A Self-Adaptive Cuckoo Search Algorithm Using a Machine Learning Technique

Nicolás Caselli 1,*, Ricardo Soto 1, Broderick Crawford 1, Sergio Valdivia 2 and Rodrigo Olivares 3, *

Abstract: Metaheuristics are intelligent problem-solvers that have been very efficient in solving huge optimization problems for more than two decades. However, the main drawback of these solvers is the need for problem-dependent and complex parameter setting in order to reach good results. This paper presents a new cuckoo search algorithm able to self-adapt its configuration, particularly its population and the abandon probability. The self-tuning process is governed by using machine learning, where cluster analysis is employed to autonomously and properly compute the number of agents needed at each step of the solving process. The goal is to efficiently explore the space of possible solutions while alleviating human effort in parameter configuration. We illustrate interesting experimental results on the well-known set covering problem, where the proposed approach is able to compete against various state-of-the-art algorithms, achieving better results in one single run versus 20 different configurations. In addition, the result obtained is compared with similar hybrid bio-inspired algorithms illustrating interesting results for this proposal.

Keywords: clustering techniques; metaheuristics; machine learning; self-adaptive, parameter setting; exploration; exploitation

1. Introduction

Recent studies about bio-inspired procedures to solve complex optimization problems have demonstrated that finding good results and the best performance are laborious tasks, so it is necessary to apply an off-line parameter adjustment on metaheuristics [1–5]. This adjustment is considered an optimization problem itself, and several studies are proposing some solutions to solve that, but it always depends on his static terms [6]. Many of these studies use mathematical ways to change the values of one of each parameter during their execution. In this context, parameters like the population size of metaheuristics are initially set without considering their variation or behaviors. We consider using a machine learning (ML) technique that lets us analyze the population and determine the values of the number of solutions and the abandon probability.

As we can see in [7], machine learning and optimization are two topics of artificial intelligence that are rapidly expanding, having a wide range of computer science applications. Due to the rapid progress in the performance of computing and communication techniques, these two research areas have proliferated and drawn widespread attention in a wide variety of applications [8]. For example, in [9], the authors present different clustering techniques to evaluate the credit risk in a determinate population of Europe. Although both fields belong to different communities, they are fundamentally based on artificial intelligence, and the techniques from ML and optimization frequently interact with each other and themselves to improve their learning and/or search capabilities.
On the other hand, advances in operations research and computer science have brought forward new solution approaches in optimization theory, such as heuristics and metaheuristics. While the former are experience-based procedures, which usually provide good solutions in short computing times, metaheuristics are general templates that can easily be tailored to address a wide range of problems. They have been shown to provide near-optimal solutions in reasonable computing times to problems for which traditional methods are not applicable [10]. Moreover, as we can see in [11], the tendency to use a hybrid method to solve some type of recent problem, such as those that COVID-19 has brought with it, has recently proven its effectiveness. In one of the important studies of these two main topics, we are interested in exploring the integration to use ML into metaheuristics, in terms to enhance any of the characteristics or attributes of those algorithms: the solutions, performance, or time to get the results. In this way, we propose to use an unsupervised machine learning technique that let us learning in the search space of the metaheuristic, exploiting the characteristic of their attributes that let us use to enhance the metaheuristic parameters in an online way: spatial clustering based on noise application density (DBSCAN) is one of those techniques that gathers those characteristics. We propose to use noise clustering to associate with the abandon probability and the solutions clustering result for determining the number of nests of the cuckoo search algorithm (CSA). They are studies that include some hybrid propose to enhance the CSA whit an ML technique, as in [12], when the authors present a Kmean technique to determinate the discrete parameters to improve the CSA. In [13], the contribution is a hybrid method with a sin-cousin algorithm to enhance the search space to be used on the CSA. Other different scenarios, but similar cases are in [14], they present a CuckooVina, a combination of cuckoo search and differential evolution in a conformational search method. Thus, some studies propose a hybridization to enhance CSA, but they do not respond to our search to find some ML technique that allows enriching the self-adaptation capabilities of CSA in the way we present.

To illustrate and prove our approach, we apply an improvement of the cuckoo search algorithm with a self-adaptive capability (SACSADBSCAN). This approach was tested on the set covering problem, whose goal is to cover a range of needs at the lowest cost and is a widely used problem to demonstrate research.

The remainder of the paper is structured as follows: in Section 2, we introduce the theoretical background on the optimization and clustering algorithms that have been integrated into the proposed approach, then, in Section 3, we present the original CSA, and his parameters that need to be set to run, moreover we introduce to the DBSCAN algorithm, how is the characteristic operation of this algorithm, and how we propose to exploit them in the metaheuristics. Following this, in Section 4 we present the integration of DBSCAN on CSA, how parameter values are configured in the execution of our approach. Then, in Section 5 experimental results and discussions are shown. Finally, we conclude and suggest some lines of future research in Section 6.

2. Related Work

As previously mentioned, parameter adjustment of metaheuristics is a complex task and is considered an optimization problem itself, in many cases, this depends on the try-error test to find a good combination of parameters. In this scenario, some studies use different ways, many of those with a mathematical formula to vary the values of the parameters of the CSA.

For example, to determine the step size $\alpha$ and $Pa$ parameter of the Cuckoo Search, Ref. [15] uses a mathematical formula that depends on the range of those minimal and maximal values of those parameters and the number of iterations of the algorithm. With that idea, the target is to use a bigger target area, also those neighboring areas. In [16], authors propose a self-adaptive step size in his studies to vary the $\alpha$ parameter according to the fitness in each iteration, applying a mathematical formula to get the value to set. In addition, Ref. [17] vary the $\alpha$ and $Pa$ values according to their propose, $\alpha$ changes his value according
to the algorithm iterations, meanwhile $Pa$ vary randomly depending on the dimension map of the problem. Similar studies are in [18] where Q-Learning is used to set the step size value of the cuckoo search algorithm in a self-adaptive way.

Another study case of adaptive CSA is [19], where the author varies the step size and the abandon probability considering the historical values of them, including their best and worst values to determine the new one. Following with the vary in population, in [20] the authors divide the cuckoo population between making an analysis to them and improve the results.

In another case, such as in [21] use a hybrid algorithm to set the values of their parameters and reduce the population, when the diversity of population decreases, the population is reduced, allowing the algorithm to diversify in subsequent iterations.

As we can see, the machine learning techniques that we have cited up to this point are used to determine a static value in the metaheuristic parameters, which is used as it is the initial value. However, it is used to vary the value of the metaheuristic parameters. During its execution, it is seen minimally in the literature.

In other terms, machine learning has also been applied to the metaheuristic to compact the heuristic space through a clustering procedure [22] and artificial networks [23]. These manuscripts use forecasting and classification. In deep, machine learning integrates with metaheuristics is proposed to forecast different classes of problems, such as electric load [24], economic recessions [25], optimise the low-carbon flexible job shop scheduling problem [26] and other industrial problems [27]. As we mentioned, several dataset groups can be classified by employing machine learning. In this context, we can found image classification problem [28], electronic noise classification of bacterial food bones pathogens [29], urban management [30] and to mention some recently studies. Finally, in [31] the authors use the DBSCAN algorithm to make a binarization strategy and transform the continuous search space to a binary one.

Although machine learning has a big presence in the metaheuristic discipline, its use to set values is quite limited. The use to vary, in an online way, the value of the parameters of the metaheuristic is an area to dig deeper. In this way, we believe that the self-adaptive with DBSCAN, beyond the enhancement of metaheuristics, there seems to be a lot of scope for exploration, using a machine learning technique like DBSCAN with his characteristic we can make an analysis, and thanks to that, change the values of the parameters, is the aim pursued in this work.

3. Theoretical Background

3.1. Cuckoo Search Algorithm

Metaheuristics belong to a class of approximation methods. These are types of higher-level general-purpose algorithms that can be used for a wide range of different types of problems. They have been widely used to solve large-scale combinatorial optimization problems within acceptable computational time, but they do not guarantee optimum solutions [32].

There are a lot of metaheuristics. A full set of all nature-inspired algorithms can be found in [33], and one of them is CSA, which has several study cases. CSA [34] is inspired by the obligate brood parasitism of some cuckoo species by laying their eggs in the nests of other bird species. The steps of CSA are described below:

1. Each cuckoo lays an egg at a time and drops it into a randomly selected nest.
2. The best nests with high-quality eggs will be carried over to the next generations.
3. The number of available host nests is fixed, and the egg laid by a cuckoo is discovered by the host bird with a probability $Pa \in [0, 1]$. In this case, a new random solution is generated.
Every new generation is determinate by Lévy flight [34], that is given by the Equation (2)

\[
x^d_i(t+1) = x^d_i + \alpha \otimes \text{Levy}(\beta)
\]
\[\forall i \in \{1, ..., n\} \vee \forall d \in \{1, ..., m\}\] (1)

where \(x^d_i\) is the element \(d\) of a solution \(i\) at iteration \(t\). \(x^d_i(t+1)\) is a solution in the iteration \(t+1\). \(\alpha > 0\) is the step size which should be related to the scales of the problem of interest, the upper (\(U_b\)) and lower bounds (\(L_b\)) that the problem need to be determinated, in this scenario values between 0 and 1.

\[
\text{Levy} \sim u = t^\beta, (0 < \beta < 3)
\] (2)

The Lévy flight represents a random walk while the random step length is drawn from a Lévy distribution which has an infinite variance with an infinite mean.

