Journal of Al-Azhar University Engineering Sector

Vol. 18, No. 69, October 2023, 884 - 899

# A Metaheuristic to Solve the Capacitated Clustering Problem

**Banan Nasser AlTuwaim**

Department of Computer Science, College of Computer & Information Sciences, Al Imam Mohammad Ibn Saud Islamic University, Riyadh, Saudi Arabia

*Correspondence: banan2746@gmail.com

**Citation:**

B. N. AlTuwaim, " A Metaheuristic to Solve the Capacitated Clustering Problem", Journal of Al-Azhar University Engineering Sector, vol. 18, pp. 884 - 899, 2023.

**Received: 9 August 2023**

**Accepted: 23 September 2023**

DoI:10.21608/auej.2023.235083.1420



## ABSTRACT

One of the most important combinatorial optimisation problems (COPs) is the capacitated clustering problem (CCP). This sort of problem involves dividing a set of nodes into a predefined number of clusters within specific limits (upper and lower); for each node in the cluster, a pair of nodes is defined that has a benefit value, and the total values of these benefits in the cluster must be maximised.

The CCP is of a non-deterministic polynomial-time hard (NP-hard) nature. Such problems cannot be practically solved by exact optimisation algorithms, so we must choose one of the metaheuristic algorithms to solve them. The CCP has many applications in a variety of domains.

This paper aims to design a metaheuristic method that solves the CCP, with an artificial bee colony (ABC) as the metaheuristic algorithm.

The effectiveness of the proposed ABC based algorithm was confirmed through several computational experiments. The results demonstrate that the proposed algorithm produced competitive outcomes compared to the state-of-the-art metaheuristics developed for the CCP.

**KEYWORDS**: Capacitated clustering problem (CCP), Local search, Neighbouring operators, Constraints handling, Artificial bee colony (ABC).

# خوارزمية استرشادية لحل مشكلة التجميع مقيد السعة

بنان ناصر آل تويم*

قسم علوم الحاسب، كلية علوم الحاسب والمعلومات، جامعة الإمام محمد بن سعود الإسلامية، الرياض، المملكة العربية السعودية

**البريد الاليكتروني للباحث الرئيسي*:** banan2746@gmail.com

## الملخص

مشكلة التجميع مقيد السعه واحد من اهم مسائل الاستمثال التوافقى ، هذا النوع من المشكلات ترتكز على تقسيم مجموعة من الكائنات الى مجموعات منفصلة بحيث تكون الاوزان الاجمالية لهذه الكائنات ضمن حدود معينة (حد علوى وسفلى) ، وفى نفس الوقت يتم تحقيق اقصى قدر من اجمالى اوزان الاضلاع بين ازواج الكائنات فى نفس المجموعة.

هذه المسشكلة تعد مسالة كثيرة الحدود وغير قطعية ومعقدة وبالتالى فان خوارزميات التحسن الدقيقة غير عملية لحل مثل هذه المشكلات لذلك توجب علينا اختيار احدى خوارزميات الاسترشاد. وتوجد العديد من التطبيقات لحل مشكلة التجميع مقيد السعة والتى يمكن تطبيقها فى مجالات ومشكلات مختلفة.

إن الهدف الرئيسى من هذا البحث هو تصميم خوارزمية استرشادية فعالة لحل مشكلة التجميع مقيد السعة باستخدام خوارزمية مجتمع النحل الاصطناعى كخوارزمية استرشادية.

خوازمية البحث المحلى لها اثر ملموس على قيمة دالة الهدف، وهذا يحدث عند مقارنة النتائج الحسابية مع بعض نتائج الدراسات السابقة، وهذة المقارنة تبين ان خوارزمية النحل الاصطناعى هى خوارزمية منافسة لخوارزميات الاسترشاد الاخرى بخصوص حل مشكلة التجميع مقيد السعه.

**الكلمات المفتاحية :** مشكلة التجمع مقيد السعه، البحث المحلي، العوامل المجاورة، التعامل مع القيود، مستعمرة النحل الاصطناعية.

# 1. INTRODUCTION

The capacitated clustering problem (CCP) is a non-deterministic polynomial-time hard (NP-hard) combinatorial optimisation problem (COP). The process involves being divided into a set of nodes into a specific number of clusters, with certain constraints, to maximize the total benefits of the connected pairs of nodes (edges). The edges in the cluster have specific benefits (distances). In addition, each node has a specific weight, and the clusters have specific upper and lower weight limits [1].

The objective function sums all the pair benefits in each cluster. In the COP, the aim is to identify the best solution from a finite set of solutions, regardless of the time it takes to do so. After an in-depth study and investigation of COPs, we have chosen the CCP because it has many applications in different domains, like mail delivery, facility location [2]. Also, the CCP has been applied to computational biology, garbage collection zone design [3]. These applications have been applied in network design, sales force territory design, pattern recognition, manufacturing, habitat classification, statistical data analysis [2], telecommunications [3], mobility networks, and vehicle routing, very large-scale integration (VLSI) design [4].

CCP is a well-known problem, Deng & Bard (2011) mention that the main problem that motivates them to investigate the CCP is their association with the US Postal Service, which might facilitate planners at the mail processing and distribution centers. The US Postal Service pursues to recurrent the task of designing regions to justify the bulk movement of mail which is powered by industrial vehicles [2].

Other applications of CCP is computational biology, specifically the sibling reconstruction problem (SRP) [5]. The SRP is one of the significant problems in genetic biology [5]. In this form, they proposed a randomized greedy optimization algorithm. The algorithm has two stages: construction and enhancement. The constriction stage consists of a randomized greedy perturbation approach and in the enhancement stage, they use a two-phases of local search with a memory method. The proposed model shows that it is an effective model by applying biological data sets and compression with the literature.

Martínez- Gavara et al. (2015) mathematically formulated the CCP as follows: ‘given a graph G = (V, E) where V is a set of nodes n and E is a set of edges, let wi ≥ 0 be the weight of node i ∈ V and let cij be the benefit of edge (i, j) ∈ E. The CCP is the process of partitioning V into P clusters in a way that the total weights of the objects in each cluster are within specific capacity limits, L and U, and the total benefits between the pairs of nodes in the same cluster are maximized. The CCP can be formulated as a quadratic integer program with binary variables Xik that gives the value 1 if the node i is found in the cluster k and otherwise gives the value 0.

Therefore, the CCP can be defined as:

$$\text{Maximise } \sum_{k=1}^{p}\sum_{i=1}^{n-1}\sum_{j>i}^{n} c_{ij}x_{ik}x_{jk} \tag{1.1}$$

$$\text{Subject to } \sum_{k=1}^{p} x_{ik} = 1 \quad i = 1,2,\dots,n \tag{1.2}$$

$$L \leq \sum_{i=1}^{n} w_i x_{ik} \leq U \; k = 1,2,\dots,p \tag{1.3}$$

$$x_{ik} \in \{0,1\} \quad i = 1,\dots,n \; k = 1,\dots,p \tag{1.4}$$

The objective function of the CCP is expressed in (1.1) and it leads us to the total benefits of all node pairs belonging to the related cluster being maximized. The initial set of constraints (1.2) guarantees the selection of all nodes to a cluster. The following set of constraints (1.3) guarantees that the total weights of the pairs of nodes that refer to a similar cluster are between L and U.’ Variable definitions are given in (1.4) [1].

