Artificial Bee Colony (ABC) algorithm and Its Implementation in Clustering

Yangzhuoran (Fin) Yang (28150295)

Contents

1 Introduction

Artificial Bee Colony (ABC) algorithm is a meta-heuristic optimization algorithm recently introduced by Karaboga (2005). It simulates the behaviour of a honey bee swarm in the attempt to find the optimal solution. As a general optimization algorithm, it is not limited to Clustering problem. We now give a brief overview of how the ABC algorithm works and summarize the algorithm in the pseudo-code that follows. Introduction of each component in detail is given later in the clustering section. Part of the notation and formulations are adopted from Dervis Karaboga and Ozturk (2011).

1.1 Overview

The ABC algorithm can be structured into three phases: the employed bee phase, the onlooker bee phase, and the scout bee phase. Each phase mimics the behaviour of a group of bees in a honey bee swarm.

Before the algorithm enters the employed bee phase, the total number of food sources (the solution set) is to be determined and denoted as SN (swarm size). We will discuss the impact of swarm size in more detail in the parameter section. Akay and Dervis Karaboga (2009) argue the robustness of the algorithm is not affected by different swarm size in general.

After SN being decided, the ABC algorithm will simulate the position of initial food sources. The way to simulate the food sources has been tailored in different problems in the literature: they can be evenly assigned across the solution space (Vega Yon and Muñoz (2017)), randomly generated from a distribution (Dervis Karaboga and Ozturk (2011)), or they can be randomly selected from different data points for the problem of clustering. The main idea is to cover the solution space as much as possible.

The number of employed bees or the number of the onlooker bees is the same as the swarm size, or to be specified by the user. In the employed bee phase, the bees search locally to find a neighbour, create a new solution by combining the existing solutions, then decide if to replace the current solution with the new solution using a greedy selection approach. The onlooker bee then performs similar neighbourhood search, the difference is the onlooker bee will search neighbours and create new solutions around existing solutions with better quality, so the onlooker bees provide a tendency towards where it is likely to produce a good solution. The way that the employed bees and onlooker bees find a neighbour and create new solutions and which solution that the onlooker bees select can be calculated using the general methods in the ABC algorithm, or can be different depending on the problem at hand.

After the phases of the employed bee and onlooker bee, whether one solution has been improved is recorded for each food source. If one solution could not be improved up to some certain number of iteration, it will be discarded and the scout bees will find a replacement in the solution space to fill the position, where the way to find new solutions can be independent with the current solutions. The employed bee and the onlooker bee search locally while the scout bee is in charge of the global search. In other words, the employed bee and the onlooker bee emphasis intensification by producing better solutions based on the current solution set, while the scout bee emphasis diversification search solutions independently from the current set of solutions. The ABC algorithm aims to achieve a balance between intensification and diversification through three different phases mimicking three different types of bee.

The updated solution in each iteration is the best solution among all the food sources in that iteration, and the final solution chosen by the algorithm is the best solution among all the food sources tried in the past up until the time the condition is met. Stopping condition are required to determine when to stop the algorithm. Dervis Karaboga and Ozturk (2011) choose to use the maximum number of iteration as the stopping condition. Vega Yon and Muñoz (2017) includes the number of unimproved iteration: the algorithm stops when the result does not improve up to a certain number.

The ABC algorithm can be summarized in the following pseudo-code.

```
Algorithm: Artifical Bee Colony
 1. Load the training data
 2. Generate the initial food sources 1,2,..., SN
 3. Evaluate the quality of nectar (the fitness of initial solutions)
 4. While (Condition not met)
        The employed bee phase
 5.     For each employed bee{
          Produce a new solution using neighbourhood search
          Calculate the fitness
          Select the better-fitted solution Greedily }
 6.     Calculate the probabilities of selecting each solution
        The onlooker bee phase
 7.     For each onlooker bee{
          Select a solution based on the probability calculated above
          Produce a new solution using neighbourhood search
          Calculate the fitness
          Select the better-fitted solution Greedily }
```

```
 8.     Abandon the solution that the number of unimproved iteration reaches the limit
        The scout bee phase
 9.       Increase the number of food source to SN by finding new solution randomly
10.     Record the best solution among all food sources
11. End
```

1.2 Parameters

There are three parameters in the ABC determines the quality of optimization and needs to be taken carefully: 1. the number of food sources (SN) which may also be the number of employed bees or the onlooker bees; 2. the value of limit that decides when to abandon food sources; 3. the stopping conditions, such as the maximum cycle number (MCN) or the maximum number of unchanged iteration.

The number of employed bees, the onlooker bees, or the scout bees can also be customized based on the specified problem. As Dervis Karaboga and Ozturk (2011) emphasis the three parameters mentioned above, we will assume the number of the employed bees or the onlooker bees are the same as the swarm size SN and assume the number of scout bees is one per iteration in the following discussion, which would be sufficient in general optimization problem.

1.2.1 The number of food sources (SN)

A large number of food sources can increase the execution time for each iteration dramatically. It is the number of neighbourhood search and greedy evaluation conducted in the employed bee phase or the onlooker bee phase. For each iteration, there are effective $2 \times SN$ neighbourhood searches conducted by the employed bee and the onlooker bee. While a large SN increase the time it needs to run for each iteration, it also means the local search is more thoroughly implemented: the number of the solution we are considering at the same time is large. Therefore, there is a higher chance of finding a better solution.

1.2.2 The limit

The limit controls the balance between intensification and diversification. If the limit is set to be too large, a useless food source needs more time to be dropped and more time is needed to trigger the global search implemented by the scout bee, which may lead to unnecessary intensification. If the limit is set to be too small, a promising food source may be dropped before it runs the time it needs to produce a good solution, and the scout bee is triggered too early, leading to diversification more then intensification.

1.2.3 The stopping condition

If we consider the maximum cycle number (MCN) as the stopping condition, a small value of the MCN may stop the algorithm too early then the optimization settles down to the optimal solution. If MCN is set to be too large, the algorithm may run many extra iterations without any improvement, leading to poor efficiency. The maximum number of unchanged iteration works the same way: the algorithm may stops before the solution converges.

1.3 Comparison

We compare the ABC algorithm to two other meta-heuristics: The Simulated Annealing and the Genetic Algorithm. The ABC algorithm shares some of the same features of other meta-heuristics while having its different technic to some of the other problems. We now look at how the elements in those algorithms are presented in the ABC algorithm.

1.3.1 Escape local optimum differently from Simulated Annealing

The simulated annealing differs from the basic neighbourhood by allowing the algorithm to accept a solution with worse fitness with certain probabilities. With such a feature that mimic the process of annealing metals

to climb the "hill" in the curve of the objective function, it can escape the local minimum (when it is a minimization problem). The probability of accepting a worse move decreases with time.

The ABC algorithm does not have the feature to accept a worse move if we consider the path of the solution in ABC as the best solution among all the food sources in each iteration. However, it has similar functionality to avoid being stuck in the local minimum: it abandons the food source that has not been improved for some certain amount of iteration. By dropping the unimproved food source, it effectively drops the solutions that are in the local minimum. By consider all the food sources collectively and add in new food sources independently from local search, it expands its search space.