3.2. Density Based Spatial Clustering Application with Noise

Density-Based Spatial Clustering of Applications with Noise [35] is a popular data grouping algorithm that uses automated analysis techniques to similar group information together. It can be used to find clusters of any shape in a data collection with noise and outliers. Clusters are dense sections of data space separated by regions with a lower density of points.

The goal is to define dense areas that can be determined by the number of things in close proximity to a place. It is crucial to understand that DBSCAN has two important parameters that are required for work.

1. Epsilon (\(\epsilon\)): Determines how close points in a cluster can be seen to each other.
2. Minimum points (\(\text{MinPts}\)): The minimal amount of points required to produce a concentrated area.

The underlying premise is that there must be at least a certain number of points in the vicinity of a particular radius for each given group. The \(\epsilon\) parameter determines the surrounding radius around a given point; every point \(x\) in the data set is tagged as a central point with a neighbor above or equal to \(\text{MinPts}\); \(x\) is a limit point if the number of neighbors is less than \(\text{MinPts}\). Finally, if a point is neither a center nor a border, it is referred to as a point of noise or an outlying point.

When users do not know much about the data being modified, this technique proves helpful in metaheuristics. See pseudo-code in Algorithms 1 and 2 to understand how this strategy works.

The computed groups are shown on the right side of Figure 1. Each iteration of the process adjusts the positions until the algorithm converges. It is important to mention that the noise points are those that do not remain in any cluster at the time of iterating the algorithm and we can visualize them as the white points around the clusters identified in the graph (Red, Yellow, Blue and Green), all these noise points are grouped into a single cluster, and in this way all the points will belong to a specific cluster.
Algorithm 1: Cuckoo Search pseudo-code.

1. Input: \(\alpha, \beta, n, Pa, T\);
2. Output: a set of solutions;
3. //Produce the first generation of \(n\) nests;
4. foreach nest \(n_i, (\forall i = 1, \ldots, n)\) do
5.   foreach dimension \(d, (\forall d = 1, \ldots, m)\) do
6.     \(x_d^i \leftarrow \text{Random}\{0, 1\}\);
7.   end
8.   \(f_i \leftarrow z(x_i)\);
9. end
10. \(global\ f\ it \leftarrow +\infty\);
11. //Produce \(T\)-generations of \(n\) nests;
12. while \(t < T\) do
13.   //Evaluate the fitness and update the best solution;
14.   \{\(\text{min fit, minindex}\) \(\leftarrow \text{min}\{f\}\); \}
15.   if \(\text{min fit} < \text{global fit}\) then
16.     \(\text{global fit} \leftarrow \text{min fit}\);
17.     \(x_d(t) \leftarrow x_{\text{minindex}}^d(t)\);
18. end
19. //the bird abandon the nest with probability \(Pa\);
20. foreach nest \(n_i, (\forall i = 1, \ldots, n)\) do
21.   foreach dimension \(d, (\forall d = 1, \ldots, m)\) do
22.     if \(\text{rand} > Pa\) then
23.       \{ Select the worst nest according to \(Pa[0, 1]\) and replace them for new random solutions \};
24.     end
25. end
26. end
27. //the bird fly to another new nest with Lévy flight;
28. foreach nest \(n_i, (\forall i = 1, \ldots, n)\) do
29.   foreach dimension \(d, (\forall d = 1, \ldots, m)\) do
30.     if \(\text{rand} > Pa\) then
31.       \{ Generate new solutions through Equations (1) and 2\};
32.       \(x_d(t + 1) \leftarrow x_d(t) + a \odot \text{Lévy}(\beta)\);
33.       //In previous step, the value generated belongs to the real domain an it must be brought to a binary domain;
34.       \(x_d(t + 1) \leftarrow T(x_d^i(t + 1))\);
35.     end
36. end
37. end
38. Postprocess results and visualizations;
Algorithm 2: DBSCAN pseudo-code.

1. Input: dataset of \( P = \{p_1, ..., p_n\} \);
2. Input: \( \epsilon \);
3. Input: \( \text{MinPts} \);
4. Output: a set of \( k \) clusters;
5. \( \text{foreach } \text{unvisited point in } P \text{ do} \)
6. \( \quad \) determine the state as visited;
7. \( \quad \) neighborPts = getNeighbors(point, \( \epsilon \));
8. \( \quad \) \( C = 0 \);
9. \( \quad \) if \( \text{count} \text{neighborPts} \lt \text{MinPts} \) then
   10. \( \quad \quad \) set the state as noise;
   11. \( \quad \) else
   12. \( \quad \quad \) \( C = \) next cluster;
   13. \( \quad \quad \) expandCluster(point, neighborPts, \( C \), \( \epsilon \), \( \text{MinPts} \))
   14. \( \quad \) end
15. \( \) end
16. \( \text{expandCluster} \text{point, neighborPts, } C, \epsilon, \text{MinPts});
17. \( \text{add point to cluster } C; \)
18. \( \text{foreach } \text{pnt in neighborPts do} \)
19. \( \quad \) if \( \text{visitedPoints} \text{pnt}) \) then
20. \( \quad \quad \) add pnt to visited ones;
21. \( \quad \quad \) newNeighborPts = getNeighbors(pnt, \( \epsilon \));
22. \( \quad \quad \) if \( \text{sizeof} \text{neighborPts} \geq \text{MinPts} \) then
23. \( \quad \quad \quad \) \( \text{neighborPts} = \text{neighborPts} \cup \text{newNeighborPts} \)
24. \( \quad \quad \) end
25. \( \quad \) end
26. \( \quad \) if pnt is not yet member of any cluster then
27. \( \quad \quad \) add pnt to cluster \( C \)
28. \( \) end
29. \( \text{getNeighbors} \text{point, } \epsilon); \)
30. \( \text{return points within pnt } \epsilon\text{-neighborhood} \)

Figure 1. Application of the DBSCAN algorithm to the solution space as an example of clustering.
4. Proposed Approach: Integrating DBSCAN in CSA

In this section, we describe how DBSCAN was integrated on CSA, in addition to all the keys elements for this technique to work. We add a shortcode at the end of each iteration to decide whether intervention is appropriate to implement parameter control intervention and use the DBSCAN algorithm to calculate the values of those parameters (see Algorithm 3).

### Algorithm 3: Integration CSA with DBSCAN.

```plaintext
1 Input: α, β, n, Pa, T;
2 Output: a set of solutions;
3 //Produce the first generation of n nests;
4 foreach nest n_i (∀i = 1, . . . , n) do
5   foreach dimension d (∀d = 1, . . . , m) do
6       x^d_i ← Random{0, 1};
7   end
8   f_i ← z(x_i);
9 end
10 global fit ← +∞;
11 //Produce T-generations of n nests;
12 while t < T do
13   //the bird abandon the nest with probability Pa;
14   foreach nest n_i (∀i = 1, . . . , n) do
15     foreach dimension d (∀d = 1, . . . , m) do
16       if rand > Pa then
17         { Select the worst nest according to Pa[0, 1] and replace them for new random solutions };
18     end
19   end
20 //the bird fly to another new nest with Lévy flight;
21 foreach nest n_i (∀i = 1, . . . , n) do
22   foreach dimension d (∀d = 1, . . . , m) do
23     if rand > Pa then
24       { Generate new solutions through Equations (1) and 2};
25       x^d_i(t + 1) ← x^d_i(t) + α ⊙ Lévy(β);
26       //In previous step, the value generated belongs to the real domain an it must be brought to a binary domain;
27       x^d_i(t + 1) ← T(x^d_i(t + 1));
28     end
29   end
30 //Evaluate the fitness and update the best solution;
31 {min fit, minindex} ← min(f);
32 if min fit < global fit then
33   global fit ← min fit;
34   x̂^d(t) ← x^d_{minindex}(t);
35 end
36 //Integration of DBSCAN parameter tuning;
37 if LibertyParameter % t == 0 and t > 1 then
38   { Run DBSCAN with nest solutions and evaluate results to set parameter settings};
39 end
40 Post-process results and visualizations;
```

In the following sections, we explain some relevant topics that are important to mention to understand our approach. Section 4.1 explains under which criteria the DBSCAN intervening in the metaheuristic to make an analysis on the search space and make the
clustering on the solutions. Then, Section 4.2 indicates how the noise cluster is fundamental to determinate the abandon probabilities.

4.1. Free Execution Parameter

We decide to include a variable to control the moment to intervene the parameter values of the CSA to make the metaheuristic maintain its independence of executions and its specific behavior, in this case, we consider it prudent to set it on one hundred iterations of a free run. If these execution values are reached, then the algorithm performs a procedure to update the CSA parameter values, as is described in line 39 of the Algorithm 3.

4.2. Online Parameter Setting

As we mention in the previous section, the update of the values of the parameters in the CSA occurs when the value of the free execution parameter has reached its limit, then the DBSCAN algorithm can be run and used the generated clusters to infer in the parameter values of the metaheuristic. How to associate the parameters of the $Pa$ and $Nest$ is detailed in the following sections.

4.2.1. Probability Abandon Nest

To set the value of the probability of nest abandon, we use the value of the noise point obtained from the DBSCAN execution. That value indicates the points that are excluded from any cluster, so we can associate the number of noise points to make the metaheuristic consider those probabilities to explore new points of the search space. To make this possible, we associate the percentage number of noise points to set the abandon probabilities value. To let the metaheuristic keep the normal execution we use bounds between 10 to 40 percentage. That indicates, that if the noise points are more than 40% then, the $Pa$ values are set to 0.4, in the same way, if the noise point is less than 10%, the $Pa$ values are set to 0.1. In other cases, the values are set to the percentage values of noise points.