This paper aims to design and implement the ABC algorithm for solving the CCP. To our knowledge, there has been no attempt to investigate the application of the ABC to the CCP. We will also study the outcomes of the ABC algorithm with the outcomes of other algorithms for the same problem and datasets.

The applications of the CCP have been investigated by many researchers. We will mention the main research according to the most relative to my paper, I will state it according to historical appearance. Firstly, Mulvey and Beck [6] expanded the uncapacitated clustered problem to the CCP by adding capacity constraints on each cluster. They solve the CCP problem by a hybrid heuristic-subgradient and primal heuristic methods. Osman & Christofides [7] proposed a hybrid algorithm of the TS and SA with an elementary construction heuristic of a λ-interchange generation mechanism. Ahmadi and Osman [8], merged GRASP and adaptive memory programming (AMP) into their framework. Franga, Zuben and Castro [9] proposed an adjusted MAX–MIN ant system (MMAS), which was related to evolutionary algorithms. Likewise, Scheuerer and Wendolsky [10], whose proposed population-based algorithm, known as the scatter search-based heuristic approach. After that, Deng and Bard [2] suggested to couple GRASP with path relinking (PR). They used an HWE algorithm and a constrained minimum cut algorithm for the initial solution. Then, they used the cyclic neighbourhood search (CNS), VND, and randomised VND (RVND) methods for the LS. Darani, Ahmadi, Eskandari and Yousefikhoshbakht [11] also later proposed a hybrid 3 staged meta-heuristic algorithm (HTMA), containing a sweep algorithm for the initial solution phase, and an ant colony optimisation (ACO) with two LSs: insert exchange (IE) and swap exchange (SE) algorithms, for improving the solution to solve the CCP. Lai and Hao [12] proposed an effective IVNS algorithm for resolving the CCP. The IVNS integrated an extended VND method and a randomised shake procedure to investigate the search space. Brimberg, Mladenović, Todosijević, and Urošević [13] proposed three neighbourhood methods and used them within a sequential VND heuristic. The three neighbourhood operators were insertion, swap and two out−one in 2–1 exchange operators. The same authors extended their work and proposed two VNS-based heuristics: the first was the same as that which had previously been used by [14] and [15], and the second was a skewed VNS that added acceptance criteria to grant a skewed move to ideal regions of the solution in the search space. Zhou, Benlic, Wu and Hao [4] proposed a memetic algorithm (MA) that merged TS algorithms, which explore the feasible and unfeasible regions of the search space (denoted as FITS), with a specified cluster-based crossover technique.

The ABC algorithm, proposed by Derviş Karaboğa in 2005 [16], is an optimisation algorithm that builds on the behaviour of bees foraging for honey in a swarm. In the ABC, which is a population-based algorithm, the food source illustrates the feasible solutions to the optimisation problem and the amount of nectar in a food source presents the quality (fitness) of the related solution [17]. The ABC has been applied successfully to a variety of problems and has recently been used by many researchers.

The algorithm categorizes the bee swarm into three jobs: employee, onlooker and scout bees. Employee bees are accountable for collecting information about food source locations and their quality and saving this information in memory. They are also responsible for the LS implementation, using neighbouring locations of the food source to explore the preferred places for foods in the neighbouring zones of the current value [16].

Onlooker bees are accountable for determining which food source is the best. The onlooker's decision is based on the previous information that was gathered by the employee bees, and they are responsible for discovering the global optimum [16]. Scout bees are responsible for discovering new areas that have not yet been discovered by the employee bees, using a random search. Scout bees avoid the search process for fear of getting trapped in the local optimum [16]. The colony is divided in half between the onlooker and employed bees. The quality of the solutions is evaluated through the objective function value, which is associated with each solution. The total number of onlooker or employed bees is equivalent to the total number of solutions in the population [17].

The ABC algorithm was originally proposed for multi-modal and multi-variable continuous optimisation problems and numerical optimisation problems [18]. Many variations of the ABC algorithm have later been practiced to many types of optimisation problems, like symbolic regressions [19], the quadratic minimum spanning tree problem [20], the design and operation of assembly lines [21], the leaf-constrained minimum spanning tree problem [22], image segmentation [23], binary optimisation problems [24], recurrent neural network designs [25], constrained optimisation problems [26], the development of routing protocols for wireless sensor networks [27], [28], multi-objective optimisation problems [29], [30], as cited in [17], and the p-median problem [31], [32], uncapacitated facility location problems [33], [34], the knapsack problem (KP) [35], [36], the vehicle routing problem (VRP) [37], [38], the job shop scheduling problem (JSSP) [39], [40], the TSP [41], [42] and clustering problems [43], [44], [45], as cited in [46]. A complete review of the ABC algorithms and hybrid approaches and applications can be found in [47] and [48], and a comparative study between the ABC and the GA and PSO, as well as different evaluation algorithms can be found in [18]. The wide use of the ABC algorithm is attributable to its flexibility, efficiency, robustness and simplicity. The ABC algorithm is an example of how a natural process can be modelled to solve optimisation problems [17], [16].

One of the related problems to the CCP is the maximally diverse grouping problem (MDGP). Martínez-Gavara et al. [1] stated that the MDGP could be considered a unique case of the CCP or a related problem to which the whole of the nodes has a weight of one unit. The MDGP involves grouping a collection of objects so that the diversity between the objects in each group is maximized. The diversity between the objects within the group can be measured as the total distances between the pairs of objects. The problem aims to maximize the total diversity of all groups when the size of each group is within a specific limit [49].

As was mentioned before, the MDGP is a unique case for which wi = 1 for all nodes i, and the distance between each pair of nodes (i, j) is the benefit cij. From the CCP formulation above, Gallego et al. [49] mathematically formulated the MDGP as follows:

$$\text{Maximise } \sum_{g=1}^{G}\sum_{i=1}^{M-1}\sum_{j>i}^{M} d_{ij}x_{ig}x_{jg} \tag{1.5}$$

$$\text{Subject to } \sum_{g=1}^{G} x_{ig} = 1, \quad i = 1,2,\dots,M \tag{1.6}$$

$$\sum_{i=1}^{M} x_{ig} = S, \quad g = 1,2,\dots,G \tag{1.7}$$

$$x_{ig} \in \{0,1\} \quad i = 1,\dots,M \; g = 1,\dots,G \tag{1.8}$$

Rodriguez et al. [17] suggested an ABC algorithm for the MDGP problem. The proposed framework starts with an initial solution that is produced by greedy constructive method, then its employees three different techniques for generating neighborhood. The local search of that algorithm consists of move and swap operators. The proposed framework has achieved a significant balance among global and local searches and the experimental outcomes show that the suggested algorithm is competitive with the current literature of algorithms which address the MDG problem.

