proposals.

Keeping in mind the notation introduced in Section 5.3, we will refer to the results obtained by a given state-of-the-art method as sample *A*, and to the results obtained with $DILS_{CC}$ as sample *B*.

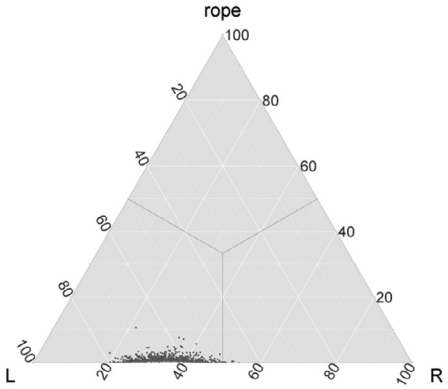
One of the major advantages of the Bayesian sign test is that we can obtain a very illustrative visual representation of its results. We can produce a representation of the triplet set in the form of a heatmap where each triplet constitutes one point whose location is given by barycentric coordinates. 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. As 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.



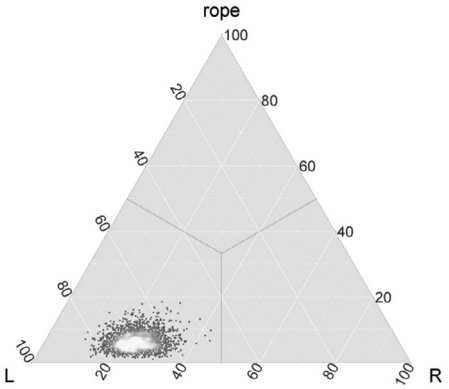
(a) BRKGA+LS



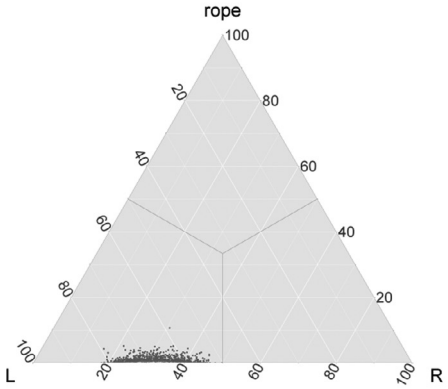
(b) COPKM



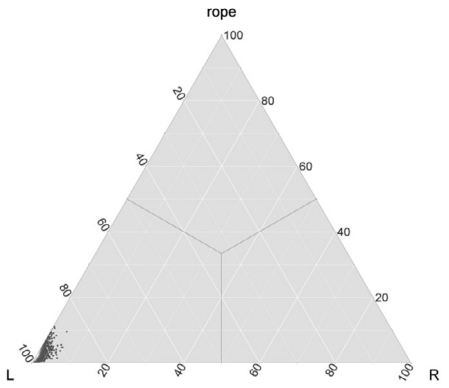
(c) LCVQE



(d) TVClust



(e) RDPM



(f) CECM

Fig. 2. Heatmaps comparing DILS with the state-of-the-art methods. With *L* being equivalent to *A* and *R* being equivalent to *B* (results obtained by $DILS_{CC}$).

Fig. 2 shows all six heatmaps obtained when applying the Bayesian sign test to $DILS_{CC}$ and the state-of-the-art methods. All heatmaps suggest that the test assigns a high probability to $A - B < 0$, since most triplets are presented in the lower left third of the triangle. Bearing this in mind, and considering that ARI is a measure to maximize, we can say that the Bayesian sign test provides strong evidence that $DILS_{CC}$ does represent a significant improvement as compared to all the other six methods, and the results offered by the latter are not equivalent to those offered by $DILS_{CC}$ and are lower in quality.

8. Discussion and future work

In this Section we present a discussion regarding the main discoveries carried out during the development of our study, as well as future lines of research that may be of interest for the reader. We have shown the suitability of the proposed method to address highly constrained problems, although in some cases DILS is not able to provide better results than those a random model would produce for certain datasets. Also note that no optimization procedure was used to tune any parameter, so it is possible that with a different parameter configuration better results could be obtained for these datasets.