4.2.2. Number of Nests

To determine when varying the number of nests on the CSA, our approach considers keeping in memory the last best fitness value to compare with the new candidate solutions, if the best value of fitness does not vary on the fourth intervention of the DBSCAN, we consider that it is necessary to increase the number of nests, to amplify the search space. So, in this case, the number of nests increases in five on five every time this scenario occurs. On the other hand, if the global best value improves four times consecutively, then the number of nest decrease, eliminating the worst five.

4.3. Exploration Influence

During the tests, we realize that the CSA solution space converges to a single large cluster that considers all possible solutions while evaluating the executions, in some instances, certain clusters appear to have values compressed very close to each other. We deem it entirely appropriate in this scenario to have CSA explore the search space. For this, half of the cluster points are renovated to new random ones, allowing the metaheuristic to diversify the search space. The replacement criteria are according to the following Formula (3)

$$ExpInfl = \frac{\sum_{ClusterSol=1}^{n} Fitness_{ClusterSol}}{n}$$

where $ClusterSol$ corresponding to all solutions in the current cluster evaluates. $ExpInfl$ is evaluated with the best possible solution of the global population, if the absolute value of the difference between both is larger than one, then we renew a half-point on those clusters.
4.4. Set Covering Problem

Many studies use the Set Covering Problem (SCP) to represent real scenarios into the studies, as we can see in the area of management of crew on airlines [36], in the optimization of the location of the emergency buildings [37], manufacturing [38] and production [39]. As we can see, SCP is a classical combinatorial optimization problem. It is belong to the NP-hard class [40] and is formally defined as well: let \( A = (a_{ij}) \) be a binary matrix with \( M \)-rows (\( \forall i \in I = \{1, \ldots, M\} \)) and \( N \)-columns (\( \forall j \in J = \{1, \ldots, N\} \)), and let \( C = (c_j) \) be a vector representing the cost of each column \( j \), assuming that \( c_j > 0 \), \( \forall j = \{1, \ldots, N\} \). Then, it observes that a column \( j \) covers a row \( i \) if \( a_{ij} = 1 \). Therefore, it has:

\[
a_{ij} = \begin{cases} 
1, & \text{if row } i \text{ can be covered by column } j \\
0, & \text{otherwise}
\end{cases}
\]

The SCP entails identifying a group of materials that can be used to address a lot of purposes for the least amount of money. A feasible solution corresponds to a subset of columns in its matrix form, and the demands are associated with rows and regarded as constraints. The goal of the challenge is to find the columns that best cover all of the rows.

The Set Covering Problem identifies a low-cost subset \( S \) of columns that covers each row with at least one column from \( S \). The SCP can be expressed using integer programming as follows:

\[
\begin{align*}
\text{minimize} & \quad \sum_{j=1}^{n} c_j x_j \\
\text{subject to:} & \quad \sum_{j=1}^{n} a_{ij} x_j \geq 1 \quad \forall i \in I \\
& \quad x_j \in \{0, 1\} \quad \forall j \in J
\end{align*}
\]

Instances: We use 65 instances from Beasley’s OR-library, which are arranged into 11 sets, to evaluate the algorithm’s performance when solving the SCP. To represent the instances, we present on Table 1 the following details: instance group, the number of rows \( M \), number of columns \( N \), the cost range, density (percentage of non-zeroes in the matrix).

Table 1. Instances taken from the Beasley’s OR-Library.

| Instance Group | \( M \) | \( N \) | Cost Range | Density (%) | Best Known |
|----------------|--------|--------|------------|-------------|-------------|
| 4              | 200    | 1000   | [1, 100]   | 2           | Known       |
| 5              | 200    | 2000   | [1, 100]   | 2           | Known       |
| 6              | 200    | 1000   | [1, 100]   | 5           | Known       |
| A              | 300    | 3000   | [1, 100]   | 2           | Known       |
| B              | 300    | 3000   | [1, 100]   | 5           | Known       |
| C              | 400    | 4000   | [1, 100]   | 2           | Known       |
| D              | 400    | 4000   | [1, 100]   | 5           | Known       |
| NRE            | 500    | 5000   | [1, 100]   | 10          | Unknown     |
| NRF            | 500    | 5000   | [1, 100]   | 20          | Unknown     |
| NRG            | 1000   | 10,000 | [1, 100]   | 2           | Unknown     |
| NRH            | 1000   | 10,000 | [1, 100]   | 5           | Unknown     |

Reducing the instance size of SCP: In [41] different pre-processing approaches have been proposed in particular to reduce the size of the SCP, with Column Domination and Column Inclusion being the most effective. These methods are used to accelerate the processing of the algorithm.

Column Domination is the process of removing unnecessary columns from a problem in such a way that the final solution is unaffected.
Steps:

- All the columns are ordered according to their cost in ascending order.
- If there are equal cost columns, these are sorted in descending order by the number of rows that the column \( j \) covers.
- Verify if the column \( j \) whose rows can be covered by a set of other columns with a cost less than \( c_j \) (cost of the column \( j \)).
- It is said that column \( j \) is dominated and can be eliminated from the problem.

Column Inclusion: when the domination process has terminated, the process of inclusion is performed, which means if a row is covered only by one column, means that there is no best column to cover those rows, which implies, that column must be included in the optimal solution. All of this process will be included to data of instances and let the new solutions satisfying the constraints.

5. Experimental Results

To evaluate the performance of our proposal, we test the instances of the SCP [42], comparing the original CS with different configurations, versus the SACSDBCAN.

5.1. Methodology

To adequately evaluate the performance of metaheuristics, a performance analysis is required [43]. For this work, we compare the supplied best solution of the CSA to the best-known result of the benchmark retrieved from the OR-Library [44]. Figure 2 depicts the procedures involved in doing a thorough examination of the enhanced metaheuristic. We create objectives and recommendations for the experimental design to show that the proposed approach is a viable alternative for determining metaheuristic parameters. Then, as a vital indicator for assessing future results, we evaluate the best value. We use ordinal analysis and statistical testing to evaluate whether a strategy is significantly better in this circumstance. Lastly, we detail the hardware and software aspects that were used to replicate computational experiments, and we present all of the results in tables and graphs.

![Figure 2. Evaluation stages to determine the performance of an metaheuristic.](image)

As a result, we conduct a contrast statistical test for each case, using the Kolmogorov–Smirnov–Lilliefors process [45] to measure sample autonomy and the Mann–Whitney–Wilcoxon [46] test to statistically evaluate the data, in Figure 3 we describe and determinate the organization.

The Kolmogorov-Smirnov-Lilliefors test allows us to assess sample independence by calculating the \( Z_{\text{MIN}} \) or \( Z_{\text{MAX}} \) (depending on whether the task is minimization or maximization) obtained from each instance's 31 executions.
The relative percentage deviation is used to assess the results (RPD). The RPD value computes the difference between the objective value $Z_{\text{min}}$ and the minimal best-known value $Z_{\text{opt}}$ for each instance in our experiment, and it is determined as follows:

$$\text{RPD} = \left( \frac{Z_{\text{min}} - Z_{\text{opt}}}{Z_{\text{opt}}} \right)$$

5.2. Set Covering Problem Results
Infrastructure: Java 1.8 was used to implement SACSDBSCAN. The Personal Computer (PC) has the common attributes: MacOS with a 2.7 GHz Intel Core i7 CPU and 16 GB of RAM.
Setup variables: The configuration for our suggested approach is shown in Table 2 below.

Table 2. SACSDBSCAN Parameters for SCP.

| Population | Initial Abandon Probability | $\alpha$ | Max Iterations | $L_b$ and $U_b$ |
|------------|-----------------------------|----------|----------------|-----------------|
| 10         | 0.1–0.4                     | 0.01     | 5000           | 0 and 1         |

Sixty-five SCP instances were considered, each of which was run 31 times; the results are presented in Tables 3 and 4.
Overview: The algorithms are ranked in order of $Z_{\text{min}}$ achieved. The instances that obtained $Z_{\text{min}}$ are also displayed.
- Score
  1. CS got 38/65 $Z_{\text{min}}$.
  2. SACSDBSCAN got 10/65 $Z_{\text{min}}$.
- The two algorithm got $Z_{\text{min}}$ in the same instances: 5.4–6.4–A.4–B.2–B.3–B.4–B.5–D.3–D.5–NRE.1–NRE.2–NRE.3–NRF.1–NRF.3–NRF.4–NRF.5–NRH.5
As can be seen in the results of the algorithms that solved SCP, we compare the distribution of the samples of each instance using a violin plot, which allows us to observe the entire distribution of the data. We provide and discuss the most difficult instances of each group to create a resume of all the instances below (4.10, 5.10, 6.5, A.5, B.5, C.5, D.5, NRE.5, NRF.5, NRG.5 and NRH.5):

In [47], the authors display the results obtained, the information is detailed, and the configuration that they use. The information is organized as follows: MIN: the minimum...
value reached, MAX: the maximum value reached, AVG: the average value, BKS: the best-known solution, RPD is determined by Equation (43), and lastly the average of the fitness obtained.