## 2. The application of ABC algorithm to the CCP

In this part, we will explain the different components of the ABC algorithm that were designed to obtain high-quality solutions for the CCP. Our algorithm was inspired by an algorithm designed by Rodriguez et al. [17] for the MDGP problem. Figure 1 depicts the ABC algorithm. We will describe the general framework first and then describe each main component of the framework in detail in the following parts.

The ABC framework requires six input values and five parameters: instance I represents the dataset instance, $t_{max}$ represents the computation time limit, limit represents the maximum number of iterations before the decision to discard, NP presents the number of populations or food sources, $p_{ls}$ presents the probability of applying the LS to the solutions. $q_s$ presents the number of queue swap operators. The algorithm returned the ideal solution found so far.

```
   Input  : Instance I, t_max, limit, NP, p_ls, q_s
   Output: optimal solution of ABC S*
 1 for i=1 to NP do
 2 |  S_i ← initialSolution ();
 3 |  if U(0,1) < p_ls then
 4 |  |  S_i ← LS (S_i);
 5 |  end if
 6 end for
 7 while t_max not met do
 8 |  for i = 1 < NP do
 9 |  |  E ← generateNeighbouring(S_i);
10 |  |  if U(0,1) < p_ls then
11 |  |  |  E ← LS(E);
12 |  |  end if
13 |  |  if E is better than S_i, // Employed Bees
14 |  |   then
15 |  |  |  S_i ← E;
16 |  |  |  if S_i = BKS, then
17 |  |  |  |  Return best solution S*;
18 |  |  |  |  break;
19 |  |  |  end if
20 |  |  end if
21 |  end for
22 |  for i = 1 < NP do
23 |  |  j ← binaryTournament (1, ..., NP);
24 |  |  O ← generateNeighbouring(S_j);
25 |  |  if U(0,1) < p_ls then
26 |  |  |  O ← LS(O);
27 |  |  end if
28 |  |  if O is better than S_j, // Onlooker Bees
29 |  |   then
30 |  |  |  S_j ← O;
31 |  |  |  if S_j = BKS, then
32 |  |  |  |  Return best solution S*;
33 |  |  |  |  break;
34 |  |  |  end if
35 |  |  end if
36 |  end for
37 |  for i = 1 < NP do
38 |  |  if S_i reach limit then
39 |  |  |  s_i ← initial solution ();
40 |  |  |  if U(0,1) < p_ls then
41 |  |  |  |  S_i ← LS(S_i); // Scout bees
42 |  |  |  end if
43 |  |  end if
44 |  end for
45 |  Return best solution S*;
46 end while
```

Figure 1: Pseudocode of ABC algorithm.

## 2.1 General Framework of the ABC Algorithm to Solve the CCP

Our ABC algorithm is presented in Figure 1. The framework starts with a population of solutions produced by the function *initialSolution ( )* (Line 2). The function *initialSolution ( )* implements a 3-stage procedure to create an initial populations solution for the CCP. Next, the following phases can be repeated until the time limit ($t_{max}$) is reached [17]:

- The employed bee stage generates new solutions for the employed bees through the *generateNeighbouring( )* function (Line 9). This method uses one neighbouring operator, which is based on swaps (see 4.2.3 for details on this function). Next, an LS procedure is applied to the solutions with a probability $p_{ls}$ (Lines 10–11), meaning that not necessarily all of the solutions will be improved by the LS procedure. At the end of this phase, if the solution is improved after the *generateNeighbouring( )*

and LS procedures, it is saved (Line 15). In the situation that the solution value is equal to the value of the BKS, the algorithm will terminate (Lines 16–18) [17].

- The onlooker bee stage generates new solutions for the onlooker bees from the solutions by taking from the method *binaryTournament( )* (Line 23), which randomly chooses two solutions from the populations and selects the better one of the two. Next, the *generateNeighbouring( )* function is implemented on the solutions (Line 24) and the LS procedure is applied to the solutions with a probability $p_{ls}$ (Lines 25–26). At the end of this phase, if the solution is improved, it is saved (Line 30). In the situation that the solution value is equal to the value of the BKS, the algorithm will terminate (Lines 31–33) [17].
- The scout bee stage selects the discarded solutions for the scouts and exchanges them with solutions recently produced by the method *initialSolution( )* (Line 39). The decision to discard is taken based on a parameter value of *limit*; in other words, solutions that have not changed for specific iterations (within a certain *limit*) are excluded. Next, the LS procedure is applied to the solutions with a probability $p_{ls}$ (Lines 40–41) [17].

At the end of the framework and as long the value of the best objective function value does not reach the value of the BKS, the best objective function value is returned as the best solution value. The complexity of the ABC algorithm is $O(n^2)$.The main component functions of the framework are stated in the following sections [17].

### 2.2 Initial Solution Function

The initial solution procedure takes place at the beginning of the framework and in the scout bee phase (Lines 2 and 39). The initial solution function produces a solution using a 3-stage procedure. It first assigns nodes randomly to the clusters, then randomly assigns nodes to each cluster sequentially until the lower bound constraints are met (i.e., assign nodes until the total sum of their weights in the cluster is larger than or equal to $L$). Lastly, it assigns the remaining nodes [13]. The pseudocode of initial solution function is given in Figure 2.

| Function: initial solution<br>Input: instance I<br>Output: feasible initial solution |
|---|
| 1- Randomly select C nodes according to the cluster number and assign one to each cluster.<br>2- Repeat the following steps until $W_p \geq L$ for all clusters.<br>- Randomly select unassigned node n<br>- Assign node n to cluster C if $W_p < L$<br>3- Repeat the following steps until all nodes are assigned.<br>- Randomly select unassigned node n<br>- Assign node n to cluster C if $W_p \leq U$ |

Figure 2: Pseudocode of initial solution.

### 2.3 Generate Neighbouring Function

The generate neighbouring function is used at the beginning of the employed bee stage and the onlooker bee stage (Lines 9 and 24). The function simply implements N2, but this swap operation is a queue swap (multiple swaps), not a single swap. This function performs $q_s$ randomly swaps to attain a neighbouring solution [17]. The value of $q_s$ is determined as a parameter. The pseudocode of suggested Generate Neighbouring function is given in Figure 3.

| Function: *generateNeighbouring ( )*<br>Input: instance I, $q_s$<br>Output: feasible solution |
|---|
| 1- Reap until reach the value of $q_s$<br>- Randomly select a solution.<br>- Do the swap () if the swap Don't violate the node weight $L \leq W_p \leq U$.<br>2- Return the solutions after $q_s$ swapping operation. |

Figure 3: Pseudocode of *Generate Neighbouring*.