Regarding future lines or research, we propose performing the previously mentioned optimization process for DILS parameters. This would lead to a better understanding of the influence over the DILS optimiation process. Even if the majority of the datasets used in our experiments are real-world datasets, it would be interesting to apply DILS to a real-world problem in which the amount of side information is probably limited. This would let us test the robustness of the proposed method in a less controlled environment, this is, out of the laboratory.

We also want to put clear that the experimental study presented in this paper can be extended in the future. Dozens of constrained clustering methods have been proposed over the years, among which we have chosen the most settled ones for our comparison. Future research lines may include other methods in different frameworks such as GC+PR+LS (de Oliveira et al., 2017), classic spectral learning approaches (Kamvar et al., 2003), density-based clustering (Ruiz et al., 2007), and genetic multi-objective optimization (Matake et al., 2007), among many others.

9. Conclusions

In this paper we have proposed the DILS heuristic method, along with its application to the SSL constrained clustering problem, which we called $DILS_{CC}$. Focusing on instance-level ML and CL constraints, $DILS_{CC}$ has proven that heuristic techniques constitute a competitive approach to constrained clustering, being able to scale the quality of the results in a way that is directly proportional to the number of constraints.

Supported by the Bayesian statistical tests, we were able to objectively prove that the $DILS_{CC}$ approach is significantly better than the state-of-the-art methods. $DILS_{CC}$ has proven to be better when considering the quality and availability of the solutions, especially in cases where large sets of constraints are analyzed.

Compliance with ethical standards

Declaration of competing interest

The authors declare that there is no conflict of interest.

CRediT authorship contribution statement

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

CRediT authorship contribution statement

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

Acknowledgements

Our work has been supported by the research project TIN2017-89517-P and PP2016.PRI.I.02.

Appendix A. Computational times

In this appendix we include theTables A.9–A.11, which display the computational times for the experimental results shown in Section 6. As we can expect, heuristic algorithms ($DILS_{CC}$ and BRKGA+LS) take more time to produce results than non-heuristic algorithms, which are able to produce results in a short time at the cost of a loss of quality. Also notice how both heuristic methods and part of the state-of-the-art methods are able to scale the number of constraints analyzed without a significant loss in time efficiency. This is because constraints are provided to the implementation of the fitness function in the form of a list, which makes it linear with respect to the number of constraints.

Table A.9
Computational times obtained for CS_{10} by $DILS_{CC}$ and the state-of-the-art methods.

