Artificial Bee Colony Optimization Algorithm for Uncapacitated Facility Location Problems

Nukhet Tuncbilek*, Fatih Tasgetiren**, and Sakir Esnaf**

Abstract. In this paper, a new multiple facility location and allocation method is proposed. This algorithm assigns demand points to pre-determined facilities which have infinite capacities, by using an artificial bee colony optimization with local search to solve discrete uncapacitated multiple facility location problems. This method is tested and compared with continuous and discrete particle swarm optimization based facility location methods on alternative models with or without local search for optimizing the well-known benchmark problems and generated problems. Similar test is conducted on actual demand and transportation cost data from a fertilizer manufacturer from Turkey, which includes 11 facility locations and 768 demand points. Proposed ABC-based algorithm exhibited better performance than the PSO-based algorithms.

JEL Classification Codes: C60, C61
Keywords: Discrete Uncapacitated Facility Location; Artificial Bee Colony Optimization; Local Search
1. Introduction

The facility location problem consists of determining the locations of the facilities and the flows of the commodity from facilities to clients, such that total costs are minimized. Costs are composed of fixed and variable costs. Costs related to establishment of facilities are fixed costs. Operational costs and transportation costs are variable costs. If an arbitrary number of customers can be connected to a facility, the problem is called an uncapacitated facility location (UFL) problem [Wuet al., 2006]. UFL problem ignores capacity of facilities, such that each facility can produce and ship unlimited quantities of the commodity. Numerous mathematical models were proposed for this problem. [Ghosh, (2003), Hsieh and Tien (2004), Levin and Ben-Israel (2004), Xu and Xu (2005), Greistorfer and Rego (2006), Dohn et. Al (2007)].

Most of studies on facility location problems are on continuous problems, which have infinite sets of possible locations. In discrete facility location problems only finite set of locations are probable [Seppala (1997)]. In this study, a discrete uncapacitated multiple facility location models are taken into consideration. In these models, locations of potential facilities are certain at the beginning, but which of the locations will be opened is under question.

Due to the NP-hard nature of the discrete UFL problem, heuristic algorithms are studied; since exact algorithms may not be sufficient to solve larger problems. The number of studies which employ heuristics and meta-heuristics such as Alves and Almeida (1992), Al-Sultan and Al-Fawzan (1999), Ghosh (2003), Aydin and Fogarty (2004), Michel and Van Hentenryck (2004) and Sun (2006) are very limited. Similarly there are few particle swarm optimization based algorithms for solving discrete uncapacitated multiple facility location problems. The continuous particle swarm optimization (CPSO) algorithm is proposed by Şevkli and Güner (2006) for discrete UFL problem. In CPSO, each particle has multi-dimensional continuous position and velocity values same as in PSO. Since problem domain is discrete in UFL, continuous values of particles need to be mapped to discrete values. For discrete optimization problems, discrete PSO (DPSO) algorithm was first proposed by Pan et al. (2008) for the no-wait flowshop scheduling problem. Güner and Şevkli (2008) developed DPSO algorithm for UFL problem.
Artificial Bee Colony Optimization Algorithm
for Uncapacitated Facility Location Problems

There is a recent study that uses artificial bee colony algorithm with local search for solving discrete uncapacitated multiple facility location problems by Kashan et al. (2010). They proposed a new method DisABC, which is designed for binary optimization. In their study, the algorithm is used to solve discrete UFL problem in order to test effectiveness. Unlike our study, they used smaller problems from OR-Library.

In this paper, artificial bee colony (ABC) algorithm which is a new meta-heuristic approach, proposed by Karaboğa (2005) is adopted to solve the UFL problem. A local search heuristic is used in order to refine the solutions found by ABC algorithm. The proposed ABC algorithm based method is compared against the CPSO approach proposed in Şevkli and Güner (2006), and the DPSO approach proposed in Güner and Şevkli (2008). Computational results demonstrate the effectiveness of ABC with local search in comparison with these approaches.

The remainder of this paper is organized as follows. UFL problem is described in Section 2. Procedures of methodology and the proposed method are presented in Section 3. Experimental study is presented in Section 4, including comparison of proposed ABC and ABC\_LS algorithms for UFL problem with CPSO, CPSO\_LS, DPSO, DPSO\_LS. A case study on facility location problem of a fertilizer manufacturer from Turkey is explained in Section 5. Finally, the concluding remarks are presented in Section 6.