### 2.4 LS Function

The LS procedure takes place after production of novel solutions in the initial solution function and scout bee phase (Lines 4 and 41) and generation of neighbouring solutions in the employed bee phase and the onlooker bee phase (Lines 11 and 26). The LS procedure is used with a $p_{ls}$ probability value in order to improve the quality of the solutions [17].

In our framework, LS rejects any infeasible solutions. We depended on the best improvement strategy, and we will refer to the move neighbouring method as N1 and the move and swap neighbouring methods as N1+N2. The pseudocode of suggested LS is given in Figure 4.

| Algorithm: LS |
|---|
| Function: construction solution<br>Input: feasible initial solution<br>Output: optimal solution for the LS |
| 1- Repeat until no neighbouring solutions improve on the current solution.<br>- If a move violates node weight $L \leq W_p \leq U$, reject this move.<br>- Do all possible improving moves, *move ( )* if $L \leq W_p \leq U$<br>- Do all possible improving swaps, *swap ( )* if $L \leq W_p \leq U$<br>2- Return the highest objective function value solution |

Figure 4: Pseudocode of the LS.

As the initial solution had already been produced by the initial solution procedure, we investigated the performance of applying two neighbouring operators (N1+N2) because we found that the integration of such operators had a significant impact on the solution quality in the case of time and objective function value. The move operators searched through all of the possible neighbouring movements that provided an increase in the objective function value. The best or the first movement operation to produce an increase in the fitness function value was made. It executed each possible improving movement until no more improvements were found, thus producing the local optimum. [17]

The swap operator searched through all of the possible neighbouring swaps that provided a rise in the objective function value. The best or first swap operation that produced an increase in the fitness function value was made [17].

The used LS constraint-handling technique will simply reject any infeasible solutions so there would be no violations at all of the constraints.

## 3. Experimental Results and Comparisons

On Amazon Web Services (AWS) infrastructure, the suggested algorithms were coded in Java and executed on a virtual server in Amazon's Elastic Compute Cloud EC2. The hardware specifications for EC2 were c4.8xlarge type, 36 vCPUs and 60 GB of memory with 10 GB of network performance. The instance processors were 2.9 GHz Intel Xeon E5-2666 v3 processors.

To measure the performance of our algorithm, the experiment had a set of ten instances of DB (introduced by Deng and Bard), which were publicly available.[1] The datasets had been used by many researchers in testing other literature of algorithms (e.g., Lai et al. [13], Deng and Bard [2], … etc.). This set of ten DBs was based on the problem introduced by Deng and Bard [2] in the situation of the MDGP, with $n$=82 and $p$=8. However, in the MDGP, the node weights were equal to 1, so Deng and Bard randomly generated the node weights with a uniform distribution $U$ (0,10) to be suitable for the CCP. The set was based on 10 instances with $n$=82, $p$=8, $L$=25 and $U$=75.

### 3.1 Parameter Tuning for the ABC Algorithm

The ABC algorithm has four different control parameters: $p_{ls}$, which represents the probability of applying the LS improvement procedures; $q_s$, which indicates the number of queue swap moves required to obtain a neighbouring solution; and the *limit*, which represents the procedure of abandoning solutions that are not improvements through limited trials and that produces a scout bee phase. The algorithm will also have some fixed parameters that are NPs and that represent the number of produced solutions. The computation time limit before the ABC iteration is terminated is indicated by $t_{max}$.

We exploited the parameter-tuning for the ABC algorithm through various experiments. In all the experiments, the NP value was 20 and the $t_{max}$ value was $n \times 1$ ($n$ represents the number of nodes); this stop condition (time limit) is the same as the stop condition for the benchmark algorithms. The experiments were applied on a set of capacitated clustering problem library (CCPLIB) benchmark instances (1) given in Tables 1 and 2. The tables report the experimental best results, average values and standard deviations of the twenty runs. Besides examining the results of the ABC algorithm configured by the different parameter values, in which the results were also compared to those of the BKS and the TS, GRASP and IVNS. We intended to decide, by tuning the parameters, which configuration could provide a balance of quality, time and maximised objective function.

We built twenty-four ABC configurations by examining two instances to avoid overfitting. These configurations investigated the effects of different values of $P_{ls}$ = {0.5,0.6,0.7,0.8,0.9,1}, $q_s$ = {0.1$n$, 0.2$n$, 0.3$n$} and *limit* = {0.5$n$, $n$, 2$n$}, and the impact of applying the LS1, LS2 and LS3.

The tables provided below outline the performance measures as follow:

1- The best solution value (Best).

2- The average solution value (Average).

3- The average RPD (AvgRPD), which can be calculated as $\left[\frac{BKS-Average\ Solution}{BKS} \times 100\right]$.

4- The standard deviation (SD).

$$SD = \sqrt{\frac{\sum(x-\bar{x})}{n}}$$

SD = standard deviation of objective values of 20 runs

$\sum$ = sum of.

$x$= objective values of solution.

$\bar{x}$= mean of all objective solutions, average of sigma x.

$n$= number of solutions (Times)

5- The average complete computational time (CTime) in seconds.

[1] http://grafo.etsii.urjc.es/optsicom/ccp/

6- The average time for the final solution to be found for the first time (FTime) in seconds [50].

7- The number of found solutions that are equal to the BKS (#BEST).

| Results of ABC algorithm | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $P_{ls}$ | $q_s$ | *limit* | LS Type | Best | Average | SD | #BEST | RPD | CTime | FTime |
| 0.5 | 0.1*n* | 0.5*n* | LS1 | 1342.17 | 1339.65 | 2.67 | 6/20 | 0.18% | 75.7 | 72.15 |
| 0.6 | 0.1*n* | 0.5*n* | LS1 | 1342.17 | 1340.80 | 1.35 | 6/20 | 0.10% | 77.5 | 73.1 |
| 0.7 | 0.1*n* | 0.5*n* | LS1 | 1342.17 | 1340.49 | 2.80 | 8/20 | 0.12% | 77.3 | 71.7 |
| 0.8 | 0.1*n* | 0.5*n* | LS1 | 1342.17 | 1340.43 | 2.25 | 8/20 | 0.12% | 75.0 | 70.65 |
| 0.9 | 0.1*n* | 0.5*n* | LS1 | 1342.17 | 1341.02 | 1.99 | 12/20 | 0.08% | 72.6 | 69.1 |
| **1** | **0.1*n*** | **0.5*n*** | **LS1** | **1342.17** | **1341.40** | **1.99** | **15/20** | **0.05%** | **65.1** | **61.6** |
| 1 | 0.2*n* | 0.5*n* | LS1 | 1342.17 | 1341.48 | 1.05 | 12/20 | 0.05% | 77.9 | 71.5 |
| 1 | 0.3*n* | 0.5*n* | LS1 | 1342.17 | 1340.92 | 1.54 | 10/20 | 0.09% | 83.7 | 79.25 |
| 1 | 0.1*n* | *n* | LS1 | 1342.17 | 1340.26 | 2.56 | 11/20 | 0.14% | 73.25 | 67.7 |
| 1 | 0.1*n* | 2*n* | LS1 | 1342.17 | 1341.18 | 1.66 | 13/20 | 0.07% | 72.55 | 66.95 |