Dataset	$DILS_{CC}$	BRKGA+LS	COPKM	LCVQE	RDPM	TVClust	CECM
Appendicitis	836.462 ± 3.433	970.740 ± 18.440	0.043 ± 0.002	0.002 ± 0.000	0.750 ± 0.241	1.413 ± 0.699	225.811 ± 16.605
Breast Cancer	8572.686 ± 19.675	11202.543 ± 151.995	15.900 ± 0.337	1.593 ± 0.012	42.017 ± 2.687	261.365 ± 359.739	1422.571 ± 101.571
Bupa	3222.815 ± 6.915	4112.423 ± 32.108	1.303 ± 0.006	1.615 ± 0.015	8.093 ± 1.191	4.951 ± 1.339	588.799 ± 83.518
Circles	2491.072 ± 4.877	3150.693 ± 64.218	0.424 ± 0.004	1.326 ± 0.008	5.456 ± 0.576	5.539 ± 3.769	330.378 ± 1.947
Ecoli	2655.791 ± 7.462	3437.415 ± 117.749	0.451 ± 0.037	0.322 ± 0.561	14.914 ± 4.216	21.499 ± 5.761	-
Glass	1657.830 ± 3.594	2023.889 ± 99.500	2.775 ± 1.559	0.739 ± 0.241	3.084 ± 0.773	9.249 ± 3.927	-
Haberman	2639.809 ± 8.646	3384.526 ± 11.339	0.367 ± 0.007	1.416 ± 0.006	10.457 ± 1.232	6.156 ± 2.035	638.829 ± 95.621
Hayesroth	1211.519 ± 0.950	1431.753 ± 63.275	0.093 ± 0.002	0.116 ± 0.189	2.340 ± 0.598	3.542 ± 1.005	273.295 ± 37.550
Heart	2586.032 ± 6.040	3175.445 ± 55.638	0.314 ± 0.002	0.032 ± 0.017	7.172 ± 1.408	3.501 ± 0.800	368.672 ± 77.120
Ionosphere	4392.813 ± 9.885	5406.449 ± 92.855	0.599 ± 0.028	1.834 ± 0.009	12.409 ± 3.341	99.158 ± 3.940	335.589 ± 42.539
Iris	1128.839 ± 4.124	1333.852 ± 54.492	0.088 ± 0.006	0.003 ± 0.000	1.033 ± 0.283	2.554 ± 1.188	243.140 ± 19.235
Led7Digit	4101.794 ± 14.003	5922.440 ± 109.094	1.201 ± 0.144	1.324 ± 0.832	25.612 ± 5.026	43.737 ± 19.834	-
Monk2	4255.216 ± 9.605	5515.127 ± 188.474	4.103 ± 0.052	1.376 ± 0.005	20.652 ± 3.277	4.937 ± 2.943	1296.628 ± 418.658
Moons	2483.187 ± 2.605	3129.539 ± 58.251	0.361 ± 0.003	0.009 ± 0.000	6.392 ± 1.182	4.395 ± 1.558	316.631 ± 20.471
Movement Libras	3403.667 ± 13.402	4388.207 ± 179.541	16.433 ± 31.591	0.433 ± 1.052	15.042 ± 3.621	4104.008 ± 7.381	-
Newthyroid	1709.559 ± 6.213	2044.073 ± 65.880	0.170 ± 0.004	0.007 ± 0.001	5.938 ± 1.460	3.301 ± 1.393	415.572 ∓ 77.652
Saheart	4929.838 ± 20.132	6462.592 ± 117.612	3.437 ± 0.033	2.926 ± 0.018	20.264 ± 1.827	14.887 ± 6.838	56120.113 ± 70.439
Sonar	2513.643 ± 1.995	2944.467 ± 59.983	0.207 ± 0.002	0.014 ± 0.000	3.911 ± 0.614	219.576 ± 1.650	340.242 ∓ 40.630
Soybean	367.298 ± 0.882	400.437 ± 8.622	0.046 ± 0.012	0.002 ± 0.000	0.146 ± 0.028	21.713 ± 1.546	653.286 ± 150.376
Spectfheart	3449.794 ± 10.031	4107.915 ± 140.424	0.322 ± 0.014	1.180 ± 0.004	7.695 ± 2.166	140.794 ± 1.207	41056.731 ± 135.823
Spiral	2490.608 ± 3.487	3178.641 ± 56.783	0.502 ± 0.003	0.048 ± 0.000	7.187 ± 2.350	4.590 ± 2.103	289.820 ± 33.303
Tae	1158.875 ± 0.804	1345.990 ± 30.893	0.084 ± 0.001	0.015 ± 0.000	1.410 ± 0.215	2.713 ± 0.878	41246.329 ± 132.452
Vehicle	10269.071 ± 21.384	14854.942 ± 548.902	21.060 ± 0.549	7.579 ± 0.034	44.661 ± 5.008	1034.223 ± 303.728	5142.601 ± 325.421
Wine	1486.731 ± 3.676	1730.424 ± 39.631	0.123 ± 0.003	0.005 ± 0.000	2.126 ± 0.497	3.112 ± 1.713	295.154 ± 22.435
Zoo	744.820 ± 2.582	811.080 ± 28.777	0.091 ± 0.014	0.063 ± 0.173	0.568 ± 0.117	1.810 ± 0.378	-
Mean	**0.641**	0.490	-0.070	0.225	0.233	0.258	-0.196

Table A.10
Computational times obtained for CS_{15} by $DILS_{CC}$ and the state-of-the-art methods.