The first method was the standard cuckoo search algorithm with various settings, and the second was SACSDBSCAN, as previously indicated. Tables 5–8 show the behavior of our proposed algorithm versus the original algorithm. The best results are highlighted with underline and maroon color. For example, in the instance 4.1, the best reached solution by our proposal overcomes than the classical CS algorithm. The same strategy is used in all comparisons.

Table 3. Results of SACSDBSCAN for instance 4, 5, 6 and A.

| Instance | SACSDBSCAN |
|----------|-------------|
|          | Nest = 10 to 50 | Pa = 0.1 to 0.45 |
|          | BKS | Fit | RDP | AVG | MIN | MAX | STD |
| 4.1      | 429 | 429 | 0.00 | 429 | 429 | 430 | 0.352 |
| 4.2      | 512 | 512 | 0.00 | 513 | 512 | 518 | 1.552 |
| 4.3      | 514 | 516 | 0.00 | 517 | 516 | 520 | 1.407 |
| 4.4      | 494 | 494 | 0.00 | 495 | 494 | 497 | 0.775 |
| 4.5      | 512 | 512 | 0.00 | 512 | 512 | 514 | 0.828 |
| 4.6      | 560 | 560 | 0.00 | 560 | 560 | 564 | 1.121 |
| 4.7      | 430 | 430 | 0.00 | 430 | 430 | 433 | 0.799 |
| 4.8      | 492 | 492 | 0.00 | 495 | 492 | 499 | 2.640 |
| 4.9      | 641 | 641 | 0.00 | 646 | 641 | 653 | 3.680 |
| 4.10     | 514 | 514 | 0.00 | 514 | 514 | 516 | 0.743 |
| 5.1      | 253 | 253 | 0.00 | 253 | 253 | 254 | 0.516 |
| 5.2      | 302 | 303 | 0.00 | 306 | 303 | 312 | 3.355 |
| 5.3      | 226 | 226 | 0.00 | 227 | 226 | 229 | 1.345 |
| 5.4      | 242 | 242 | 0.00 | 243 | 242 | 245 | 1.234 |
| 5.5      | 211 | 211 | 0.00 | 211 | 211 | 212 | 0.458 |
| 5.6      | 213 | 213 | 0.00 | 213 | 213 | 213 | 0.000 |
| 5.7      | 293 | 293 | 0.00 | 293 | 293 | 296 | 0.816 |
| 5.8      | 288 | 288 | 0.00 | 288 | 288 | 289 | 0.258 |
| 5.9      | 279 | 279 | 0.00 | 280 | 279 | 281 | 0.632 |
| 5.10     | 265 | 265 | 0.00 | 266 | 265 | 268 | 1.047 |
| 6.1      | 138 | 138 | 0.00 | 141 | 138 | 144 | 1.807 |
| 6.2      | 146 | 146 | 0.00 | 148 | 146 | 150 | 1.598 |
| 6.3      | 145 | 145 | 0.00 | 148 | 145 | 148 | 1.060 |
| 6.4      | 131 | 131 | 0.00 | 131 | 131 | 133 | 0.737 |
| 6.5      | 161 | 161 | 0.00 | 163 | 161 | 167 | 1.751 |
| A.1      | 253 | 254 | 0.00 | 256 | 254 | 259 | 1.234 |
| A.2      | 252 | 254 | 0.01 | 256 | 254 | 259 | 1.502 |
| A.3      | 232 | 232 | 0.00 | 234 | 232 | 236 | 1.060 |
| A.4      | 234 | 235 | 0.00 | 238 | 235 | 242 | 2.586 |
| A.5      | 236 | 236 | 0.00 | 237 | 236 | 238 | 0.640 |
Table 4. Results of SACSDSCAN for instance B, C, D, NRE, NRF, NRG and NRH.

| Instance | SACSDSCAN |
|----------|-----------|
| Nest = 10 to 50 | Pa = 0.1 to 0.45 |
| BKS | Fit | RDP | AVG | MIN | MAX | STD |
| B.1 | 69 | 73 | 0.05 | 76 | 73 | 79 | 2.610 |
| B.2 | 76 | 76 | 0.00 | 80 | 76 | 87 | 3.218 |
| B.3 | 80 | 80 | 0.00 | 84 | 80 | 88 | 2.673 |
| B.4 | 79 | 79 | 0.00 | 82 | 79 | 88 | 2.560 |
| B.5 | 72 | 72 | 0.00 | 73 | 72 | 78 | 1.710 |
| C.1 | 227 | 227 | 0.00 | 229 | 227 | 234 | 1.993 |
| C.2 | 219 | 219 | 0.00 | 221 | 219 | 225 | 1.685 |
| C.3 | 243 | 243 | 0.00 | 246 | 243 | 250 | 2.131 |
| C.4 | 219 | 219 | 0.00 | 221 | 219 | 224 | 1.309 |
| C.5 | 215 | 215 | 0.00 | 216 | 215 | 217 | 0.799 |
| D.1 | 60 | 61 | 0.02 | 64 | 61 | 66 | 1.414 |
| D.2 | 66 | 70 | 0.06 | 70 | 70 | 70 | 0.000 |
| D.3 | 72 | 76 | 0.05 | 79 | 76 | 82 | 1.859 |
| D.4 | 62 | 63 | 0.02 | 66 | 63 | 67 | 1.486 |
| D.5 | 61 | 63 | 0.03 | 64 | 63 | 66 | 0.910 |
| NRE.1 | 29 | 29 | 0.00 | 30 | 29 | 30 | 0.516 |
| NRE.2 | 30 | 31 | 0.03 | 32 | 31 | 34 | 0.862 |
| NRE.3 | 27 | 28 | 0.04 | 29 | 28 | 31 | 0.976 |
| NRE.4 | 28 | 30 | 0.07 | 31 | 30 | 33 | 0.834 |
| NRE.5 | 28 | 29 | 0.03 | 29 | 29 | 30 | 0.516 |
| NRF.1 | 14 | 15 | 0.07 | 15 | 15 | 16 | 0.458 |
| NRF.2 | 15 | 15 | 0.00 | 16 | 15 | 17 | 0.458 |
| NRF.3 | 14 | 16 | 0.13 | 17 | 16 | 17 | 0.507 |
| NRF.4 | 14 | 15 | 0.07 | 15 | 15 | 16 | 0.389 |
| NRF.5 | 15 | 15 | 0.00 | 15 | 15 | 16 | 0.258 |
| NRG.1 | 176 | 188 | 0.06 | 194 | 188 | 197 | 2.789 |
| NRG.2 | 154 | 160 | 0.04 | 165 | 160 | 167 | 2.282 |
| NRG.3 | 166 | 180 | 0.08 | 182 | 180 | 183 | 0.862 |
| NRG.4 | 168 | 177 | 0.05 | 183 | 177 | 186 | 2.728 |
| NRG.5 | 168 | 181 | 0.07 | 183 | 181 | 184 | 0.799 |
| NRH.1 | 63 | 69 | 0.09 | 71 | 69 | 72 | 0.976 |
| NRH.2 | 63 | 66 | 0.05 | 67 | 66 | 67 | 0.414 |
| NRH.3 | 59 | 66 | 0.11 | 67 | 66 | 68 | 0.738 |
| NRH.4 | 63 | 63 | 0.00 | 64 | 63 | 66 | 1.528 |
| NRH.5 | 55 | 59 | 0.07 | 60 | 59 | 61 | 0.707 |
### Table 5. Comparison results for set of groups 4, 5, 6. A. SACSDBSCAN v/s CS Nest = 20, 30.