Table 1: Parameter-tuning for the instance Sparse82_01.

| Results of ABC algorithm | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| *Pls* | *qs* | *limit* | LS Type | Best | Average | SD | #BEST | RPD | CTime | FTime |
| 0.5 | 0.1*n* | 0.5*n* | LS1 | 1354.61 | 1348.70 | 5.92 | 8/20 | 0.43% | 77.5 | 71 |
| 0.6 | 0.1*n* | 0.5*n* | LS1 | 1354.61 | 1349.65 | 4.58 | 7/20 | 0.36% | 77.85 | 70.25 |
| 0.7 | 0.1*n* | 0.5*n* | LS1 | 1354.61 | 1351.09 | 3.15 | 6/20 | 0.25% | 81.3 | 74.8 |
| 0.8 | 0.1*n* | 0.5*n* | LS1 | 1354.61 | 1352.10 | 2.57 | 10/20 | 0.18% | 73.65 | 68.2 |
| 0.9 | 0.1*n* | 0.5*n* | LS1 | 1354.61 | 1352.04 | 2.35 | 9/20 | 0.18% | 72.65 | 66.3 |
| **1** | **0.1*n*** | **0.5*n*** | **LS1** | **1354.61** | **1352.05** | **3.63** | **12/20** | **0.18%** | **61.8** | **56.35** |
| 1 | 0.2*n* | 0.5*n* | LS1 | 1354.61 | 1350.94 | 2.52 | 4/20 | 0.27% | 81.45 | 77 |
| 1 | 0.3*n* | 0.5*n* | LS1 | 1354.61 | 1347.63 | 3.99 | 2/20 | 0.51% | 85.3 | 78.8 |
| 1 | 0.1*n* | *n* | LS1 | 1354.61 | 1352.93 | 3.16 | 15/20 | 0.12% | 60.5 | 56.15 |
| 1 | 0.1*n* | 2*n* | LS1 | 1354.61 | 1351.89 | 2.32 | 8/20 | 0.2% | 69.05 | 63.65 |

Table 2: Parameter-tuning for the instance Sparse82_06.

From the results provided in Tables 1 and 2, we have drawn the following conclusions:

- The $P_{ls}$ parameter had the most significant impact on ABC's performance. Overall, the best-classified configuration (the bold row) employed a high value for the $P_{ls}$ parameter because it records the lowest time. Therefore, the ABC's behaviour was better when we applied the LS with a higher probability.
- The $q_s$ and *limit* parameters showed that there were no significant variations between the best configuration and the remaining ones for the selected instances. As a result, the values $q_s = 0.1n$ and $limit = 0.5n$ were fixed.
- The best configuration was found using $P_{ls} = 1$, $q_s = 0.1n$, $limit = 0.5n$ and LS without violation of all the remaining experiments.

### 3.2 Comparison of the ABC Algorithm Results and State-of-the-art Metaheuristics

In this part we will display the results of employing the ABC algorithm to solve the CCP after the selection process of best parameters values for all of the dataset instances, and we will compare the performance results of our suggested ABC algorithm with the results of the following benchmark algorithms: TS, GRASP and IVNS, which were taken from [12].

The source code for these benchmark algorithms was available online and cited in the paper [12]. We were encouraged to retest it on our EC2 instances using the same code, coded in C++, and compare our results with those of this retested algorithm to get a reasonable comparison between the algorithms. Lai and colleagues' [12] benchmark algorithms had a stop condition: a time limit of $1.0 \times n$. In our proposed LSs, we addressed the average complete computational time (CTime) in the twenty runs in seconds, but the benchmark algorithms addressed the average running time to get the final objective value.

| Results of ABC algorithm | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Instances | $p_{ls}$ | *qs* | *limit* | Best | Average | SD | #Best | Avg RPD | CTime | FTime |
| Sparse82_01 | 1 | 0.1*n* | 0.5*n* | 1342.17 | 1341.40 | 1.99 | 15/20 | 0.05% | 65.1 | 61.6 |
| Sparse82_02 | 1 | 0.1*n* | 0.5*n* | 1306.64 | 1302.17 | 2.86 | 2/20 | 0.34% | 81.75 | 78.4 |
| Sparse82_03 | 1 | 0.1*n* | 0.5*n* | 1353.94 | 1351.05 | 2.43 | 3/20 | 0.21% | 81.1 | 78.55 |
| Sparse82_04 | 1 | 0.1*n* | 0.5*n* | 1291.22 | 1286.46 | 3.31 | 2/20 | 0.36% | 79.75 | 76.35 |
| Sparse82_05 | 1 | 0.1*n* | 0.5*n* | 1352.35 | 1351.67 | 0.71 | 6/20 | 0.05% | 79.7 | 74.2 |
| Sparse82_06 | 1 | 0.1*n* | 0.5*n* | 1354.61 | 1352.05 | 3.63 | 12/20 | 0.18% | 61.8 | 56.35 |
| Sparse82_07 | 1 | 0.1*n* | 0.5*n* | 1266.94 | 1264.75 | 3.42 | 6/20 | 0.17% | 81.7 | 79.25 |
| Sparse82_08 | 1 | 0.1*n* | 0.5*n* | 1393.02 | 1393.02 | 0 | 20/20 | 0 | 65.8 | 60.3 |
| Sparse82_09 | 1 | 0.1*n* | 0.5*n* | 1294.12 | 1293.57 | 0.39 | 6/20 | 0.04% | 81.55 | 79.05 |
| Sparse82_10 | 1 | 0.1*n* | 0.5*n* | 1356.98 | 1353.30 | 5.30 | 1/20 | 0.27% | 81.0 | 78.5 |

Table 3: The values of best, average and SD objective function values for all instances run twenty times and average RPD, number best, CTime and Ftime of applying the ABC.