Dataset	$DILS_{CC}$	BRKGA+LS	COPKM	LCVQE	RDPM	TVClust	CECM
Appendicitis	870.419 ± 1.519	939.865 ± 5.393	0.043 ± 0.001	0.003 ± 0.000	0.678 ± 0.083	1.359 ± 0.912	187.019 ± 14.924
Breast Cancer	9223.639 ± 14.159	10055.680 ± 199.189	46.269 ± 0.813	0.036 ± 0.000	38.293 ± 5.735	401.649 ± 767.407	2010.572 ± 171.597
Bupa	3482.537 ± 7.532	3759.465 ± 51.309	5.584 ± 0.075	3.629 ± 0.024	7.608 ± 1.001	5.789 ± 1.405	670.719 ± 75.621
Circles	2673.520 ± 4.024	2945.964 ± 18.957	3.311 ± 0.012	2.850 ± 0.019	6.345 ± 1.942	4.818 ± 1.735	298.994 ± 17.132
Ecoli	2890.102 ± 7.694	3123.048 ± 123.568	0.853 ± 0.009	2.259 ± 0.112	13.965 ± 7.504	32.028 ± 7.317	-
Glass	1766.649 ± 2.193	1877.678 ± 60.728	0.217 ± 0.016	1.321 ± 0.031	3.713 ± 1.521	9.225 ± 3.386	-
Haberman	2867.311 ± 6.534	3053.066 ± 56.844	2.060 ± 0.015	3.024 ± 0.012	10.845 ± 1.315	7.996 ± 1.714	875.831 ± 170.819
Hayesroth	1266.781 ± 2.358	1379.870 ± 31.659	0.101 ± 0.002	0.771 ± 0.232	2.038 ± 0.264	3.947 ± 1.914	285.862 ± 33.246
Heart	2776.324 ± 3.260	3033.346 ± 7.757	2.087 ± 0.010	2.317 ± 0.014	7.503 ± 2.985	4.646 ± 2.338	358.149 ± 74.820
Ionosphere	4688.942 ± 19.362	5115.442 ± 35.421	5.541 ± 0.026	3.954 ± 0.023	13.649 ± 4.267	100.865 ± 2.738	319.996 ± 61.974
Iris	1180.887 ± 2.598	1268.143 ± 19.266	0.085 ± 0.003	0.004 ± 0.000	1.029 ± 0.198	2.440 ± 0.793	270.320 ± 22.175
Led7Digit	4622.592 ± 10.484	5184.134 ± 12.602	0.910 ± 0.011	2.812 ± 0.055	26.301 ± 6.681	44.455 ± 11.260	-
Monk2	4643.210 ± 14.164	5029.269 ± 79.819	17.264 ± 0.140	2.512 ± 0.014	17.034 ± 3.838	5.386 ± 2.067	2012.146 ± 491.719
Moons	2679.188 ± 5.036	2956.292 ± 58.417	3.270 ± 0.016	1.303 ± 0.004	6.142 ± 1.000	2.993 ± 1.229	286.964 ± 29.530
Movement Libras	3724.620 ± 14.630	4124.618 ± 30.036	2.219 ± 1.722	3.825 ± 1.218	11.585 ± 2.164	4133.348 ± 11.891	-
Newthyroid	1820.379 ± 1.483	1952.237 ± 8.736	0.221 ± 0.003	0.075 ± 0.198	6.007 ± 2.143	3.899 ± 1.348	715.212 ∓ 120.123
Saheart	5437.260 ± 15.772	5937.302 ± 26.829	17.303 ± 0.282	6.491 ± 0.042	17.993 ± 7.372	10.861 ± 5.433	49653.622 ± 177.717
Sonar	2623.070 ± 10.864	2875.667 ± 14.249	0.643 ± 0.002	1.470 ± 0.010	3.669 ± 0.753	219.351 ± 0.936	570.987 ∓ 54.156
Soybean	375.322 ± 0.975	401.800 ± 8.244	0.046 ± 0.010	0.002 ± 0.000	0.149 ± 0.044	22.594 ± 1.159	900.286 ± 550.376
Spectfheart	3660.131 ± 12.829	3903.092 ± 60.601	8.006 ± 0.050	2.325 ± 0.019	5.512 ± 0.667	143.261 ± 0.490	41116.071 ± 9.599
Spiral	2681.662 ± 3.100	2953.111 ± 32.974	3.266 ± 0.016	2.869 ± 0.021	7.464 ± 2.636	5.006 ± 2.475	317.322 ± 22.482
Tae	1206.891 ± 2.128	1299.835 ± 21.167	0.085 ± 0.001	0.803 ± 0.004	1.631 ± 0.346	2.926 ± 1.055	41306.865 ± 2.798
Vehicle	11791.329 ± 16.295	12704.562 ± 219.254	120.891 ± 2.506	16.109 ± 0.078	45.575 ± 7.152	1825.403 ± 903.173	5652.986 ± 542.349
Wine	1564.367 ± 3.616	1658.676 ± 36.886	0.145 ± 0.002	0.772 ± 0.003	1.930 ± 0.196	3.859 ± 1.284	467.915 ± 48.057
Zoo	769.762 ± 1.269	799.395 ± 12.215	2.149 ± 1.338	0.007 ± 0.003	0.688 ± 0.236	2.464 ± 1.098	-
Mean	**0.641**	0.490	-0.070	0.225	0.233	0.258	-0.196