### 2. The Uncapacitated Facility Location (UFL) Problem

An UFL problem with \( m \) customers and \( n \) candidate facility sites can be represented by a network with \( m+n \) nodes and \( m\times n \) arcs. In the UFL model, \( f_j \) is used to represent the cost of opening facility \( j \) and \( c_{ij} \) is used to represent the cost of serving customer \( i \) from facility \( j \) or assigning customer \( i \) to facility \( j \). We assume that \( c_{ij} \geq 0 \) for all \( i=1, \ldots, m \) and \( j=1, \ldots, n \) and \( f_j > 0 \) for all \( j=1, \ldots, n \). A binary variable \( y_j \) is used to represent the status of facility \( j \) in the model. Facility \( j \) will be open only if \( y_j = 1 \) in the solution. A binary variable \( x_{ij} \) is used for the road from customer \( i \) to facility \( j \) in the model. Customer \( i \) will be served by facility \( j \) only if \( x_{ij} = 1 \) in the solution. However, each \( x_{ij} \) can be treated as a continuous variable and will have a binary value in the solution. The solution process of the UFL problem is to find an optimal solution that satisfies all
customer demand and minimizes the total cost.

The UFL problem can be formally stated as follows (Sun, 2006):

\[
\text{minimize } \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} + \sum_{j=1}^{n} f_{j} y_{j} \quad (1)
\]

subject to

\[
\sum_{j=1}^{n} x_{ij} = 1, \forall i = 1, \ldots, m \quad (2)
\]

\[
x_{ij} \leq y_{ij}, \forall i = 1, \ldots, m, j = 1, \ldots, n \quad (3)
\]

\[
x_{ij} \geq 0, \forall i = 1, \ldots, m, j = 1, \ldots, n \quad (4)
\]

\[
y_{j} = \{0, 1\}, \forall j = 1, \ldots, n \quad (5)
\]

3. Methodology

In this section the details of the proposed an artificial bee colony (ABC) algorithm with local search is explained for the UFL problem.

3.1. Artificial Bee Colony (ABC) Optimization Algorithm

Artificial Bee Colony (ABC) algorithm is one of most recently introduced swarm intelligence based meta-heuristic method by Derviş Karaboğa in 2005. The algorithm has been motivated by the intelligent behaviour of honey bees. This algorithm is as simple as Particle Swarm Optimization (PSO) and Differential Evolution (DE) algorithms.

Karaboğa proposed the artificial bee colony (ABC) algorithm which is inspired by foraging behaviour of honey bees. In the ABC algorithm, a problem is solved by exploring good solutions which are represented as food sources. The quality of the solution is represented by the nectar amount of that food source. In this algorithm, the first half of the bee colony are employed bees, the second half are the onlooker bees. The number of food sources is same as the number of employed bees. It is assumed that there is only one employed bee for every food source. Each employed bee is placed on a food source, and starts extracting nectar. The employed bee becomes a scout when the food source has no more nectar and moves away to look for another food source. As soon as a scout bee finds a new food source it again becomes an employed bee. The ABC algorithm initially places all employed bees on randomly generated food sources (solutions). Then iteratively, every employed bee determines a food source nearby their currently associated
Artificial Bee Colony Optimization Algorithm
for Uncapacitated Facility Location Problems

food source and evaluates its nectar amount (fitness). If it has more nectar than that of its current food source, then that employed bee moves to this new food source, otherwise it stays on its current food source.

The search can be materialized with following steps:

• Employed bees locate a food source close to current food source which is in their memory.
• The other half of the colony, onlooker bees wait in the hive and get information about rich food sources from employed bees which returned into the hive. Then the onlookers decide to go to one of the food sources and locate to a food source which is close to this food source.
• After some period, food source may be exhausted. An employed bee on such a food source becomes a scout and starts to search a new food source randomly.