| Results of Benchmark Algorithms | | | | | |
|---|---|---|---|---|---|
| Instances | TS or GRASP or IVNS | Best | Average | SD | $Time_{avg}$ |
| Sparse82_01 | TS | 1341.46 | 1321.99 | 12.43 | 11.92 |
| | GRASP | 1342.17 | 1342.00 | 0.45 | 31.47 |
| | IVNS | 1342.17 | **1342.17** | 0 | 0.29 |
| Sparse82_02 | TS | 1304.07 | 1286.35 | 12.11 | 10.80 |
| | GRASP | 1306.64 | 1304.47 | 0.61 | 36.91 |
| | IVNS | 1306.64 | **1306.64** | 0 | 1.70 |
| Sparse82_03 | TS | 1353.94 | 1334.06 | 13.01 | 10.67 |
| | GRASP | 1352.41 | 1348.17 | 1.39 | 36.59 |
| | IVNS | 1353.94 | **1353.94** | 0 | 0.27 |
| Sparse82_04 | TS | 1291.22 | 1272.75 | 17.70 | 8.92 |
| | GRASP | 1289.79 | 1286.77 | 1.66 | 43.17 |
| | IVNS | 1291.22 | **1291.22** | 0 | 6.48 |
| Sparse82_05 | TS | 1352.35 | 1330.77 | 24.17 | 10.24 |
| | GRASP | 1352.35 | 1352.26 | 0.22 | 25.59 |
| | IVNS | 1352.35 | **1352.35** | 0 | 0.13 |
| Sparse82_06 | TS | 1354.61 | 1336.46 | 20.07 | 2.97 |
| | GRASP | 1354.61 | 1349.38 | 3.36 | 39.52 |
| | IVNS | 1354.61 | **1354.61** | 0 | 0.12 |
| Sparse82_07 | TS | 1266.94 | 1236.01 | 21.41 | 10.12 |
| | GRASP | 1266.94 | 1266.28 | 0.84 | 35.94 |
| | IVNS | 1266.94 | **1266.94** | 0 | 0.82 |
| Sparse82_08 | TS | 1390.90 | 1359.68 | 34.56 | 2.53 |
| | GRASP | 1393.02 | **1393.02** | 0 | 0.52 |
| | IVNS | 1393.02 | **1393.02** | 0 | 0.05 |
| Sparse82_09 | TS | 1294.12 | 1277.14 | 19.05 | 6.65 |
| | GRASP | 1294.12 | 1293.39 | 0.22 | 20.62 |
| | IVNS | 1294.12 | **1294.12** | 0 | 1.02 |
| Sparse82_10 | TS | 1356.98 | 1331.81 | 20.42 | 12.67 |
| | GRASP | 1356.98 | 1356.51 | 0.10 | 25.50 |
| | IVNS | 1356.98 | **1356.98** | 0 | 0.67 |

Table 4: Best, average and SD objective function values and time average of the TS, GRASP and IVNS run over twenty times on ten instances.

The experimental results from Table 3 and the comparison of the results of the benchmark algorithms from Table 4 have led us to the following conclusions:

- In terms of the best solution, the performance of the ABC was equivalent to that of the TS, GRASP and the IVNS for the following instances: Sparse82_05, Sparse82_06, Sparse82_09 and Sparse82_10. In the instances Sparse82_01, Sparse82_02 and Sparse82_08, ABC slightly outperformed the TS. In the instances Sparse82_03 and Sparse82_04, the ABC slightly outperformed GRASP.

- Regarding the average solution, the performance of the ABC was comparable to that of the IVNS for all instances (the IVNS recorded the BKS for all the instances). In the instances Sparse82_01, Sparse82_02, Sparse82_04, Sparse82_05 Sparse82_07 and Sparse82_10, GRASP slightly outperformed the ABC. In the instances Sparse82_03, Sparse82_06, and Sparse82_09, the ABC slightly outperformed GRASP. In the instance of Sparse82_08, the ABC, the IVNS and GRASP recorded the same performance results, which also equalled to the BKS value. The TS algorithm recorded the worst results for all the instances compared to the ABC, GRASP and the IVNS.

- With respect to the average RPD, all values were less than 1.0, meaning that all of the average results were close to the IVNS values (we compared the average results with the IVNS because it recorded the best values).

- With respect to FTime, all results were under 80 seconds, meaning that the algorithm recoded the best solution before approximately 97% of the full time ($t_{max}$).

The plots represent the value of the objective functions of ABC and the results of the benchmark algorithm values in twenty runs for the ten instances. The *x*-axis presents the test file name, and the *y*-axis presents the objective function values. The results of the comparison between the ABC and the benchmark algorithms are plotted in the following figures:

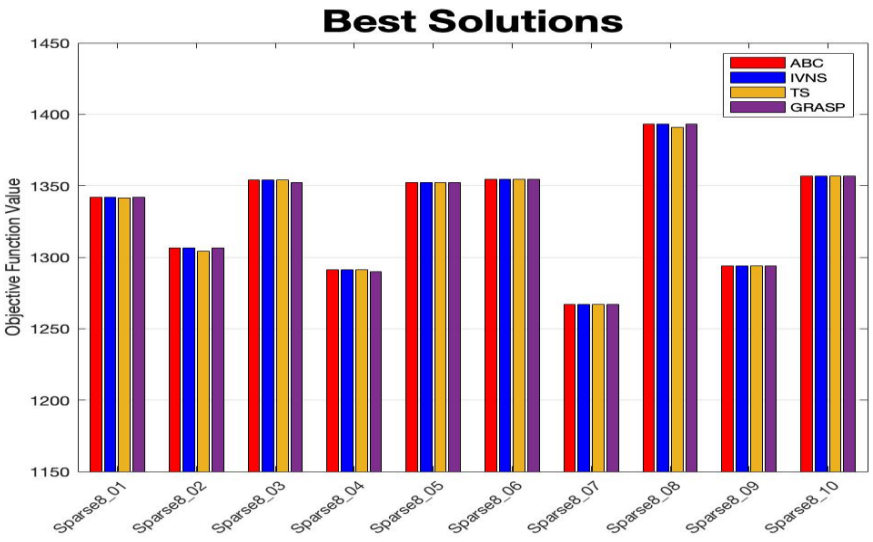


Figure 5: Best results of the ABC and benchmark algorithms.

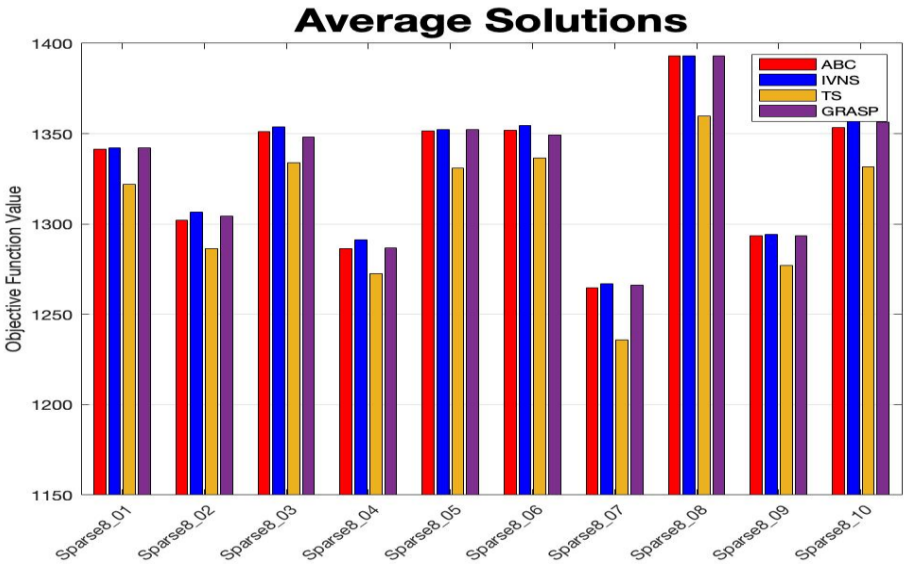


Figure 6: Average results of the ABC and benchmarks algorithms.