Table A.11
Computational times obtained for CS_{20} by $DILS_{CC}$ and the state-of-the-art methods.

Dataset	$DILS_{CC}$	BRKGA+LS	COPKM	LCVQE	RDPM	TVClust	CECM
Appendicitis	941.596 ± 24.602	970.740 ± 18.440	0.049 ± 0.002	0.682 ± 0.014	0.667 ± 0.085	1.952 ± 1.330	220.754 ± 21.526
Breast Cancer	10625.842 ± 32.150	11202.543 ± 151.995	78.992 ± 2.811	0.041 ± 0.001	37.674 ± 4.900	252.363 ± 314.979	2968.872 ± 240.198
Bupa	3934.482 ± 11.365	4112.423 ± 32.108	10.645 ± 0.239	5.953 ± 0.128	7.022 ± 0.836	5.902 ± 0.855	850.963 ± 133.290
Circles	3016.738 ± 82.722	3150.693 ± 64.218	6.834 ± 0.188	4.829 ± 0.115	7.053 ± 3.683	4.480 ± 1.497	297.416 ± 17.076
Ecoli	3376.047 ± 8.604	3437.415 ± 117.749	2.653 ± 0.058	3.011 ± 0.117	12.735 ± 3.458	42.049 ± 17.447	-
Glass	1956.745 ± 52.613	2023.889 ± 99.500	0.445 ± 0.009	2.021 ± 0.080	3.590 ± 0.612	14.198 ± 4.366	-
Haberman	3274.779 ± 6.785	3384.526 ± 11.339	4.542 ± 0.095	5.363 ± 0.113	7.733 ± 1.651	6.427 ± 1.495	1275.631 ± 219.198
Hayesroth	1390.373 ± 38.503	1431.753 ± 63.275	0.225 ± 0.005	1.196 ± 0.026	2.260 ± 0.493	7.155 ± 3.366	296.097 ± 20.053
Heart	3080.272 ± 10.436	3175.445 ± 55.638	4.116 ± 0.095	3.760 ± 0.077	5.933 ± 1.225	4.433 ± 1.555	798.212 ± 100.896
Ionosphere	5272.488 ± 23.794	5406.449 ± 92.855	11.025 ± 0.261	6.650 ± 0.133	12.436 ± 2.677	102.043 ± 3.655	312.728 ± 0.654
Iris	1302.971 ± 7.542	1333.852 ± 54.492	0.160 ± 0.004	0.005 ± 0.000	1.370 ± 0.308	2.825 ± 0.995	262.222 ± 23.205
Led7Digit	5550.213 ± 19.090	5922.440 ± 109.094	2.289 ± 0.049	4.317 ± 0.124	28.359 ± 8.789	58.753 ± 15.515	-
Monk2	5394.441 ± 27.467	5515.127 ± 188.474	28.226 ± 0.794	0.066 ± 0.001	21.581 ± 8.029	7.001 ± 2.436	6098.149 ± 1207.096
Moons	3003.966 ± 79.507	3129.539 ± 58.251	6.434 ± 0.165	0.016 ± 0.000	7.065 ± 1.328	2.957 ± 1.261	309.228 ± 16.421
Movement Libras	4146.830 ± 108.300	4388.207 ± 179.541	0.844 ± 0.037	4.941 ± 0.090	13.562 ± 2.248	4133.322 ± 11.153	-
Newthyroid	2074.650 ± 16.282	2044.073 ± 65.880	0.266 ± 0.004	0.894 ± 0.018	5.899 ± 3.721	3.754 ± 1.386	1858.098 ∓ 360.190