| Instance | BKS | SACSDBSCAN | CS | Nest = 10 to 50 | Nest = 20 | Nest = 30 | Fit | RPD | AVG | Fit | RPD | AVG | Fit | RPD | AVG | Fit | RPD | AVG | Fit | RPD | AVG | Fit | RPD | AVG | Fit | RPD | AVG | Fit | RPD | AVG | Fit | RPD | AVG | Fit | RPD | AVG |
|----------|-----|------------|----|----------------|--------|----------|-----|-----|----|-----|-----|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 4.1      | 429 | 429        | 429 | 428            | 427   | 426      | 426 | 426 | 425 | 425 | 425 | 425 | 425 | 425 | 425 | 425 | 425 | 425 | 425 | 425 | 425 | 425 | 425 | 425 | 425 | 425 | 425 | 425 | 425 | 425 | 425 | 425 | 425 | 425 | 425 |
| 4.2      | 512 | 512        | 512 | 512            | 511   | 511      | 511 | 511 | 511 | 511 | 511 | 511 | 511 | 511 | 511 | 511 | 511 | 511 | 511 | 511 | 511 | 511 | 511 | 511 | 511 | 511 | 511 | 511 | 511 | 511 | 511 | 511 | 511 | 511 | 511 |
| 4.3      | 516 | 516        | 516 | 516            | 515   | 515      | 515 | 515 | 515 | 515 | 515 | 515 | 515 | 515 | 515 | 515 | 515 | 515 | 515 | 515 | 515 | 515 | 515 | 515 | 515 | 515 | 515 | 515 | 515 | 515 | 515 | 515 | 515 | 515 |
| 4.4      | 494 | 494        | 494 | 494            | 493   | 493      | 493 | 493 | 493 | 493 | 493 | 493 | 493 | 493 | 493 | 493 | 493 | 493 | 493 | 493 | 493 | 493 | 493 | 493 | 493 | 493 | 493 | 493 | 493 | 493 | 493 | 493 | 493 | 493 | 493 |
| 4.5      | 512 | 512        | 512 | 512            | 512   | 512      | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 | 512 |
| 4.6      | 560 | 560        | 560 | 560            | 560   | 560      | 560 | 560 | 560 | 560 | 560 | 560 | 560 | 560 | 560 | 560 | 560 | 560 | 560 | 560 | 560 | 560 | 560 | 560 | 560 | 560 | 560 | 560 | 560 | 560 | 560 | 560 | 560 | 560 |
| 4.7      | 410 | 410        | 410 | 410            | 410   | 410      | 410 | 410 | 410 | 410 | 410 | 410 | 410 | 410 | 410 | 410 | 410 | 410 | 410 | 410 | 410 | 410 | 410 | 410 | 410 | 410 | 410 | 410 | 410 | 410 | 410 | 410 | 410 | 410 |
| 4.8      | 492 | 492        | 492 | 492            | 492   | 492      | 492 | 492 | 492 | 492 | 492 | 492 | 492 | 492 | 492 | 492 | 492 | 492 | 492 | 492 | 492 | 492 | 492 | 492 | 492 | 492 | 492 | 492 | 492 | 492 | 492 | 492 | 492 | 492 |
| 4.9      | 641 | 641        | 641 | 641            | 641   | 641      | 641 | 641 | 641 | 641 | 641 | 641 | 641 | 641 | 641 | 641 | 641 | 641 | 641 | 641 | 641 | 641 | 641 | 641 | 641 | 641 | 641 | 641 | 641 | 641 | 641 | 641 | 641 | 641 |
| 4.10     | 514 | 514        | 514 | 514            | 514   | 514      | 514 | 514 | 514 | 514 | 514 | 514 | 514 | 514 | 514 | 514 | 514 | 514 | 514 | 514 | 514 | 514 | 514 | 514 | 514 | 514 | 514 | 514 | 514 | 514 | 514 | 514 | 514 | 514 |

For example, the nested parameter values for SACSDBSCAN and CS at Nest = 20 are:

- **SACSDBSCAN**
  - **Fit**: 0.15
  - **RPD**: 0.15
  - **AVG**: 0.15
- **CS**
  - **Fit**: 0.15
  - **RPD**: 0.15
  - **AVG**: 0.15

The comparison results show a close similarity between SACSDBSCAN and CS methods.
### Table 6. Comparison results for sets of groups B, C, D, NRE, NRG, NRE, NRH. SACSDBSCAN vs CS

| Instance | BKS | SACSDBSCAN | CS | CS |
|----------|-----|-------------|----|----|
| | | Fit | RPD | Fit | RPD | Fit | RPD | Fit | RPD | Fit | RPD | Fit | RPD |
| B.1 | 69 | 0.04 | 76 | 88 | 91 | 1031 | 77 | 89 | 91 | 1031 | 78 | 90 | 92 | 1031 |
| B.2 | 76 | 0.05 | 80 | 90 | 93 | 100 | 79 | 90 | 92 | 100 | 80 | 90 | 92 | 100 |
| B.3 | 80 | 0.09 | 85 | 91 | 93 | 102 | 85 | 91 | 93 | 102 | 90 | 95 | 92 | 102 |
| B.4 | 80 | 0.23 | 102 | 107 | 109 | 104 | 106 | 107 | 109 | 104 | 106 | 107 | 109 | 104 |
| B.5 | 72 | 0.07 | 75 | 86 | 90 | 172 | 81 | 89 | 90 | 172 | 86 | 90 | 90 | 172 |
| C.1 | 227 | 0.24 | 239 | 267 | 270 | 172 | 267 | 270 | 172 | 267 | 270 | 172 | 267 | 270 |
| C.2 | 219 | 0.03 | 219 | 265 | 267 | 270 | 267 | 267 | 270 | 267 | 267 | 270 | 267 | 267 |
| C.3 | 243 | 0.03 | 246 | 308 | 313 | 267 | 308 | 313 | 267 | 308 | 313 | 267 | 308 | 313 |
| C.4 | 219 | 0.18 | 217 | 264 | 267 | 270 | 264 | 267 | 270 | 264 | 267 | 270 | 264 | 267 |
| D.1 | 60 | 0.04 | 61 | 70 | 72 | 166 | 70 | 72 | 166 | 70 | 72 | 166 | 70 | 72 |
| D.2 | 66 | 0.04 | 67 | 77 | 80 | 190 | 77 | 80 | 190 | 77 | 80 | 190 | 77 | 80 |
| D.3 | 72 | 0.04 | 72 | 76 | 77 | 190 | 76 | 77 | 190 | 76 | 77 | 190 | 76 | 77 |
| D.4 | 62 | 0.03 | 63 | 66 | 72 | 163 | 72 | 66 | 163 | 72 | 66 | 163 | 72 | 66 |
| D.5 | 61 | 0.04 | 63 | 70 | 71 | 147 | 70 | 71 | 147 | 70 | 71 | 147 | 70 | 71 |
| NRE.1 | 29 | 0.03 | 30 | 51 | 31 | 69 | 31 | 32 | 69 | 31 | 32 | 69 | 31 | 32 |
| NRE.2 | 30 | 0.04 | 31 | 73 | 36 | 233 | 36 | 37 | 233 | 36 | 37 | 233 | 36 | 37 |
| NRE.3 | 27 | 0.02 | 28 | 35 | 36 | 29 | 35 | 36 | 29 | 35 | 36 | 29 | 35 | 36 |
| NRE.4 | 28 | 0.07 | 29 | 34 | 34 | 21 | 34 | 34 | 21 | 34 | 34 | 21 | 34 | 34 |
| NRE.5 | 28 | 0.05 | 29 | 33 | 34 | 17 | 33 | 34 | 17 | 33 | 34 | 17 | 33 | 34 |
| NRF.1 | 14 | 0.03 | 15 | 18 | 18 | 28 | 18 | 18 | 28 | 18 | 18 | 28 | 18 | 18 |
| NRF.2 | 15 | 0.04 | 16 | 19 | 18 | 20 | 18 | 19 | 20 | 18 | 19 | 20 | 18 | 19 |
| NRF.3 | 14 | 0.13 | 17 | 20 | 21 | 42 | 21 | 21 | 42 | 21 | 21 | 42 | 21 | 21 |
| NRF.4 | 14 | 0.04 | 15 | 19 | 20 | 31 | 20 | 19 | 31 | 20 | 19 | 31 | 20 | 19 |
| NRF.5 | 15 | 0.03 | 15 | 16 | 17 | 66 | 16 | 17 | 66 | 16 | 17 | 66 | 16 | 17 |
| NRG.1 | 178 | 0.02 | 187 | 230 | 235 | 308 | 230 | 235 | 308 | 230 | 235 | 308 | 230 | 235 |
| NRG.2 | 154 | 0.04 | 160 | 187 | 192 | 213 | 192 | 192 | 213 | 192 | 192 | 213 | 192 | 192 |
| NRG.3 | 168 | 0.02 | 172 | 198 | 198 | 200 | 198 | 198 | 200 | 198 | 198 | 200 | 198 | 198 |
| NRG.4 | 168 | 0.03 | 175 | 216 | 220 | 220 | 220 | 220 | 220 | 220 | 220 | 220 | 220 | 220 |
| NRG.5 | 168 | 0.03 | 172 | 223 | 306 | 228 | 223 | 306 | 228 | 223 | 306 | 228 | 223 | 306 |
| NRH.1 | 63 | 0.09 | 71 | 82 | 84 | 100 | 82 | 100 | 84 | 100 | 82 | 100 | 84 | 100 |
| NRH.2 | 63 | 0.05 | 67 | 79 | 82 | 25 | 79 | 82 | 25 | 79 | 82 | 25 | 79 | 82 |
| NRH.3 | 59 | 0.33 | 74 | 76 | 77 | 25 | 76 | 77 | 25 | 76 | 77 | 25 | 76 | 77 |
| NRH.4 | 63 | 0.06 | 64 | 74 | 76 | 17 | 74 | 76 | 17 | 74 | 76 | 17 | 74 | 76 |
| NRH.5 | 55 | 0.09 | 57 | 74 | 76 | 21 | 74 | 76 | 21 | 74 | 76 | 21 | 74 | 76 |
| Instance | BKS | Nest = 10 to 50 | CS | Nest = 40, 50 |
|----------|-----|----------------|----|--------------|
| Pa = 0.1 to 0.45 | Pa = 0.1 | Pa = 0.15 | Pa = 0.25 | Pa = 0.35 | Pa = 0.45 | Pa = 0.45 |
| Fit | RPD | AVG | Fit | RPD | AVG | Fit | RPD | AVG | Fit | RPD | AVG | Fit | RPD | AVG | Fit | RPD | AVG | Fit | RPD | AVG | Fit | RPD | AVG | Fit | RPD | AVG |
| 4.1 | 429 | 429 | 0 | 429 | 470 | 475 | 9.56 | 471 | 476 | 9.79 | 467 | 476 | 8.86 | 471 | 476 | 9.79 | 469 | 475 | 9.32 | 470 | 475 | 9.56 | 470 | 475 | 9.56 | 470 | 475 | 9.56 | 470 | 475 | 9.56 |
| 4.2 | 512 | 512 | 0 | 513 | 597 | 611 | 16.60 | 597 | 611 | 16.60 | 597 | 611 | 16.60 | 597 | 611 | 16.60 | 597 | 611 | 16.60 | 597 | 611 | 16.60 | 597 | 611 | 16.60 | 597 | 611 | 16.60 | 597 | 611 | 16.60 |
| 4.3 | 516 | 516 | 0 | 517 | 578 | 585 | 12.45 | 575 | 585 | 11.87 | 575 | 585 | 11.87 | 575 | 585 | 11.87 | 586 | 585 | 10.51 | 575 | 584 | 11.87 | 575 | 584 | 11.87 | 575 | 584 | 11.87 | 575 | 584 | 11.87 |
| 4.4 | 494 | 494 | 0 | 395 | 567 | 579 | 14.78 | 575 | 579 | 14.98 | 598 | 580 | 11.40 | 561 | 581 | 13.36 | 560 | 581 | 13.50 | 560 | 581 | 13.50 | 560 | 581 | 13.50 | 560 | 581 | 13.50 | 560 | 581 | 13.50 |
| 4.5 | 512 | 512 | 0 | 517 | 575 | 612 | 12.30 | 586 | 591 | 15.04 | 589 | 594 | 16.02 | 592 | 614 | 15.63 | 597 | 618 | 16.80 | 592 | 615 | 15.63 | 597 | 618 | 16.80 | 592 | 615 | 15.63 | 597 | 618 | 16.80 |
| 4.6 | 560 | 560 | 0 | 560 | 639 | 651 | 14.11 | 639 | 651 | 14.11 | 639 | 651 | 14.11 | 649 | 639 | 12.32 | 639 | 652 | 14.11 | 637 | 650 | 13.75 | 637 | 650 | 13.75 | 637 | 650 | 13.75 | 637 | 650 | 13.75 |
| 4.7 | 430 | 430 | 0 | 330 | 517 | 524 | 20.23 | 517 | 525 | 19.61 | 517 | 525 | 18.37 | 518 | 525 | 19.77 | 517 | 525 | 20.25 | 518 | 525 | 20.25 | 517 | 525 | 20.25 | 518 | 525 | 20.25 | 517 | 525 | 20.25 |
| 4.8 | 492 | 492 | 0 | 295 | 543 | 555 | 10.37 | 556 | 556 | 11.84 | 556 | 556 | 10.98 | 559 | 559 | 10.98 | 556 | 556 | 10.98 | 556 | 556 | 10.98 | 559 | 559 | 10.98 | 556 | 556 | 10.98 | 556 | 556 | 10.98 |
| 4.9 | 641 | 641 | 0 | 641 | 763 | 784 | 19.03 | 763 | 784 | 19.03 | 763 | 784 | 19.03 | 790 | 780 | 19.03 | 763 | 784 | 19.03 | 780 | 784 | 19.03 | 780 | 784 | 19.03 | 780 | 784 | 19.03 | 780 | 784 | 19.03 |
| 4.10 | 514 | 514 | 0 | 514 | 588 | 608 | 14.40 | 588 | 608 | 14.40 | 596 | 605 | 15.98 | 588 | 608 | 14.40 | 596 | 605 | 15.98 | 588 | 607 | 14.40 | 597 | 609 | 15.76 | 588 | 605 | 14.40 | 597 | 609 | 15.76 |