1.3.2 Reproduce like Genetic Algorithm (GA)

The genetic algorithm imitates the process of evolution. It selects multiple solutions (selection), creates a new solution from those existing solutions (crossover), modifies the solution to create another new solution (mutation), calculates the objective value of the new solution (evaluation) and determines whether to replace an existing solution using the new solution (update).

Many components in the ABC algorithm largely share the same process in the genetic algorithm. The bees finding neighbours is corresponding to the selection in GA. Creating a new solution using the neighbour is corresponding to the crossover in GA. calculating the fitness of the new solution and deciding whether to take the new solution are corresponding to evaluation and update. The difference is for the step of mutation. In GA, the mutation is done separately from the crossover, but in ABC the mutation step is embedded in the crossover: after selecting a neighbour, the bee creates the new solution not only based on the existing ones but also adding a random component. (The random number generation in Equation (3) which is to be discussed later.)

1.3.3 Additional features

1.3.3.1 Global search

The scout bee phase in the ANC provides an additional global search that is lacked in both the simulated annealing and the genetic algorithm. Although the diversification can be achieved using different parameters in those two methods, using separate component guarantees certain exposure to the whole solution space.

1.3.3.2 Tendency towards the better-performed solution

The onlooker bees find the food sources with better quality and conduct search around them: the create new solutions using good solutions. This improves the efficiency of the algorithm as intuitively good solutions may cluster together. A similar feature can be found in some modified genetic algorithm. The way to select the parent solutions may be weighted towards good solutions.

2 Applying the ABC Algorithm to Solve Sudoku Puzzles

Pacurib, Seno, and Yusiong (2009) provides solution of Sudoku puzzles using the ABC algorithm. Sudoku puzzle is a logic-based combinatorial puzzle. Players are given a map of $n \times n$ squares (cells) with some of the squares filled with numbers called the starting squares. The player aims to fill out the rest of the cells based on the following three rules

1. A unique number can only appear once in each row
2. A unique number can only appear once in each column
3. A unique number can only appear once in a $m \times m$ predefined sub-block.

Figure 1 is an example of 9×9 sudoku puzzle with 3×3 sub-blocks.

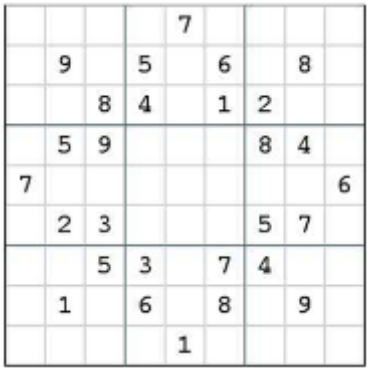

Figure 1: An example of sudoku puzzle from Pacurib, J. A., Seno, G. M. M., and Yusiong, J. P. T. (2009, December)

Pacurib, Seno, and Yusiong (2009) apply the ABC algorithm by imposing the third constraint in the solution space while only evaluate objective function using the other two constraints in the algorithm. The solution representation would simply be a vector of numbers, where the position of the number corresponds to the index of the cell they are in. The optimization problem is to minimize the number of duplicate digits found on each row and column, while the third constraint is set to be always satisfied for a valid solution.

2.1 The parameters specification

The number of food sources in the experiments of Pacurib, Seno, and Yusiong (2009) is defined to be 100, the same as the number of the employed bees, while the number of onlooker bees to be set as 200. The number of scout bees is set to be 10% of the employed bees which is 10. The maximum number of iteration is set to be 100,000.

In general ABC algorithm, the food sources are abandoned when the number that they do not change accumulates to the limit. In the Sudoku problem, the worst-performing food sources are paired to a randomly generated new food source by the scout bee, and if new food source has higher fitness than the old one, the old one is replaced, so the value of the limit is no longer a parameter needed in this problem.

There are two stopping criteria. While the objective function is the number of duplicate digits, one of the criteria is having a fitness value of 1. If this criterion is met it means that the optimal solution to the puzzle has been found. If this criterion is not met, instead the algorithm stops when it reaches the maximum number of cycles, it means the algorithm has not yet found the optimal solution to the puzzle and it produced the best solution obtained at the time the algorithm stops.

2.2 Adjusted approach

The initial food sources are simulated by randomly place digits within the bounds in the cells which satisfying the third constraint. The approach of creating a new solution is adjusted to fit this specific problem. Given the solution representation of a vector of digits, a random number j is chosen for the feasible solution X_i and the randomly chosen neighbour X_k. If we denote the new solution as V_i, the value of each element (denoted using subscript j) of V_i is determined by the following equation:

$$V_{ij} = X_{ij} + rand[0, 1] \times |X_{ij} - X_{ij}|$$

The function to generate new solution is defined using the uniform random number between 0 and 1 and the absolute value of the difference to ensure no negative solution is created. If the value obtained is greater than the value that is allowed in the sudoku puzzle, for example, 9 in the 9×9 puzzle, the modulo of the value plus one is used as the final value of V_{ij}.

There is no guarantee that the new solution is going to satisfied the third rule above. If a new solution violates the sub-block constraint, a swap operation is triggered: the original location of the violating element V_{ij} us replaced with X_{ij}. Then the feasible new solution will be considered in the greedy selection approach comparing the current solution.

The fitness is calculated using the generic function in Equation (1) and the probability for the onlooker bee to find food source is calculated using the general function in Equation (4). We will introduce them in the clustering section.

2.3 Experiments and Conclusion

Pacurib, Seno, and Yusiong (2009) apply the ABC algorithm to sudoku puzzles with three different difficulties. By comparing the average number of cycles needed to solve the sudoku puzzles and the average time needed, they conclude that the more difficult the puzzle is, the longer it needs to solve the puzzle. They also find that the modified ABC algorithm outperforms the Genetic-Algorithm-based sudoku solver.

3 Detailed process in Clustering

We now give the ABC algorithm in the context of clustering, with the decisions of choosing problem specific parts of the heuristic discussed in detail in the process.

3.1 Solution Representation and Objective Function

The problem of clustering involves dividing data points into different groups with respect to the similarity in their features without any information on the true classification of the data. It is an unsupervised learning problem. The solution representation needs to indicates the group distributed to each data point. We use the closest allocation scheme, where the data points are allocated to the nearest centroids, the centres of the clusters. The clustering problem effectively becomes optimization for finding the best location of centroids, while the unique solution of the cluster of each point can be found using the closest allocation.

At the beginning of the algorithm, the number of clusters k needs to be given. Once we have the data with dimension D, we can write the centroids in the form of a $k \times D$ matrix where each row indicates one centroid and each column corresponds to one coordinate. The solution representation is effectively a vector with a length of $k \times D$.

The objective function we are using is the Davis-Bouldin index, where the distance is calculated using the Euclidean norm. The DB index measures the distances within a cluster relative to the distance to between clusters, which matches our approach of closest allocation.

3.2 Data processing

To generalize our approach, we add the step of data standardization before we run the algorithm. Depending on the magnitude of a variable in the data set, variables may be weight differently in the clustering problem. Although the decision of whether to standardize the data depends largely on the nature of the data set, we propose the standardization step to give even weight for each variable.