Saheart	6176.552 ± 166.034	6462.592 ± 117.612	31.765 ± 1.024	11.126 ± 0.212	20.136 ± 2.959	11.825 ± 6.662	48548.465 ± 9.947
Sonar	2848.079 ± 10.806	2944.467 ± 59.983	1.585 ± 0.035	2.448 ± 0.053	3.578 ± 1.275	217.391 ± 1.589	1570.098 ∓ 301.090
Soybean	402.416 ± 4.042	400.437 ± 8.622	0.021 ± 0.008	0.003 ± 0.001	0.141 ± 0.024	22.588 ± 1.252	1500.884 ± 518.905
Spectfheart	3978.011 ± 97.213	4107.915 ± 140.424	8.845 ± 0.230	3.843 ± 0.072	5.967 ± 1.661	144.650 ± 1.265	40879.419 ± 2.494
Spiral	3027.213 ± 82.836	3178.641 ± 56.783	6.067 ± 0.153	4.884 ± 0.112	6.653 ± 1.999	4.442 ± 2.408	287.252 ± 30.888
Tae	1321.552 ± 35.344	1345.990 ± 30.893	0.175 ± 0.003	1.386 ± 0.028	1.294 ± 0.356	3.458 ± 1.314	40245.960 ± 7853.474
Vehicle	14368.552 ± 33.154	14854.942 ± 548.902	204.742 ± 9.065	27.449 ± 0.582	47.316 ± 7.189	1325.217 ± 516.388	16698.002 ± 1982.918
Wine	1724.394 ± 8.122	1730.424 ± 39.631	0.348 ± 0.009	0.017 ± 0.000	1.965 ± 0.430	6.102 ± 3.111	441.470 ± 32.803
Zoo	843.426 ± 2.294	811.080 ± 28.777	0.109 ± 0.028	0.007 ± 0.002	0.656 ± 0.189	2.705 ± 0.729	-
Mean	**0.641**	0.490	-0.070	0.225	0.233	0.258	-0.196

References

Antoine, V., Quost, B., Masson, M.-H., Denoeux, T., 2012. CECM: constrained evidential c-means algorithm. Comput. Stat. Data Anal. 56 (4), 894–914.

Archetti, C., Feillet, D., Mor, A., Speranza, M.G., 2018. An iterated local search for the traveling salesman problem with release dates and completion time minimization. Comput. Oper. Res. 98, 24–37.

Avci, M., Topaloglu, S., 2017. A multi-start iterated local search algorithm for the generalized quadratic multiple knapsack problem. Comput. Oper. Res. 83, 54–65.

Ayed, S.B., Elouedi, Z., Lefevre, E., 2019. CEVM: Constrained evidential vocabulary maintenance policy for CBR systems. In: International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems. Springer, pp. 579–592.

Bean, J.C., 1994. Genetic algorithms and random keys for sequencing and optimization. ORSA J. Comput. 6 (2), 154–160.