Table 7. Comparison results for set of groups 4, 5, 6, A. SACSDBSCAN v/s CS Nest = 40, 50.
| Instance | BKS | SACSDSCAN | CS | SACSDSCAN | CS |
|----------|-----|-----------|----|-----------|----|
|          |     | Fit | RPD | Fit | RPD | Fit | RPD | Fit | RPD | Fit | RPD | Fit | RPD | Fit | RPD | Fit | RPD | Fit | RPD | Fit | RPD |
| B1       | 69  | 73  | 0.04| 76  | 88  | 91  | 27.54| 89  | 91  | 27.54| 87  | 91  | 26.09| 88  | 91  | 27.54| 89  | 91  | 26.09| 87  | 91  | 27.54| 89  | 91  | 26.09|
| B2       | 76  | 76  | 0.04| 76  | 88  | 91  | 27.54| 89  | 91  | 27.54| 87  | 91  | 26.09| 88  | 91  | 27.54| 89  | 91  | 26.09| 87  | 91  | 27.54| 89  | 91  | 26.09|
| B3       | 80  | 90  | 0.05| 84  | 90  | 92  | 12.50| 91  | 92  | 13.75| 91  | 93  | 13.75| 91  | 92  | 13.75| 91  | 92  | 13.75| 91  | 92  | 13.75| 91  | 92  | 13.75|
| B4       | 79  | 80  | 0.06| 87  | 89  | 81  | 29.11| 91  | 92  | 11.25| 91  | 92  | 13.75| 91  | 92  | 13.75| 91  | 92  | 13.75| 91  | 92  | 13.75| 91  | 92  | 13.75|
| B5       | 72  | 72  | 0.07| 87  | 89  | 81  | 20.83| 89  | 91  | 18.06| 89  | 87  | 19.44| 89  | 91  | 18.06| 89  | 87  | 19.44| 89  | 91  | 18.06| 89  | 87  | 19.44|

Comparison results for set of groups B, C, D, NRE, NRF, NRG, NRH. SACSDBSCAN v/s CS Nest = 40, 50.

Table 8.
The distribution of the data in all instances (Figures 4–14), shows that the performance of our proposal is better than the traditional cuckoo search optimizer. Concentrating the largest distribution of results in the optimal values, while in original CS they are visibly distant. For example, in the instance B.5, we can see that the distribution is better in our proposal, but at the same time, reflex the behavior to move the result on the best values in his executions, getting the center of the distribution in the best value quartile. Other instance that shows this behaviour can be observed in C.5. Here, our proposal generates again a large number of optimum results.

The scenario in D.5 and NRE.5, is similar in all of them, the behavior of the ASCSDBSCAN show that reach best results compare to CS. In instance NRF.5 we can see that the behavior of both algorithms obtains a similar figure, reflexing the results obtained from the nature of the sample data. In this scenario, we can rescue results in our proposal, obtains betters solutions. The instance NRG.5 is the only scenario where the results of this instance are at six point of distance to obtain an $Z_{opt}$ in SACSDSCAN. Finally, for the instance NRH.5, the behavior of the ASCSDSCAN is again superior than CS.

![Figure 4. Instance 4.10 distribution.](image1)

![Figure 5. Instance 5.10 distribution.](image2)
Figure 6. Instance 6.5 distribution.

Figure 7. Instance A.5 distribution.

Figure 8. Instance B.5 distribution.
Figure 9. Instance C.5 distribution.

Figure 10. Instance D.5 distribution.

Figure 11. Instance NRE.5 distribution.
Figure 12. Instance NRF.5 distribution.

Figure 13. Instance NRG.5 distribution.

Figure 14. Instance NRH.5 distribution.

5.3. Statistical Test

As previously stated, we offer the following hypotheses in order to determine independence:
- $H_0$: states that $Z_{\text{min}}/Z_{\text{max}}$ follows a normal distribution.
- $H_1$: states the opposite.

The test performed has yielded $p\_value$ lower than 0.05; therefore, $H_0$ cannot be assumed. Now that we know that the samples are independent, and it cannot be assumed that they follow a normal distribution, it is not feasible to use the central limit theorem. Therefore, for evaluating the heterogeneity of samples we use a non-parametric evaluation called Mann–Whitney–Wilcoxon test to compare all the results of the hardest instances we propose the following hypotheses:

- $H_0$: CS is better than SACSDDBSCAN.
- $H_1$: states the opposite.

Finally, the statistical contrast test reveals which technique is considerably superior.

The Wilcoxon signed rank test was used to compare SCP on the algorithms techniques for the hardest instances (Tables 9–19). Smaller $p$-values than 0.05 define that $H_0$ cannot be assumed because the significance level is also set to 0.05.

To conduct the test run that supports the study, we use a method from the PISA system. We specify all data distributions (each in a file and each data in a line) in this procedure, and the algorithm returns a $p$-value for the hypotheses.

The following tables show the result of the Mann–Whitney–Wilcoxon test. To understand them, it is necessary to know the following acronyms:

- SWS = Statistically without significance.