The data is standardized using the following function:

$$z_{ij}^* = \frac{z_{ij}}{\max_j |z_{ij}|}$$

where i denotes the observation, j is the index of the variable. Using this approach, the coordinates of the data point will be bounded between -1 and 1 (if the variable only has positive values, it will be bounded between 0 and 1). The bounded can be a valid starting point in simulation in the scout bee phase and in setting initial value in general. It also drives the values that are close to zero away from 0 when the variable is intrinsically smaller than 1.

We do not use the popular standardization that centres the variable to have mean zero and scales standard deviation to one, as the bound of data set will be too arbitrary to uniformly apply. We do recommend using different scaling method based on the nature of the data set.

3.3 Initialization

To mimic the behaviour of a bee swarm, the ABC algorithm needs a parameter that defines the size of the swarm: the number of food sources, or the number of solutions in the solution set. We denote this number as SN (swarm size). We then simulate the position of initial food sources (the set of solutions) $z_i : i = 1, 2, \ldots, SN$. In our clustering problem, the food sources are simulated such that each centroid in each solution is randomly sampled from the data points, with the constraint that each cluster has at least $\frac{n}{2k}$ points where n is the number of points in the data set. If the food source simulated does not satisfy the constraint, a new food source will be simulated to replace the old one until all food sources meet the constraint.

When we simulate the food source, we do not using the methods from Vega Yon and Muñoz (2017) where the food sources are evenly assigned across the solution space, because such an assignment will result in the solution not meeting the minimum number of points in one cluster. We do not randomly generate the food sources uniformly given the upper bound and lower bound of the solution for the same reason. A larger number of simulation is required to produce feasible food sources in such a setting.

Once the position of the inition solutions has been determined, the fitness $f_i : i = 1, 2, \ldots, SN$ can be calculated from corresponding cost function/objuective function. The quality of the nectar $fit_i : i = 1, 2, \ldots, SN$ in the ABC algorithm can be calculated correspondly, using Equation (1)

$$fit_i = \frac{1}{1 + f_i} \tag{1}$$

In the case when the cost function produces negative fitness, the quality of the nectar can be calculated by:

$$fit_i = 1 + |f_i| \tag{2}$$

We do not need to use this function as in the problem of clustering, the distance representation is always positive.

3.4 The Employed Bee

The number of employed bees is set to be the same as the number of food sources SN as in the basic ABC algorithm. For each employed bee at each food source, the bee implements a neighbourhood search to find a new solution by combining the neighbour with the current position using

$$\nu_{ij} = z_{ij} + \phi_{ij}(z_{ij} - z_{kj}) \tag{3}$$

where $i \in \{1, 2, \ldots, SN\}$ is the index of the current solution, and $j \in \{1, 2, \ldots, SN\}$ is the randomly generated index of the neighbour. If we denote D as the number of elements we need to optimize in one solution (the number of dimensions), then $k \in \{1, 2, \ldots, D\}$ is the randomly generated index denoting the position of the element. ϕ_{ij} is a random number simulated using a uniform distribution with bound -1 and 1.

Other than the above general step to generate new solutions, in the clustering problem, we add the additional constraints that the new solution needs to produce clusters that have minimum $\frac{n}{2k}$ data points in them. If the constraint is not satisfied, the bee would skip the greedy algorithm and forget the new solution before entering the next phase.

If a feasible solution is found, the employed decide on whether to jump from the current solution to the new solution by comparing the quality of the nectar of two positions. If the bee decides to jump to the new

position with a higher value of fit_i, it will forget the old position, i.e. the old solution was not stored in the memory of the algorithm.

3.5 The Onlooker Bee

The onlooker bee performs the same local search as the employed bee. The difference is the onlooker bee does not implement the search on every food source, but selectively perform the search based on the quality of the nectar of each food source. After the employed bee phase, the algorithm calculates the probabilities p_i of the onlooker bee selecting each food source based on the following equation:

$$p_i = \frac{fit_i}{\sum_{i=1}^{SN} fit_i} \tag{4}$$

Different schemes can be used to calculate the probability, depending on the problem to solve. Vega Yon and Muñoz (2017) uses the equation bellow

$$p_i = a \times \frac{fit_i}{\max_i(fit_i)} + b \tag{5}$$

with $a = 0.9$ and $b = 0.1$. Probability values are calculated by using the quality of nectar fit_i and normalized by dividing maximum fit_i.

We would adopt the first approach as the choice of the parameter a and b in the second equation is quite arbitrary. Under the above a and b values, when the value of nectar is (approximately) the same for all the food source, the probability of going to each food source is (nearly) 1, then the onlooker bees are not very different from the employed bees. Instead, using the ratio of current nectar to the total nectar gains additional randomness in such a situation, while on average the times of selecting each food source is the same as using the other method.

In our setting, the number of onlooker bees is the same as the employed bee or the number of food sources SN as in basic ABC algorithm. Each onlooker bees selects a solution as the base solution, finds a new solution like the employed bee using equation (3), and choose whether to switch solution using the same greedy approach.

3.6 The Scout Bee

The last two phases focus on the local search for a solution. The scout bee component prevents the algorithm from being stuck in a local optimum. After the employed bee and the onlooker bee phase, whether the food source has been moved by one of the bees has been recorded. If a food source has not been improved (moved) up to some certain number of iteration, called the limit, it is abandoned and a new solution would be found in the scout bee phase. Like in the basic ABC algorithm, the number of scout bee is set to be 1, meaning in each iteration, only 1 of the food source will be abandoned and replaces if multiple food sources research the limit. The scout in general randomly generate a solution in the solution space using the following equation

$$z_i^j = z_{min}^j + \delta_i^j (z_{max}^j - z_{min}^j) \tag{6}$$

where z_i is the abandoned source and $j \in 1, 2, \ldots, D$ is the index of dimension. δ_i^j is simply generated from a uniform distribution with bound 0 and 1. In other words, it is generated from a uniform distribution with lower bound z_{min} and upper bound z_{max}. However, in the context of clustering, we are still sampling from the data points, which can limit the range that the scout bee searches. We still impose the constraint where the number of data points in a cluster needs to be more than $\frac{n}{2k}$. The simulation will be redone if not satisfied, so if the range it searches is too wide it may not get a feasible solution at any time soon.

Table 1: Performance measures using the initial parameters for the training data

Data	Dimension	Final Objective Value	Number of Iteration	Best solution was first found
12_ds4c2sc.csv	2	0.7022971	679	480
18_2d-no0.csv	2	0.6223649	548	349
18_s-set1.csv	2	0.5209045	537	338
3_square2.csv	2	0.6159423	277	78
4_ds3c3sc.csv	2	0.6676816	309	110
4_tetra.csv	3	0.9277845	351	152
5_cure-t1.csv	2	0.5537723	339	140
7_aggregation.csv	2	0.6595904	290	91
7_complex.csv	2	0.6751533	776	577
9_diamond.csv	2	0.6264695	579	380

3.7 Evaluation and Parameters for Initial Testing

In each iteration, the best solution in terms of the DB index will be recorded and updated if the new solution is better than the current global solution. The algorithm will stop once it reaches the maximum number of cycles or the number of unchanged iteration reaches the criteria.