Benavoli, A., Corani, G., Demšar, J., Zaffalon, M., 2017. Time for a change: a tutorial for comparing multiple classifiers through bayesian analysis. J. Mach. Learn. Res. 18 (1), 2653–2688.

Bradley, P.S., Bennett, K.P., Demiriz, A., 2000. Constrained K-Means Clustering. Technical Report. MSR-TR-2000-65, Microsoft Research.

Brieden, A., Gritzmann, P., Klemm, F., 2017. Constrained clustering via diagrams: a unified theory and its application to electoral district design. Eur. J. Oper. Res. 263 (1), 18–34.

Carrasco, J., García, S., del Mar Rueda, M., Herrera, F., 2017. rNPBST: an R package covering non-parametric and bayesian statistical tests. In: International Conference on Hybrid Artificial Intelligence Systems. Springer, pp. 281–292.

Chapelle, O., Schlkopf, B., Zien, A., 2010. Semi-Supervised Learning, first ed. The MIT Press.

Chaves, A.A., Gonçalves, J.F., Lorena, L.A.N., 2018. Adaptive biased random-key genetic algorithm with local search for the capacitated centered clustering problem. Comput. Ind. Eng. 124, 331–346.

Chentli, H., Ouafi, R., Cherif-Khettaf, W.R., 2018. Impact of iterated local search heuristic hybridization on vehicle routing problems: application to the capacitated profitable tour problem. In: International Conference on Operations Research and Enterprise Systems. Springer, pp. 80–101.

Davidson, I., Basu, S., 2007. A survey of clustering with instance level constraints. ACM Trans. Knowl. Discov. Data 1, 1–41.

Davidson, I., Ravi, S., 2005. Clustering with constraints: feasibility issues and the k-means algorithm. In: Proceedings of the 2005 International Conference on Data Mining. SIAM, pp. 138–149.

Estrada-Moreno, A., Savelsbergh, M., Juan, A.A., Panadero, J., 2019. Biased-randomized iterated local search for a multiperiod vehicle routing problem with price discounts for delivery flexibility. Int. Trans. Oper. Res. 26 (4), 1293–1314.

Gendreau, M., Potvin, J.-Y., 2010. Handbook of Metaheuristics, second ed. Springer Publishing Company, Incorporated.

Gonçalves, J.F., Resende, M.G., 2011. Biased random-key genetic algorithms for combinatorial optimization. J. Heuristics 17 (5), 487–525.

Hruschka, E.R., Campello, R.J., Freitas, A.A., et al., 2009. A survey of evolutionary algorithms for clustering. IEEE Trans. Syst. Man Cybern.Part C 39 (2), 133–155.

Hubert, L., Arabie, P., 1985. Comparing partitions. J. Classif. 2 (1), 193–218.

Jain, A.K., Murty, M.N., Flynn, P.J., 1999. Data clustering: a review. ACM Comput. Surv. (CSUR) 31 (3), 264–323.

José-García, A., Gómez-Flores, W., 2016. Automatic clustering using nature-inspired metaheuristics: a survey. Appl. Soft Comput. 41, 192–213.

Kamvar, K., Sepandar, S., Klein, K., Dan, D., Manning, M., Christopher, C., 2003. Spectral learning. In: International Joint Conference of Artificial Intelligence. Stanford InfoLab.

Khashabi, D., Wieting, J., Liu, J. Y., Liang, F., 2015. Clustering with side information: from a probabilistic model to a deterministic algorithm. arXiv:1508.06235.

Law, M.H., Topchy, A., Jain, A.K., 2004. 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, pp. 662–670.

Lourenço, H.R., Martin, O.C., Stützle, T., 2010. Iterated local search: framework and applications. In: Handbook of Metaheuristics. Springer, pp. 363–397.

Mai, S.T., Amer-Yahia, S., Bailly, S., Pépin, J.-L., Chouakria, A.D., Nguyen, K.T., Nguyen, A.-D., 2018. Evolutionary active constrained clustering for obstructive sleep apnea analysis. Data Sci. Eng. 3 (4), 359–378.