**Table 9. $p$-values for instance 4.10.**

|       | CS            | SACSDDBSCAN |
|-------|---------------|-------------|
| CS    | Not applicable| SWS         |
| SACSDDBSCAN | $1.711 \times 10^{-12}$ | Not applicable |

**Table 10. $p$-values for instance 5.10.**

|       | CS            | SACSDDBSCAN |
|-------|---------------|-------------|
| CS    | Not applicable| SWS         |
| SACSDDBSCAN | $2.228 \times 10^{-12}$ | Not applicable |

**Table 11. $p$-values for instance 6.5.**

|       | CS            | SACSDDBSCAN |
|-------|---------------|-------------|
| CS    | Not applicable| SWS         |
| SACSDDBSCAN | $1.711 \times 10^{-12}$ | Not applicable |

**Table 12. $p$-values for instance A.5.**

|       | CS            | SACSDDBSCAN |
|-------|---------------|-------------|
| CS    | Not applicable| SWS         |
| SACSDDBSCAN | $3.055 \times 10^{-12}$ | Not applicable |

**Table 13. $p$-values for instance b.5.**

|       | CS            | SACSDDBSCAN |
|-------|---------------|-------------|
| CS    | Not applicable| SWS         |
| SACSDDBSCAN | $3.417 \times 10^{-12}$ | Not applicable |
Table 14. $p$-values for instance c.5.

|          | CS             | SACSDBSMCAN |
|----------|----------------|-------------|
| CS       | Not applicable | SWS         |
| SACSDBSMCAN | $4.100 \times 10^{-12}$ | Not applicable |

Table 15. $p$-values for instance d.5.

|          | CS             | SACSDBSMCAN |
|----------|----------------|-------------|
| CS       | Not applicable | SWS         |
| SACSDBSMCAN | $1.288 \times 10^{-12}$ | Not applicable |

Table 16. $p$-values for instance NRE.5.

|          | CS             | SACSDBSMCAN |
|----------|----------------|-------------|
| CS       | Not applicable | SWS         |
| SACSDBSMCAN | $1.915 \times 10^{-12}$ | Not applicable |

Table 17. $p$-values for instance NRF.5.

|          | CS             | SACSDBSMCAN |
|----------|----------------|-------------|
| CS       | Not applicable | SWS         |
| SACSDBSMCAN | $2.406 \times 10^{-12}$ | Not applicable |

Table 18. $p$-values for instance NRG.5.

|          | CS             | SACSDBSMCAN |
|----------|----------------|-------------|
| CS       | Not applicable | SWS         |
| SACSDBSMCAN | $3.296 \times 10^{-12}$ | Not applicable |

Table 19. $p$-values for instance NRH.5.

|          | CS             | SACSDBSMCAN |
|----------|----------------|-------------|
| CS       | Not applicable | SWS         |
| SACSDBSMCAN | $2.482 \times 10^{-12}$ | Not applicable |

In all the cases, as mentioned above, the $p$-values reported are less than 0.05, and SWS suggests that they have no statistical significance. So, with this knowledge, in each instance mentioned, we can see the SACSDBSMCAN algorithm was better than the original CS.

If we focus on the instances where our proposal improves the result obtained in comparison to the original CS algorithm, we can infer that the solutions achieved are distributed in a centered way on their optimal value, which reflects that the behavior of this algorithm is very positive. This is reflected in the violin Figure 8 or Figures 10 and 14.

5.4. Comparison Results in Similar Hybrid Algorithms

Within the literature, recent studies can be found that use hybrid algorithms that solve the coverage problem [5,48–50]. However, to compare a hybrid algorithm that resembles our proposal, we have considered making a comparison of results with hybrid algorithms that work with bio-inspired metaheuristics, improved by ML, and they solve the set covering problem. In this scheme, we have three algorithms. The first one is the crow search algorithm boosted by the DBSCAN method (called CSADBSMCAN). The second
studied approach was the integration between the crow search algorithm and the Kmean method (CSAKmean). Both hybridizations were proposed by Valdivia et al. in [51]. Finally, we employ an improved version of the cuckoo search algorithm with the Kmean transition algorithm (KMTA), recently proposed by García et al. in [52].

Tables 20 and 21 present best values reached in CSADBSCAN, CSAKmean and KMTA. Those algorithms implement different strategies to improve metaheuristics with ML. To resume and centering the results of the best values obtained, we add the AVG measure in the final row of each table. Unfortunately, KMTA only reports results of the first of each family instance, so N/R means Not Reported.

Table 20. Best values reached by an improved bio-inspired algorithm with ML that solve the set covering problem Instances 4, 5, 6 and A.

| Instance | BKS | SACSDSCAN | CSADBSCAN | CSAKmean | KMTA |
|----------|-----|-----------|-----------|----------|------|
|          |     | Best      | RDP       | Best      | RDP  |
| 4.1      | 429 | 429       | 0.000     | 429       | 0.000 |
| 4.2      | 512 | 512       | 0.000     | 512       | 0.000 |
| 4.3      | 516 | 516       | 0.000     | 516       | 0.000 |
| 4.4      | 494 | 494       | 0.000     | 494       | 0.000 |
| 4.5      | 512 | 512       | 0.000     | 512       | 0.000 |
| 4.6      | 560 | 560       | 0.000     | 560       | 0.000 |
| 4.7      | 430 | 430       | 0.000     | 430       | 0.000 |
| 4.8      | 492 | 492       | 0.000     | 492       | 0.000 |
| 4.9      | 641 | 641       | 0.000     | 641       | 0.000 |
| 4.10     | 514 | 514       | 0.000     | 514       | 0.000 |
| 5.1      | 253 | 253       | 0.000     | 253       | 0.000 |
| 5.2      | 302 | 303       | 0.003     | 302       | 0.000 |
| 5.3      | 226 | 226       | 0.000     | 226       | 0.000 |
| 5.4      | 242 | 242       | 0.000     | 242       | 0.000 |
| 5.5      | 211 | 211       | 0.000     | 211       | 0.000 |
| 5.6      | 213 | 213       | 0.000     | 213       | 0.000 |
| 5.7      | 293 | 293       | 0.000     | 293       | 0.000 |
| 5.8      | 288 | 288       | 0.000     | 288       | 0.000 |
| 5.9      | 279 | 279       | 0.000     | 279       | 0.000 |
| 5.10     | 265 | 265       | 0.000     | 265       | 0.000 |
| 6.1      | 138 | 138       | 0.000     | 140       | 0.014 |
| 6.2      | 146 | 146       | 0.000     | 146       | 0.000 |
| 6.3      | 145 | 145       | 0.000     | 145       | 0.000 |
| 6.4      | 131 | 131       | 0.000     | 131       | 0.000 |
| 6.5      | 161 | 161       | 0.000     | 162       | 0.006 |
| A.1      | 253 | 254       | 0.004     | 254       | 0.004 |
| A.2      | 252 | 254       | 0.008     | 256       | 0.016 |
| A.3      | 232 | 232       | 0.000     | 233       | 0.004 |
| A.4      | 234 | 235       | 0.004     | 236       | 0.009 |
| A.5      | 236 | 236       | 0.000     | 236       | 0.000 |
| AVG      | –   | –         | 0.001     | –         | 0.002 |

Table 20 shows how SACSDSCAN obtain better results average in comparison with the other algorithms for the instances 4, 5, 6 y A. Table 21 shows how SACSDSCAN obtain better results average in comparison with CSADBSCAN and CSAKmean in a very wide value to CSADBSCAN, a difference to 1.141 and 0.038 distance with CSAKmean. Only KMTA obtain the best AVG value with the reported best values with 0.018 difference.
Table 21. Best values reached by an improved bio-inspired algorithm with ML that solve the set covering problem Instances B, C, D, NRE, NRF, NRG, and NRH.

| Instance | BKS | SAC | DBSCAN | CSAD | CSADB | CSAK | KMTA |
|----------|-----|-----|--------|------|-------|------|------|
|          |     | Best | RDP | Best | RDP | Best | RDP |
| B.1      | 69  | 73   | 0.058 | 69  | 0.000 | 74  | 0.072 | 69  | 0.000 |
| B.2      | 76  | 76   | 0.000 | 81  | 0.066 | 83  | 0.092 | N/R | N/R   |
| B.3      | 80  | 80   | 0.000 | 82  | 0.025 | 84  | 0.050 | N/R | N/R   |
| B.4      | 79  | 79   | 0.000 | 83  | 0.051 | 84  | 0.063 | N/R | N/R   |
| B.5      | 72  | 72   | 0.000 | 78  | 0.083 | 72  | 0.000 | N/R | N/R   |
| C.1      | 227 | 227  | 0.000 | 233 | 0.026 | 228 | 0.004 | 229 | 0.009 |
| C.2      | 219 | 219  | 0.000 | 226 | 0.032 | 226 | 0.032 | N/R | N/R   |
| C.3      | 243 | 243  | 0.000 | 253 | 0.041 | 254 | 0.045 | N/R | N/R   |
| C.4      | 219 | 219  | 0.000 | 224 | 0.023 | 225 | 0.027 | N/R | N/R   |
| C.5      | 215 | 215  | 0.000 | 222 | 0.033 | 215 | 0.000 | N/R | N/R   |
| D.1      | 60  | 61   | 0.017 | 68  | 0.133 | 66  | 0.100 | 60  | 0.000 |
| D.2      | 66  | 70   | 0.061 | 73  | 0.106 | 71  | 0.076 | N/R | N/R   |
| D.3      | 72  | 76   | 0.056 | 82  | 0.139 | 82  | 0.139 | N/R | N/R   |
| D.4      | 62  | 63   | 0.016 | 70  | 0.129 | 67  | 0.081 | N/R | N/R   |
| D.5      | 61  | 63   | 0.033 | 72  | 0.180 | 66  | 0.082 | N/R | N/R   |
| NRE.1    | 29  | 29   | 0.000 | 70  | 1.141 | 30  | 0.034 | 29  | 0.000 |
| NRE.2    | 30  | 31   | 0.033 | 83  | 1.767 | 34  | 0.133 | N/R | N/R   |
| NRE.3    | 27  | 28   | 0.037 | 74  | 1.741 | 34  | 0.259 | N/R | N/R   |
| NRE.4    | 28  | 30   | 0.071 | 83  | 1.964 | 33  | 0.179 | N/R | N/R   |
| NRE.5    | 28  | 29   | 0.036 | 92  | 2.286 | 30  | 0.071 | N/R | N/R   |
| NRF.1    | 14  | 15   | 0.071 | 372 | 25.571 | 17 | 0.214 | 14  | 0.000 |
| NRF.2    | 15  | 15   | 0.000 | 74  | 3.933 | 18  | 0.200 | N/R | N/R   |
| NRF.3    | 14  | 16   | 0.143 | 335 | 22.929 | 19  | 0.357 | N/R | N/R   |
| NRF.4    | 14  | 15   | 0.071 | 52  | 2.714 | 18  | 0.286 | N/R | N/R   |
| NRF.5    | 15  | 15   | 0.000 | 65  | 3.333 | 16  | 0.067 | N/R | N/R   |
| NRG.1    | 176 | 188  | 0.068 | 287 | 0.631 | 197 | 0.119 | 176 | 0.000 |
| NRG.2    | 154 | 160  | 0.039 | 208 | 0.351 | 168 | 0.091 | N/R | N/R   |
| NRG.3    | 166 | 180  | 0.084 | 211 | 0.271 | 183 | 0.102 | N/R | N/R   |
| NRG.4    | 168 | 177  | 0.054 | 250 | 0.488 | 186 | 0.107 | N/R | N/R   |
| NRG.5    | 168 | 181  | 0.077 | 230 | 0.369 | 183 | 0.089 | N/R | N/R   |
| NRH.1    | 63  | 69   | 0.095 | N/R | N/R   | 71  | 0.127 | 64  | 0.016 |
| NRH.2    | 63  | 66   | 0.048 | N/R | N/R   | 71  | 0.127 | N/R | N/R   |
| NRH.3    | 59  | 66   | 0.119 | N/R | N/R   | 69  | 0.169 | N/R | N/R   |
| NRH.4    | 63  | 63   | 0.000 | N/R | N/R   | 68  | 0.079 | N/R | N/R   |
| NRH.5    | 55  | 59   | 0.073 | N/R | N/R   | 61  | 0.109 | N/R | N/R   |
| AVG      | –   | –    | 0.021 | –   | 1.162 | –   | 0.059 | –   | 0.003 |