The ABC algorithm is presented in Figure 1:

Initialize food sources at random positions and locate employed bees
repeat
    Move the employed bees around their food sources and determine their nectar amounts.
    Move the onlookers towards rich food sources and determine their nectar amounts.
    Determine exhausted food sources and assign employed bees as scout bees for searching new food sources.
    Memorize the best food source found so far.
until requirements are met

Figure 1: Pseudocode of ABC Algorithm

This cycle is repeated up to predefined number of iterations or predefined limit on CPU time. A food source can be interpreted as a possible solution to the optimization problem. The nectar amount of a food source represents the quality of the solution represented by that food source. Scout bees move to new directions so that colony can explore new food sources. While onlookers and employed bees exploiting good solutions in the search space, the scouts explore new unknown solutions (Karaboğa and Baştürk, 2007).

Onlooker bees move according to the information taken from an employed bee that is returned back to hive. When employed bees have finished collecting nectar, they come back to their hive and share
information with the onlooker bees by dancing longer or shorter, according to nectar amount of the last visited food source. Onlooker bees select a food source according to a probability which is proportional to the nectar amount of that food source. The probability \( p_i \) of selecting a food source \( i \) is determined using the following expression:

\[
p_i = \frac{\text{fit}_i}{\sum_{n=1}^{SN} \text{fit}_n} \tag{6}
\]

where

\( \text{fit}_i \) : fitness value of \( i^{th} \) solution which represents nectar amount at food source at \( i^{th} \) position.

\( SN \) : number of employed bees (also number of food sources)

Since the objective is minimizing \( f(x) \) values, fitness function is calculated as shown in (7), so that smaller \( f(x) \) values get higher fitness. In the mean time, \( f(x) \) values that are close to zero will get fitness close to 1. If \( f(x) \) has greater values, fitness value gets closer to zero. Hence fitness values can be used as weights for probability of selection in Equation (6).

\[
\text{fitness}(i) = \begin{cases} 
1 + \text{abs}(f(i)), & f(i) < 0 \\
1/(f(i)+1), & f(i) \geq 0
\end{cases} \tag{7}
\]

This weighted probability calculation shows that good food sources attract more onlookers than the bad ones. After all onlookers have decided which food source to move to, each of them determines a food source in the neighbourhood of their current food source and computes its fitness. Onlookers try to determine the best food source among all the neighbouring locations nearby a particular food source \( i \) and it will be the new location of the food source \( i \). After a predetermined number of iterations, if a solution represented by a particular food source is not improved, then that food source is abandoned by its associated employed bee, it becomes a scout and starts searching for a new food source randomly. In other words, this scout is assigned to a randomly generated food source (solution) and its status is changed from scout to employed bee. This process is repeated until the termination condition is satisfied.
Artificial Bee Colony Optimization Algorithm
for Uncapacitated Facility Location Problems

Movement of an employed bee around its current position is probabilistically formulated. Scanning food sources in the neighbourhood of a particular food source is done by altering the value of one randomly chosen solution parameter (dimension) and keeping other parameters unchanged. The value of the chosen parameter is changed by using the following formula:

\[ v_{ij} = x_{ij} + \phi_{ij} (x_{ij} - x_{kj}) \] (8)

where

- \( j \): 1, ..., \( D \) randomly chosen dimension
- \( v_{ij} \): new candidate location
- \( x_{ij} \): current location
- \( \phi_{ij} \): random factor between -1 and +1 which is generated by uniform probability distribution
- \( k \): a randomly chosen neighbour, where \( i \neq k \)
- \( x_{kj} \): location of a randomly chosen neighbour (\( k \)) at chosen dimension \( j \)

If the calculated value \( v_{ij} \) exceeds the acceptable range for dimension \( j \), it is set to the corresponding extreme value in that range. (\( x_{min}^j \) and \( x_{max}^j \))

Fitness values for \( x_{ij} \) and \( v_{ij} \) are compared and the one with better fitness value is chosen as the new position for the \( i^{th} \) employed bee. This is a greedy selection process. If nectar amount of the candidate location is better than the present one, the bee forgets the present location and memorizes the candidate location produced by Equation 8. Otherwise, the bee keeps its present location in the memory.

Onlooker bees also set their location according to the Equation 8, same as employed bees. Behaviour of scouts is also probabilistically defined as follows:

\[ x_{i}^j = x_{min}^j + rand[0,1](x_{max}^j - x_{min}^j) \] (9)

If a particular employed bee does not improve the solution in a predefined number of iterations called "limit", then that employed bee
becomes a scout by leaving current position and looking for a new food source at randomly set new position generated by Equation 9.