See Akay and Dervis Karaboga (2009) for a full discussion on the parameter tuning on the ABC algorithm. Based on their argument, the ABC algorithm can provide a consistent robust result with different swarm size. The minimum swarm size they use in testing is 50. We decrease the value to 20 to try to decrease the time it needs to run in each iteration. We set the maximum number of iteration to be 1000 and the maximum number of unchanged iteration to be 200, 20% of the maximum number of iteration, to achieve a reasonable running time. We use $k \times D$ as the value of the limit, as the number of iterations required for a bee to conduct a decent local search is considered to be related to the number of dimension and the number of clusters. When the number of dimension is large or when the number of cluster is large, it is harder to find a better solution so we increase the limit accordingly.

3.8 Relaxation in High Dimensional Data

For the constraint we imposed on the clustering problem that each cluster must include to a certain amount of data points sometimes is hard to satisfy, especially in the initial food sources simulation, even if we are starting with sampling form existing data point. Therefore, we set the following relaxation in each attempt to find a feasible solution. In those situations, the solution produced using the ABC algorithm may not admits the minimum cluster size constraint.

1. For the initial solution input, a maximum of 4,000 simulations is tested against the constraint.
2. For the simulation for food sources, a maximum of 2,500 simulations is tested against the constraint
3. In the scout bee phase, a maximum of 2,000 new solutions is tested against the constraint
4. If the search for the initial solution or the initial food sources admits the maximum simulations, we relax the constraint to that each cluster must include $\frac{n}{10k}$ data points.

4 Implementation

4.1 Implementation on the training instances

Table 1 gives the performance measures of the algorithm on the training data set. Note the number of iteration and the iteration the best solution was first found do not include the number of simulations that the algorithm needs to produce feasible initial food sources and valid new solution in the scout bee phase.

The `18_s-set1` data set is the one data set that research the condition to relax the constraint that each cluster must have $\frac{n}{2k}$ data points for it has 18 clusters and the sample size is 5,000, but in terms of the final

objective value, it has the best value of DB index among all the data set. The number of iteration it requires to find the optimal solution is not bad compared to other data set with a large number of cluster.

Some of the data sets that require the least time including `3_square2` and `7_aggregation`. For `3_square2` the result is not surprising for it is a two-dimensional data with only 3 clusters. The result for `7_aggregation` is doubtfull for its 7 clusters, and by looking at the difference between the total number of iteration and the time it first admits the final solution, we can say that the algorithm stops because the number of the unchanged global solution has reached 200. If we change the parameters it may improve further with more iterations.

The highest objective function belongs to `4_tetra`, which may because of its extra dimension. The number of iteration is in the middle of the distribution. In conclusion, if the final objective value is not the only measure of performance, it is quite hard to comment on the performance without seeing the plots visually.

4.2 Parameter tuning

We use a subset of the training data sets to tune our parameters. The data sets we choose are `3_square2` and `4_tetra`, for the numbers of clusters of them are relatively small so it requires less time to run the algorithm. We include `4_tetra` to investigate the power of parameters with a different number of dimensions.

The parameters we choose to evaluate and select are the limit (the number of iterations that each food source can stay the same before being abandoned), the maximum cycle number (MCN) and the maximum number of unchanged iteration (MNU). The number of food sources SN is well studied in Akay and Dervis Karaboga (2009). They argue that the algorithm is robust with different values of SN in general.

We implement a coordinate descent approach where we try different values of one parameter and keep other parameters fixed at the same time. We firstly keep the limit fixed as the current setting of the limit changes with different data set, which more likely to remain robust. We also keep MCN as 1000 since from the results above no data set reaches MCN of 1,000, but quite a few data sets stop when the number of the unchanged iteration reaches 200. We acknowledge that there might be a correlation issue where the effects of parameters are not independent with each other especially when MCN and the max number of unchanged iteration control the stopping of the algorithm together.

We then report the different values we tried in the Appendix. We start with MCN of 1000 and a limit of $SN \times D$ and increase MNU see if additional iteration can improve the optimization. The results for the two data sets are reported in Table 5 and Table 6. If we simply look at the final objective value, then increasing MNU would always increase the performance. Because the MCN and MNU control the stopping condition, it would be ideal if we can find a balance to make the number of iteration close to the maximum number, not wasting any unnecessary runs. For data `4_tetra` the algorithm always stops because of the MNU and for `3_square2` only when the MNU is larger enough to 600 it would reach MCN, so we would aim to decrease MCN instead of MNU considering the time it costs to run more iterations.

We keep MNU at 200 and try different values for MCN. The results are reported in Table 7 ann 8. MCN of 500 is optimal for `3_square2` and 600 for `4_tetra`, so we take the average of 550, but also keep in mind that the number of clusters for these two data set is relatively small. Then we try different value for the limit in Table 9 ann 10. The good values of the limit range between 30 and 50, we take an average of 40. Note the number of iteration reaches the maximum number multiple times, so we decided to increase MCN to 700 to accommodate high dimensional data and data with more clusters.

Table 2: Performance on data square2 and tetra with limit of 40, MNU of 200 and MCN of 550

	3_square2	4_tetra
Final Objective Value	0.6135	0.916
Number of Iteration	229.0000	565.000
Best solution was first found	30.0000	366.000

We report the performance of the two data set in Table 2. Both of the data sets present slightly improvement over the final objective value. The increase in the number of iteration for `4_tetra` may due to the increase in

Table 3: Performance measures using the initial parameters for the training data

Data	Dimension	Final Objective Value	Number of Iteration	Best solution was first found
12_ds4c2sc.csv	2	0.6765280	658	459
18_2d-no0.csv	2	0.6260346	432	233
18_s-set1.csv	2	0.5189548	576	377
3_square2.csv	2	0.6045702	573	374
4_ds3c3sc.csv	2	0.6397439	615	416
4_tetra.csv	3	0.8990198	647	448
5_cure-t1.csv	2	0.5514328	390	191
7_aggregation.csv	2	0.6202504	526	327
7_complex.csv	2	0.6624554	607	408
9_diamond.csv	2	0.6001826	700	658

Table 4: The Best, Average, Worest objective value for the test data, 10 runs each

	ds2c2sc	s-set1	D	2sp2globb	smile	2d-no4	square4	cure-t2	ds4c2sc	complex
No.Clusters	12.0000	14.0000	26.0000	3.0000	4.0000	5.0000	5.0000	6.0000	6.0000	9.0000
Best	0.6227	0.4665	0.5988	0.3705	0.6064	0.7606	0.7109	0.6160	0.6037	0.6770
Average	0.6391	0.4722	0.6385	0.3705	0.6103	0.8097	0.7230	0.6303	0.6062	0.6830
Worest	0.6572	0.4914	0.6749	0.3705	0.6186	0.8393	0.7364	0.6617	0.6093	0.6898

the limit from 12 to 40. As limit goes up, the global search of the scout bees is triggered less frequently, which gives the employed bees and the onlooker bees more time to implement local search. It shows by adjusting the limit, the intensification and diversification are more balanced: the neighbourhood of each solution is searched more thoroughly without being abandoned too early.