## Conclusion

An effective metaheuristic algorithm has been suggested in this paper for solving the CCP. We can draw the conclusion that when applying the ABC to solve the CCP as a metaheuristic algorithm, the experimental results were promising and compared well with other state-of-the-art algorithms in the literature. By analysing the parameter-tuning, this paper has demonstrated how parameters can affect the results of a solution in terms of time and quality. This work has clearly developed an effective metaheuristic algorithm for solving the CCP.

To better understand the implications of these results, future studies could address the application of the suggested approach on different COPs and investigate the integration of other algorithms (i.e. guided local search with the ABC to solve the CCP). Future studies could also investigate the applications of the suggested ABC algorithm for various problems and data sets. I suggest that the violation of LS constraints handling techniques should be investigated more thoroughly in future research.

## Note:

This paper is taken from my thesis (A Metaheuristic to Solve the Capacitated Clustering Problem) done in May 2020 at (Al Imam Mohammad Ibn Saud Islamic University, College of Computer & Information Sciences, Department of Computer Science).

## References

[1] A. Martínez-Gavara, V. Campos, M. Gallego, M. Laguna, and R. Martí, "Tabu search and GRASP for the capacitated clustering problem," Comput. Optim. Appl., vol. 62, no. 2, pp. 589–607, 2015, doi: 10.1007/s10589-015-9749-1.

[2] Y. Deng and J. F. Bard, "A reactive GRASP with path relinking for capacitated clustering," J. Heuristics, vol. 17, no. 2, pp. 119–152, 2011, doi: 10.1007/s10732-010-9129-z. Note: the CCP application is been mentioned by other authors such as: (Al-Sultan and Khan 1996; Bard and Jarrah 2009; Daganzo 2005; Kaufman and Roussweuw 1990; Laporte et al.1989).

[3] M. Lewis, H. Wang, and G. Kochenberger, "Exact Solutions to the Capacitated Clustering Problem: A Comparison of Two Models," Ann. Data Sci., vol. 1, no. 1, pp. 15–23, 2014, doi: 10.1007/s40745-014-0003-y. Note: the CCP application is been mentioned by other authors such as: (Negeriros and Palhano, Deng and Bard, and Chou et al.).

[4] Q. Zhou, U. Benlic, Q. Wu, and J. K. Hao, "Heuristic search to the capacitated clustering problem," Eur. J. Oper. Res., vol. 273, no. 2, pp. 464–487, 2019, doi: 10.1016/j.ejor.2018.08.043.

[5] C. A. Chou, W. A. Chaovalitwongse, T. Y. Berger-Wolf, B. Dasgupta, and M. V. Ashley, "Capacitated clustering problem in computational biology: Combinatorial and statistical approach for sibling reconstruction," Comput. Oper. Res., vol. 39, no. 3, pp. 609–619, 2012, doi: 10.1016/j.cor.2011.04.017.

[6] J. M. Mulvey and M. P. Beck, "Solving capacitated clustering problems," Eur. J. Oper. Res., vol. 18, no. 3, pp. 339–348, 1984, doi: 10.1016/0377-2217(84)90155-3.

[7] I. H. Osman and N. Christofides, "Capacitated clustering problems by hybrid simulated annealing and tabu search," Int. Trans. Oper. Res., vol. 1, no. 3, pp. 317–336, 1994, doi: 10.1016/0969-6016(94)90032-9.

[8] S. Ahmadi and I. H. Osman, "Greedy random adaptive memory programming search for the capacitated clustering problem," Eur. J. Oper. Res., vol. 162, no. 1, pp. 30–44, 2005, doi: 10.1016/j.ejor.2003.08.066.

[9] F. O. De França, F. J. Von Zuben, and L. N. De Castro, "A MAX MIN Ant System applied to the Capacitated Clustering Problem," Mach. Learn. Signal Process. XIV - Proc. 2004 IEEE Signal Process. Soc. Work., no. January, pp. 755–764, 2004, doi: 10.1109/mlsp.2004.1423042.

[10] S. Scheuerer and R. Wendolsky, "A scatter search heuristic for the capacitated clustering problem," Eur. J. Oper. Res., vol. 169, no. 2, pp. 533–547, 2006, doi: 10.1016/j.ejor.2004.08.014.

[11] N. M. Darani, V. Ahmadi, Z. S. Eskandari, and M. Yousefikhoshbakht, "Solving the Capacitated Clustering Problem by a Combined Meta-Heuristic Algorithm," J. Adv. Comput. Res., vol. 4, no. 1, pp. 89–100, 2013.

[12] X. Lai and J. K. Hao, “Iterated variable neighborhood search for the capacitated clustering problem,” Eng. Appl. Artif. Intell., vol. 56, pp. 102–120, 2016, doi: 10.1016/j.engappai.2016.08.004.

[13] J. Brimberg, N. Mladenović, R. Todosijević, and D. Urošević, “Variable neighborhood descent for the capacitated clustering problem,” in International Conference on Discrete Optimization and Operations Research, 2016, pp. 336–349.

[14] J. Brimberg, N. Mladenović, R. Todosijević, and D. Urošević, “Local and variable neighborhood searches for solving the capacitated clustering problem,” Springer Optim. Its Appl., vol. 130, pp. 33–55, 2017, doi: 10.1007/978-3-319-68640-0_3.

[15] X. Lai, J. Hao, X. Lai, J. H. Iterated, X. Lai, and J. Hao, “Iterated variable neighborhood search for the capacitated clustering problem To cite this version : HAL Id : hal-01412523 Iterated variable neighborhood search for the capacitated clustering problem,” 2017.

[16] D. Karaboga, “An idea based on honey bee swarm for numerical optimization,” Technical report-tr06, Erciyes university, engineering faculty, computer …, 2005.

[17] F. J. Rodriguez, M. Lozano, C. García-Martínez, and J. D. González-Barrera, “An artificial bee colony algorithm for the maximally diverse grouping problem,” Inf. Sci. (Ny)., vol. 230, pp. 183–196, 2013, doi: 10.1016/j.ins.2012.12.020. p.186. Note: I have benefited from the results expression in this paper.

[18] D. Karaboga and B. Akay, “A comparative study of Artificial Bee Colony algorithm,” Appl. Math. Comput., vol. 214, no. 1, pp. 108–132, 2009, doi: 10.1016/j.amc.2009.03.090. p.109.

[19] D. Karaboga, C. Ozturk, N. Karaboga, and B. Gorkemli, “Artificial bee colony programming for symbolic regression,” Inf. Sci. (Ny)., vol. 209, pp. 1–15, 2012.

[20] S. Sundar, “Singh A (2010a) A swarm intelligence approach to the quadratic minimum spanning tree problem,” Inf Sci, vol. 180.

[21] P. Tapkan, L. Ozbakir, and A. Baykasoglu, “Modeling and solving constrained two-sided assembly line balancing problem via bee algorithms,” Appl. Soft Comput., vol. 12, no. 11, pp. 3343–3355, 2012.