Masson, M.-H., Denoeux, T., 2008. ECM: an evidential version of the fuzzy c-means algorithm. Pattern Recognit. 41 (4), 1384–1397.

Matake, N., Hiroyasu, T., Miki, M., Senda, T., 2007. Multiobjective clustering with automatic k-determination for large-scale data. In: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, pp. 861–868.

Nanda, S.J., Panda, G., 2014. A survey on nature inspired metaheuristic algorithms for partitional clustering. Swarm Evol. Comput. 16, 1–18.

de Oliveira, R.M., Chaves, A.A., Lorena, L.A.N., 2017. A comparison of two hybrid methods for constrained clustering problems. Appl. Soft Comput. 54, 256–266.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., 2011. Scikit-learn: machine learning in Python. J. Mach. Learn. Res. 12, 2825–2830.

Pelleg, D., Baras, D., 2007. K-means with large and noisy constraint sets. In: European Conference on Machine Learning. Springer, pp. 674–682.

Rand, W.M., 1971. Objective criteria for the evaluation of clustering methods. J. Am. Stat. Assoc. 66 (336), 846–850.

Ruiz, C., Spiliopoulou, M., Menasalvas, E., 2007. C-DBSCAN: density-based clustering with constraints. In: International Workshop on Rough Sets, Fuzzy Sets, Data Mining, and Granular-Soft Computing. Springer, pp. 216–223.

Saidi, F., Trabelsi, Z., Ghazela, H.B., 2018. A novel approach for terrorist sub-communities detection based on constrained evidential clustering. In: 2018 12th International Conference on Research Challenges in Information Science (RCIS). IEEE, pp. 1–8.

Schmidt, J., Brandle, E.M., Kramer, S., 2011. Clustering with attribute-level constraints. In: 2011 IEEE 11th International Conference on Data Mining. IEEE, pp. 1206–1211.

Semnani, S.H., Basir, O.A., Van Beek, P., 2016. Constrained clustering for flocking-based tracking in maneuvering target environment. Rob. Auton. Syst. 83, 243–250.

Seret, A., Verbraken, T., Baesens, B., 2014. A new knowledge-based constrained clustering approach: theory and application in direct marketing. Appl. Soft Comput. 24, 316–327.

Spears, W.M., De Jong, K.D., 1995. On the virtues of parameterized uniform crossover. Technical Report. NAVAL RESEARCH LAB WASHINGTON DC.

Triguero, I., García, S., Herrera, F., 2015. Self-labeled techniques for semi-supervised learning: taxonomy, software and empirical study. Knowl. Inf. Syst. 42 (2), 245–284.

Triguero, I., González, S., Moyano, J.M., García, S., Alcalá-Fdez, J., Luengo, J., Fernández, A., del Jesús, M.J., Sánchez, L., Herrera, F., 2017. KEEL 3.0: an open source software for multi-stage analysis in data mining. Int. J. Comput. Intell.Syst. 10 (1), 1238–1249.

Wagstaff, K., Cardie, C., Rogers, S., Schrödl, S., 2001. Constrained k-means clustering with background knowledge. In: Proceedings of the Eighteenth International Conference on Machine Learning. Morgan Kaufmann Publishers Inc., pp. 577–584.

Wang, J., Wu, S., Li, G., 2010. Clustering with instance and attribute level side information. Int. J. Comput. Intell.Syst. 3 (6), 770–785.

Wu, H., Prasad, S., 2017. Semi-supervised deep learning using pseudo labels for hyperspectral image classification. IEEE Trans. Image Process. 27 (3), 1259–1270.

Zhao, F., He, X., Yang, G., Ma, W., Zhang, C., Song, H., 2019. A hybrid iterated local search algorithm with adaptive perturbation mechanism by success-history based parameter adaptation for differential evolution (shade). Eng. Optim. 1–17.

Zohali, H., Naderi, B., Mohammadi, M., Roshanaei, V., 2019. Reformulation, linearization, and a hybrid iterated local search algorithm for economic lot-sizing and sequencing in hybrid flow shop problems. Comput. Oper. Res. 104, 127–138.