4.3 Testing new parameters

We evaluate the performance of our new parameters on the training set and report the results in 3. All data sets but `18_2d-no0` showed improvement over objective value. The final objective value for `18_2d-no0` is only worsened by 0.0037. The improvement over other data sets ranges from 0.002 to 0.04. It is not surprising to see that the two data sets with the highest number of clusters appear to be either not improving (`18_2d-no0`), or least improved by 0.002 (`18_s-set1`). Other than the reason that they have larger numbers of clusters, we only tune our parameters based on the data sets with a small number of clusters. The best parameters used for data sets with many clusters may need to be considered different in this case.

The number of iteration that the best solution was first found for most of the data sets increases, likely due to the increased limit. As we discussed before, a larger limit gives room to implement better local search, emphasising intensification more. After adjusting MCN, the number of iterations used by each data set is very close to MCN, which is more efficient than before.

4.4 Test data set

The performances for the test data set are reported in Table 4. In general, the algorithm performs more stably on the data set with a small number of clusters. The most typical one is `2sp2globb` with 3 clusters. the results are indistinguishable with 4 decimal places. For `D` with 26 clusters, the final objective value ranges from 0.60 to 0.67. There are some exceptions, indicating the performance is largely related to the pattern of the data.

Note because of the huge number of clusters, simulating initial solution and initial food sources for data `D` is very hard. All ten runs triggered the relaxation condition so we have to decrease the minimum cluster size for the problem to be solvable within a reasonable time constraint.

4.5 Comments on the plots

The plots are presented in the Appendix. Most of them seem reasonable. The clustering for `2sp2globb` in Figure 5 shows perfect classification. The distribution of the points in that data set is intrinsically easy to identify. Following problems are presented in some other plots.

1. In Figure 3, 4, 7, 8, 9, and 10, there might still be some room to improve on the clustering if the number of clusters specified fits the pattern more. For example, in Figure 3, by looking at the pattern of the colours we can say one extra centroid would make a difference.
2. In Figure 7, 8, and 9, for example, there are some points on the edge of a clear cluster is identified to be in another cluster. This is because the allocation scheme we are using is closest allocation: allocation depends on the distance of the point to the centroids instead of pattens. We could improve the algorithm if take the pattern into consideration in allocation.
3. In Figure 2, 6, and 11, the patterns are more complicated than what closed allocation can detect. For example, in Figure 6 the `smile`, the cluster human can determine could be three clusters inside a cycle, which itself is a cluster. But using closest allocation and the DB index (even using other clustering algorithms), we cannot detect clusters within a cluster.
4. Note many of the identified centroids are not in the centre of the cluster. This is because the DB index tries to increase the distance between centroids and decrease within-cluster distance at the same time. Moving centroids away from each other (and away from the centre of the cluster) will improve the DB index.

4.6 Conclusion

The ABC algorithm is very flexible in the sense that we can modify it to fit different problems and objective function. It is easy to adjust between intensification and diversification by changing the behaviour properties of the bees and modifying the parameters without changing the general idea of mimicking honey bee colony. It can escape local optima quite easily using the solution abandon mechanism, and the parameters directly control the time it needs to run, producing reliable results relatively quickly.

Because of the flexibility of the algorithm, the settings of the algorithm needs to be taken carefully. For example, the function to find neighbours, the way to calculate onlooker bee probabilities, and the number of scout bees are not considered as parameters we need to train in the above discussion, but they can be and needs to be modified with respect to specific problems, which requires more testing and adjusting.

The difference between using the ABC algorithm to solve the clustering problem and to solve the general numerical problem is mostly about the different objective function, which is the same for other algorithms. The ABC algorithm is easily modified to solve the clustering problem, but it is not specifically designed for clustering. The design of the ABC algorithm does not consider the features in the clustering problem, unlike the algorithm that is specifically for clustering problem such as k-means. It treats the clustering problem using the same optimization strategy as in a numerical problem which may not be the most efficient approach. Furthermore, for the same reason, the pattern of the resulting clusters from the ABC algorithm depends entirely on the form of the objective function. It cannot incorporate complicated patterns if the objective function cannot be easily calculated.

The objective function we are using in the above practice is the DB index, which is relatively complicated to evaluate. Combined with the ABC algorithm that in each iteration each bee needs to evaluate the solution using the DB index, it will decrease the efficiency. Using the DB index in the clustering problem also increases the probability that the centroids are not in the centre of the clusters, which might be desired in some cases but in general, a different objective function may help with the process of finding patterns.

With the objective function being the DB index, the requirement of minimum cluster size helps with identifying new solutions as the DB index itself does not guarantee a proper position of the centroid: the centroid may be away from the main cluster and a small DB index can still be found. On the other hand, the minimum cluster size would greatly delay the whole algorithm, because even at the start of the process, an optimization

algorithm with a random component like ABC can struggle to find a feasible solution according to the minimum cluster size without further assumption and treatment. The minimum cluster size is needed in this scenario, but not the most efficient approach compared to a different set of objective function and constraints.

5 Appendix

Table 5: Performance on data square2 using different MNU with fixed MCN of 1000 and limit of SN*D

	MNU 300	MNU 400	MNU 500	MNU 600	MNU 700
Final Objective Value	0.6107	0.606	0.6059	0.6018	0.6067
Number of Iteration	645.0000	902.000	539.0000	1000.0000	1000.0000
Best solution was first found	346.0000	503.000	40.0000	657.0000	839.0000

Table 6: Performance on data tetra using different MNU with fixed MCN of 1000 and limit of SN*D

	MNU 300	MNU 400	MNU 500	MNU 600	MNU 700
Final Objective Value	0.8812	0.9251	0.9083	0.8864	0.9382
Number of Iteration	379.0000	783.0000	690.0000	990.0000	791.0000
Best solution was first found	80.0000	384.0000	191.0000	391.0000	92.0000

Table 7: Performance on data square2 using different MCN with fixed MNU of 200 and limit of SN*D

	MCN 300	MCN 400	MCN 500	MCN 600	MCN 700
Final Objective Value	0.6083	0.6183	0.6001	0.6163	0.6218
Number of Iteration	300.0000	258.0000	448.0000	495.0000	300.0000
Best solution was first found	174.0000	59.0000	249.0000	296.0000	101.0000

Table 8: Performance on data tetra using different MCN with fixed MNU of 200 and limit of SN*D

	MCN 300	MCN 400	MCN 500	MCN 600	MCN 700
Final Objective Value	0.9258	0.9397	0.9241	0.911	0.9458
Number of Iteration	251.0000	301.0000	361.0000	370.000	246.0000
Best solution was first found	52.0000	102.0000	162.0000	171.000	47.0000

Table 9: Performance on data square2 using different limit with fixed MNU of 200 and MCN of 550

	Limit 20	Limit 30	Limit 40	Limit 50	Limit 60	Limit 70
Final Objective Value	0.6132	0.6034	0.6054	0.6028	0.6084	0.6061
Number of Iteration	213.0000	303.0000	261.0000	401.0000	550.0000	474.0000
Best solution was first found	14.0000	104.0000	62.0000	202.0000	484.0000	275.0000

Table 10: Performance on data tetra using different limit with fixed MNU of 200 and MCN of 550