[22] A. Singh, “An artificial bee colony algorithm for the leaf-constrained minimum spanning tree problem,” Appl. Soft Comput., vol. 9, no. 2, pp. 625–631, 2009.

[23] M. Ma, J. Liang, M. Guo, Y. Fan, and Y. Yin, “SAR image segmentation based on Artificial Bee Colony algorithm,” Appl. Soft Comput., vol. 11, no. 8, pp. 5205–5214, 2011.

[24] M. H. Kashan, N. Nahavandi, and A. H. Kashan, “DisABC: A new artificial bee colony algorithm for binary optimization,” Appl. Soft Comput., vol. 12, no. 1, pp. 342–352, 2012.

[25] T.-J. Hsieh, H.-F. Hsiao, and W.-C. Yeh, “Forecasting stock markets using wavelet transforms and recurrent neural networks: An integrated system based on artificial bee colony algorithm,” Appl. Soft Comput., vol. 11, no. 2, pp. 2510–2525, 2011.

[26] D. Karaboga and B. Akay, “A modified artificial bee colony (ABC) algorithm for constrained optimization problems,” Appl. Soft Comput., vol. 11, no. 3, pp. 3021–3031, 2011.

[27] G. A. Di Caro, M. Farooq, and M. Saleem, “Swarm intelligence based routing protocol for wireless sensor networks: Survey and future directions,” Inf. Sci., vol. 181, pp. 4597–4624, 2011.

[28] M. Saleem, I. Ullah, and M. Farooq, “BeeSensor: An energy-efficient and scalable routing protocol for wireless sensor networks,” Inf. Sci. (Ny)., vol. 200, pp. 38–56, 2012.

[29] R. Akbari, R. Hedayatzadeh, K. Ziarati, and B. Hassanizadeh, “A multi-objective artificial bee

colony algorithm," Swarm Evol. Comput., vol. 2, pp. 39–52, 2012.

[30] S. N. Omkar, J. Senthilnath, R. Khandelwal, G. N. Naik, and S. Gopalakrishnan, "Artificial Bee Colony (ABC) for multi-objective design optimization of composite structures," Appl. Soft Comput., vol. 11, no. 1, pp. 489–499, 2011.

[31] B. Jayalakshmi and A. Singh, "A hybrid artificial bee colony algorithm for the p-median problem with positive/negative weights," Opsearch, vol. 54, no. 1, pp. 67–93, 2017.

[32] B. Jayalakshmi and A. Singh, "A swarm intelligence approach for the p-median problem," Int. J. Metaheuristics, vol. 5, no. 2, pp. 136–155, 2016.

[33] M. S. Kıran, E. Özceylan, and T. Paksoy, "Artificial bee colony algorithm for solving uncapacitated facility location problems," in 25th European Conference on Operational Research, 2012, p. 165.

[34] M. S. Kiran and M. Gündüz, "XOR-based artificial bee colony algorithm for binary optimization," Turkish J. Electr. Eng. Comput. Sci., vol. 21, no. Sup. 2, pp. 2307–2328, 2013.

[35] S. Sabet, F. Farokhi, and M. Shokouhifar, "A novel artificial bee colony algorithm for the knapsack problem," in 2012 International Symposium on Innovations in Intelligent Systems and Applications, 2012, pp. 1–5.

[36] S. Sundar, A. Singh, and A. Rossi, "An artificial bee colony algorithm for the 0–1 multidimensional knapsack problem," in International Conference on Contemporary Computing, 2010, pp. 141–151.

[37] A. S. Bhagade and P. V Puranik, "Artificial bee colony (ABC) algorithm for vehicle routing optimization problem," Int. J. Soft Comput. Eng., vol. 2, no. 2, pp. 329–333, 2012.

[38] S. Iqbal, M. Kaykobad, and M. S. Rahman, "Solving the multi-objective vehicle routing problem with soft time windows with the help of bees," Swarm Evol. Comput., vol. 24, pp. 50–64, 2015.

[39] A. Rossi, A. Singh, and M. Sevaux, "A metaheuristic for the fixed job scheduling problem under spread time constraints," Comput. Oper. Res., vol. 37, no. 6, pp. 1045–1054, 2010.

[40] R. Zhang, S. Song, and C. Wu, "A hybrid artificial bee colony algorithm for the job shop scheduling problem," Int. J. Prod. Econ., vol. 141, no. 1, pp. 167–178, 2013.

[41] M. S. Kıran, H. İşcan, and M. Gündüz, "The analysis of discrete artificial bee colony algorithm with neighborhood operator on traveling salesman problem," Neural Comput. Appl., vol. 23, no. 1, pp. 9–21, 2013.

[42] H. E. Kocer and M. R. Akca, "An improved artificial bee colony algorithm with local search for traveling salesman problem," Cybern. Syst., vol. 45, no. 8, pp. 635–649, 2014.

[43] A. K. Alshamiri, A. Singh, and B. R. Surampudi, "Artificial bee colony algorithm for clustering: an extreme learning approach," Soft Comput., vol. 20, no. 8, pp. 3163–3176, 2016.

[44] V. R. Dokku and A. Singh, "An artificial bee colony algorithm for the minimum average routing path clustering problem in multi-hop underwater sensor networks," in International Conference on Computing and Communication Systems, 2011, pp. 212–219.

[45] D. Karaboga and C. Ozturk, "A novel clustering approach : Artificial Bee Colony ( ABC ) algorithm," vol. 11, pp. 652–657, 2011, doi: 10.1016/j.asoc.2009.12.025.

[46] S. S. Choong, L.-P. Wong, and C. P. Lim, "An artificial bee colony algorithm with a modified choice function for the Traveling Salesman Problem," Swarm Evol. Comput., vol. 44, pp. 622–635, 2019.

[47] K. Balasubramani and K. Marcus, "A comprehensive review of artificial bee colony algorithm," Int. J. Comput. Technol., vol. 5, no. 1, pp. 15–28, 2013.

[48] D. Karaboga, B. Gorkemli, C. Ozturk, and N. Karaboga, “A comprehensive survey: artificial bee colony (ABC) algorithm and applications,” Artif. Intell. Rev., vol. 42, no. 1, pp. 21–57, 2014.

[49] M. Gallego, M. Laguna, R. Martí, and A. Duarte, “Tabu search with strategic oscillation for the maximally diverse grouping problem,” J. Oper. Res. Soc., vol. 64, no. 5, pp. 724–734, 2013, doi: 10.1057/jors.2012.66.

[50] A. Alsheddy, “A two-phase local search algorithm for the ordered clustered travelling salesman problem,” Int. J. Metaheuristics, vol. 7, no. 1, p. 80, 2018, doi: 10.1504/ijmheur.2018.10012913. Note: I have benefited from the results expression in this paper.

**Footnote:**

(1) http://grafo.etsii.urjc.es/optsicom/ccp/