6. Conclusions

In this paper, we can conclude that the use of the machine learning technique to make a metaheuristic autonomous parameters setting has the 38/65 min values fitness on the test that we make, in one single configuration over the 20 different configurations, that demonstrates that with our proposed algorithm, it is not necessary make the complex task to find the best parameter setting of the metaheuristic CS, that is, in most of the time, in a try-and-error way. The result of the experiment demonstrates that it is positive to use the DBSCAN algorithm to infer his result and use that information to let us make changes to the values of the parameters. The comparison results with other bio-inspired hybrid algorithms applied on the set covering problem demonstrate that the use of DBSCAN obtains better results on average fitness values in comparison with studies that report all
the best results values. The exploration criteria that we use can let the algorithm vary the search space to find another best candidate, as we saw in the box graphs and the distribution of the instances. In addition, the use of the noise points associate with the Pa, lets the SACSDBSCAN keep the variety on his behavior and not forget the stochastic factor that characterizes a metaheuristic.

The free execution parameter allows the metaheuristic to maintain its natural behavior across executions. At the time of reaching the freedom parameter, we can analyze the results of the metaheuristics and classify its results space to be able to make its classification and corresponding intervention of the possible candidate solutions, eliminating the worst for possible new better solutions.

As future work, we consider implementing an improvement to the criterion of population increase and decrease using clusterization strategies. In another line of work, we want to implement different machine learning techniques to be able to perform algorithms that allow effective use of the population increase/decrease in metaheuristics, and thus be able to deliver tools that make these algorithms more efficient. In addition, we are considering contributing new works that provide comparisons of this algorithm with other hybrid variants of cuckoo search, and other types of metaheuristics that use ML to improve their performance, different from those presented in this work, used to solve continuous problems.

**Author Contributions:** Formal analysis, N.C., R.S. and B.C.; Investigation, N.C., R.S., B.C., S.V. and R.O.; Methodology, R.S., R.O. and S.V.; Resources, R.S. and B.C.; Software, N.C., S.V. and R.O.; Validation, B.C. and S.V.; writing—original draft, N.C., S.V. and R.O.; writing—review and editing, N.C., R.S., B.C., S.V. and R.O. All authors have read and agreed to the published version of the manuscript.

**Funding:** Ricardo Soto is supported by grant CONICYT/FONDECYT/REGULAR/1190129. Broderick Crawford is supported by grant ANID/FONDECYT/REGULAR/1210810. Nicolás Caselli is supported by grant INF-PUCV 2019–2021.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** No new data were created or analyzed in this study. Data sharing is not applicable to this article.

**Conflicts of Interest:** The authors declare no conflict of interest. The founding sponsors had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, and in the decision to publish the results.

**Abbreviations**

**Symbol**

| Symbol | Description |
|--------|-------------|
| α      | Step size to generate the next solution |
| Pa     | Probability abandon of CSA |
| β      | Scalar number between 0 and 3 |
| ε      | Max Distance with each other point |
| T      | Max number of Iterations |
| t      | Current Iteration |
| Lb     | Lower Bound |
| Ub     | Upper Bound |
| Xid(t) | Solution ith in dimension d at iteration t |

**Acronyms**

| Acronym      | Description               |
|--------------|---------------------------|
| ML           | Machine Learning          |
| CSA          | Cuckoo Search Algorithm   |
| SACSDBSCAN   | Self-Adaptive Cuckoo Search DBSCAN |
DBSCAN
SCP
RPD
CSADBCAN
CSAKmean
KMTA

Spatial Clustering Based on Noise Application Density
Set Covering Problem
Relative Percentage Deviation
Crow Search Algorithm boosted by the DBSCAN method
crow search algorithm and the Kmean method
Kmean transition algorithm

References
1. Olivares, R.; Muñoz, F.; Riquelme, F. A multi-objective linear threshold influence spread model solved by swarm intelligence-based methods. *Knowl.-Based Syst.* 2021, 212, 106623. [CrossRef]
2. Soto, R.; Crawford, B.; Olivares, R.; Carrasco, C.; Rodriguez-Tello, E.; Castro, C.; Paredes, F.; de la Fuente-Mella, H. A reactive population approach on the dolphin echolocation algorithm for solving cell manufacturing systems. *Mathematics* 2020, 8, 1389. [CrossRef]
3. Munoz, R.; Olivares, R.; Munoz, R.; Soto, R.; Villar, M.; de Albuquerque, V.H.C. The patient bed assignment problem solved by autonomous bat algorithm. *Appl. Soft Comput.* 2019, 81, 105484. [CrossRef]
4. Huang, C.; Li, Y.; Yao, X. A survey of automatic parameter tuning methods for metaheuristics. *Artif. Intell.* 2019, 8, 4–16. [CrossRef]
5. Crawford, B.; Soto, R.; Embry, G.; Flores, D.; Palma, W.; Castro, C.; Paredes, F.; Rubio, J.M. A binary monkey search algorithm variation for solving the set covering problem. *Lot. Plann. Sci. 2021, 73, 100850.* [CrossRef]
6. Michalewicz, Z.; Fogel, D.B. *How to Solve It: Modern Heuristics*; Springer Science & Business Media: Berlin, Germany, 2013.
7. García, J.; Yepes, V.; Marti, J.V. A Hybrid k-Means Cuckoo Search Algorithm Applied to the Counterfort Retaining Walls Problem. *Mathematics* 2020, 8, 555. [CrossRef]
8. Rosli, S.J.; Rahim, H.A.; Abdul Rani, K.N.; Ngadiran, R.; Ahmad, R.B.; Yahaya, N.Z.; Abdulmalek, M.; Jasoh, M.; Yasin, M.N.M.; Sabapathy, T. et al. A Hybrid Modified Method of the Sine Cosine Algorithm Using Latin Hypercube Sampling with the Cuckoo Search Algorithm for Optimization Problems. *Electronics 2020, 9, 1786.* [CrossRef]
9. Lin, H.; Siu, S.W.I. A Hybrid Cuckoo Search and Differential Evolution Approach to Protein—Ligand Docking. *Int. J. Mol. Sci.* 2018, 19, 3181. [CrossRef] [PubMed]
10. Saeed, S.; Ong, H.C.; Sathasivam, S. Self-Adaptive Single Objective Hybrid Algorithm for Unconstrained and Constrained Test functions: An Application of Optimization Algorithm. *Arab. J. Sci. Eng.* 2019, 44, 3497–3513. [CrossRef]
11. Thirumal, V.; Soto, R.; Alonso-Sánchez, M.F.; Merino, E.; de Albuquerque, V.H.C. A new EEG software that supports emotion recognition by using an autonomous approach. *Neural Comput. Appl.* 2018, 32, 11111–11127. [CrossRef]
12. Senjyu, T.; Saber, A.; Miyagi, T.; Shimabukuro, K.; Urasaki, N.; Funabashi, T. Fast technique for unit commitment by genetic algorithm based on unit clustering. *IEEE Proc.-Gener. Transm. Distrib.* 2005, 152, 705–713. [CrossRef]
13. Tan, X.; Duan, J.; Lee, C.; Gen, M.; Kuo, W. Reliability optimization designs using a hybridized genetic algorithm with a neural-network technique. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* 2001, 84, 627–637.
14. Bouktif, S.; Fiaz, A.; Ouni, A.; Serhani, M.A. Multi-Sequence LSTM-RNN Deep Learning and Metaheuristics for Electric Load Forecasting. *Energies 2020, 13, 391.* [CrossRef]