	Limit 20	Limit 30	Limit 40	Limit 50	Limit 60	Limit 70
Final Objective Value	0.9508	0.9126	0.9145	0.9358	0.9336	0.9438
Number of Iteration	496.0000	543.0000	550.0000	550.0000	416.0000	429.0000
Best solution was first found	297.0000	344.0000	538.0000	536.0000	217.0000	230.0000

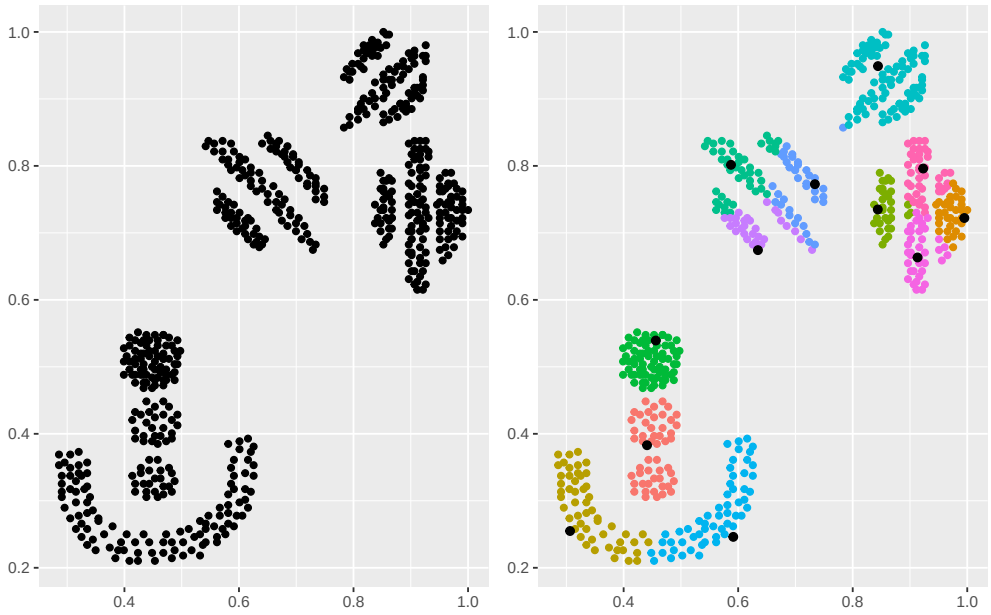

Figure 2: Clustering of ds2c2sc

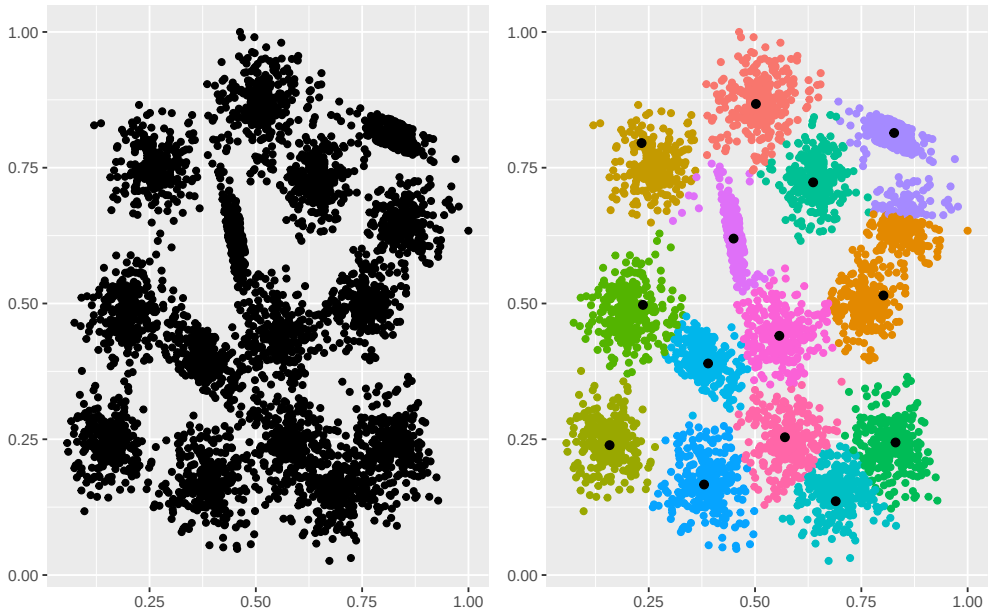

Figure 3: Clustering of s-set1

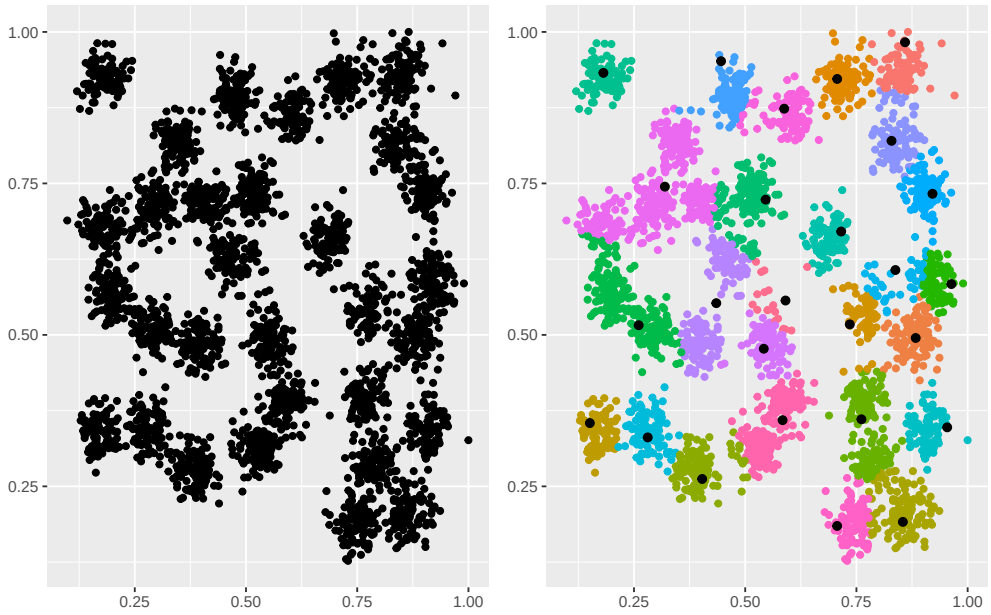

Figure 4: Clustering of D

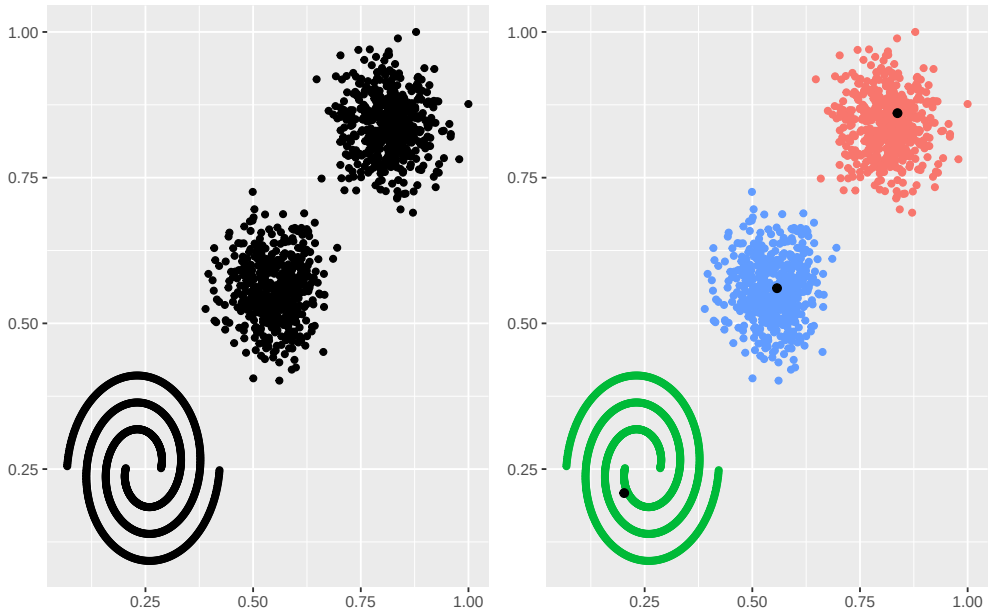

Figure 5: Clustering of 2sp2globb

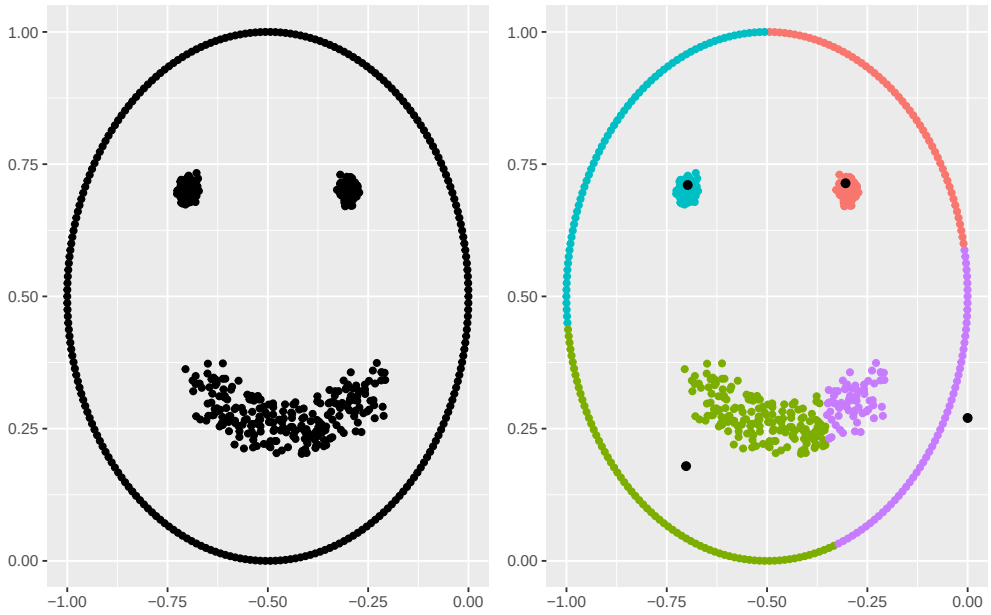

Figure 6: Clustering of smile

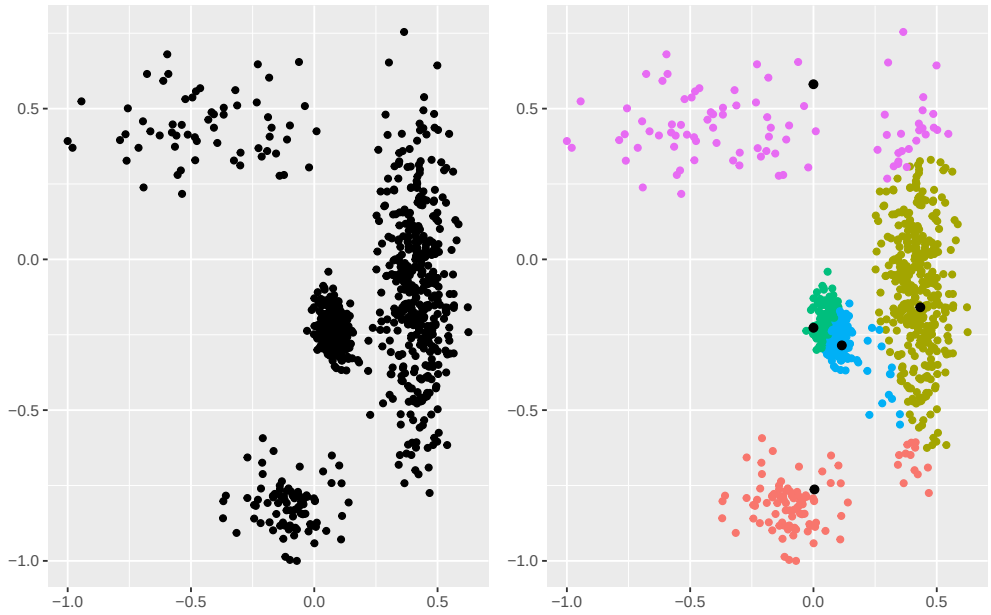

Figure 7: Clustering of 2d-no4

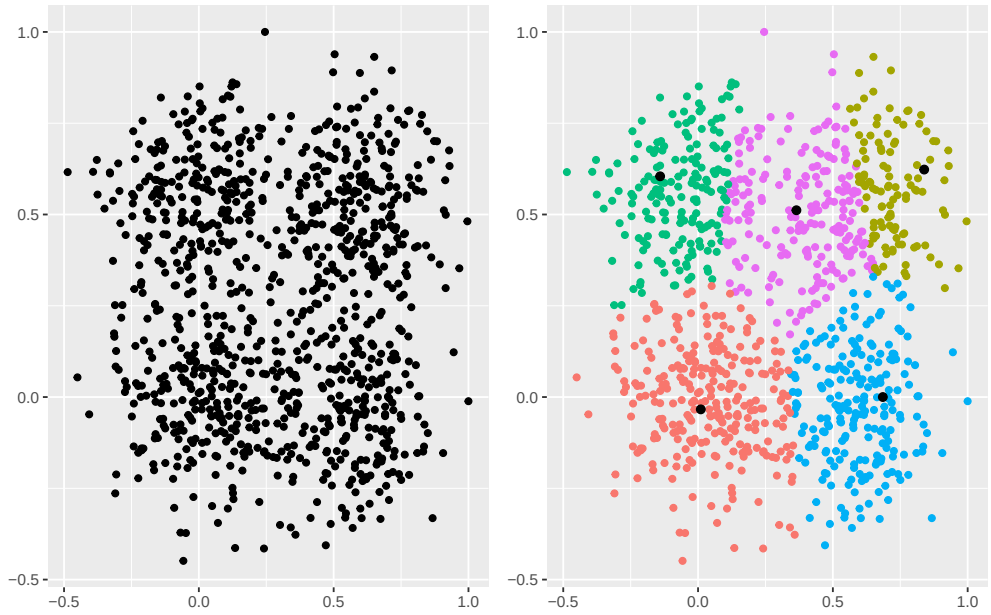

Figure 8: Clustering of square4

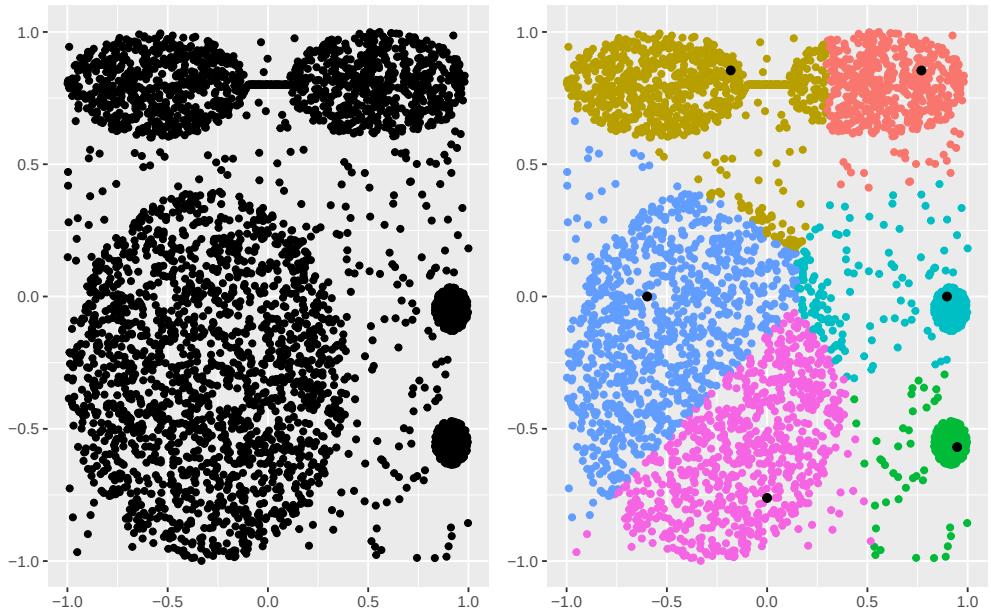

Figure 9: Clustering of cure-t2

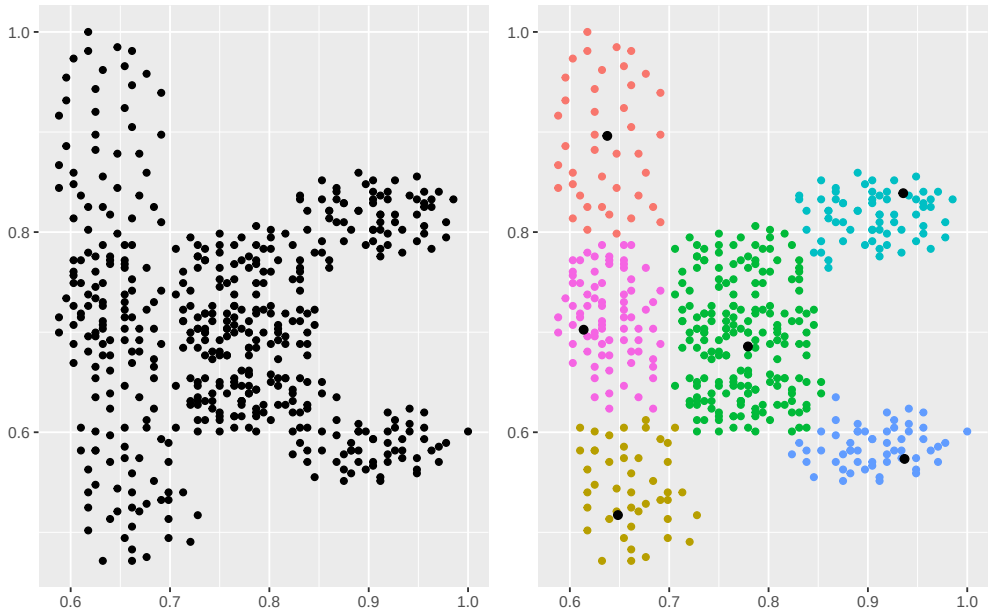

Figure 10: Clustering of ds4c2sc

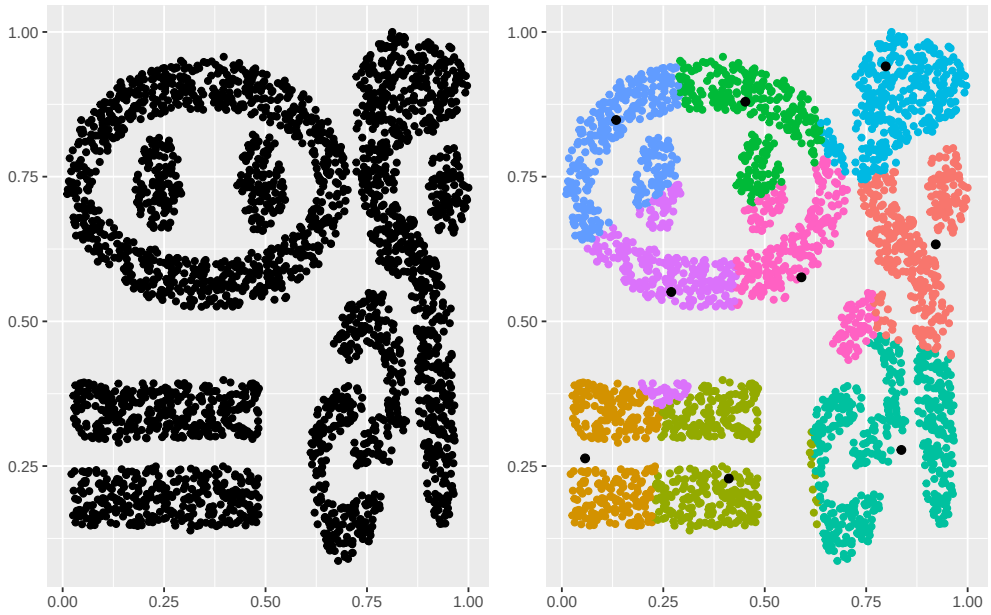

Figure 11: Clustering of complex

References

Akay, Bahriye and Dervis Karaboga (2009). "Parameter tuning for the artificial bee colony algorithm". In: *International Conference on Computational Collective Intelligence*. Springer, pp. 608–619.

Karaboga, D (2005). "An idea based on honey bee swarm for numerical optimization". In: URL: https://pdfs.semanticscholar.org/015d/f4d97ed1f541752842c49d12e429a785460b.pdf.

Karaboga, Dervis and Celal Ozturk (2011). "A novel clustering approach: Artificial Bee Colony (ABC) algorithm". In: *Applied soft computing* 11.1, pp. 652–657.

Pacurib, Jaysonne A, Glaiza Mae M Seno, and John Paul T Yusiong (2009). "Solving sudoku puzzles using improved artificial bee colony algorithm". In: *2009 fourth international conference on innovative computing, information and control (ICICIC)*. IEEE, pp. 885–888.

Vega Yon, George and Enyelbert Muñoz (2017). *ABCoptim: An implementation of the Artificial Bee Colony (ABC) Algorithm*. R package version 0.15.0. URL: https://github.com/gvegayon/ABCoptim.