In formulations explained above, it is shown that basic ABC has three control parameters: (Karaboğa and Akay, 2009).
- the number of food sources which is equal to the number of employed or onlooker bees ($SN$)
- the value of food limit
- the maximum cycle number ($MCN$)

### 3.2. Uncapacitated Facility Location Model Based on Artificial Bee Colony Optimization

Parameters of ABC algorithm are number of employed / onlooker bees, number of scout bees, food limit and number of iterations. Throughout test runs in this study, these parameters are taken as 10, 1, 250 and 1000 respectively.

In UFL problem, there are $n$ facilities under question. Some of these facilities will be opened and other will not, so that total cost of serving customers from these opened facilities is optimized. Search space of the artificial bee colony is $n$-dimensional, where $n$ is the total number of facilities. $n$-dimensional position vector of each employed bee is mapped to a binary valued vector to determine whether each facility will be opened or not. Position values are converted to binary variables as follows:

$$y_i = \left\lfloor x_i \right\rfloor \mod 2$$  \hspace{1cm} (10)

The absolute value of a position value is first divided by 2 and then the remainder is floored to nearest integer, which may be 0 or 1 (Güner and Şevkli, 2008).

These values construct open facility vector ($Y_i$) for $n$-facilities under question. $Y_i$ represents the opening or closing facilities based on the position vector ($X_i$). $Y_i = \{y_{i1}, y_{i2}, y_{i3}, \ldots, y_{in}\}$, where $y_{ik}$ represents opening or closing $k^{th}$ facility of the $i^{th}$ particle. For an $n$-facility problem, each particle contains $n$ number of dimensions.
Artificial Bee Colony Optimization Algorithm for Uncapacitated Facility Location Problems

Pseudocode for ABC algorithm for UFL problem is shown in Figure 2.

Initialize food source positions randomly
for all food sources do
    Calculate open facilities vector (10)
    Calculate and memorize fitness value
end for
while maximum iteration is not reached do
    for all employed bees do
        Randomly choose a dimension
        Move position in this chosen dimension by random value between (-1,1) (8)
        Update position (8)
        Find open facility vectors (10)
        Calculate fitness value (total cost) using open facility vector
        if new fitness value is better than previous one then
            Move to new position
        else
            Increment counter by 1
            if counter > food limit then
                Employed bee becomes a scout bee
                A scout bee becomes an employed be at a randomly set position (9)
            end if
        end if
        Memorize best solution and position vector
    end for
    for all onlooker bees do
        Randomly choose a food source pointed by employed bees that return to hive (6)
        Evaluate the chosen position, same way described for employed bees
        if fitness value is better then
            Move food source to evaluated position
        end if
    end for
end while
(ABC) Apply local search to best solution

Figure 2: Pseudocode of ABC Algorithm for UFLP

Similar to CPSO or DPSO, fitness value for each employed bee is calculated by adding up delivery costs from opened facilities to nearest customers.

3.3. Local Search for CPSO, DPSO and ABC Algorithms

It is observed that CPSO and DPSO could not reach optimal solutions for large problems. In order to improve solutions that are found by CPSO and
DPSO, Güner and Şevkli (2008) employed a local search algorithm to CPSO and DPSO. The local search method looks for better solutions in the neighbourhood of the global best particle in every generation.

The way of how neighbour solutions are produced is very important to get better results. Local search algorithm takes the global best solution at the end of each iteration, and two randomly selected position values of the position vector are modified. In CPSO_{LS}, this is done by adding 1 to $x_i$. In DPSO_{LS}, value of $y_i$ is flipped between 0 and 1, by subtracting $y_i$ from 1.

$$x_i \leftarrow x_i + 1 \quad (11)$$

$$y_i \leftarrow 1 - y_i \quad (12)$$

This operation is repeated as far as new neighbour generates better solution. The local search algorithm is shown in Figure 3. At end of each iteration, global best result of CPSO and DPSO is taken as input by the local search algorithm. In order to generate diverse alternatives, two facilities ($\eta$ and $\kappa$) are picked and their values are flipped.

The global best found at the end of each iteration of CPSO and DPSO is adopted as the initial solution by the local search algorithm. In order not to lose the best found and to diversify the solution, the global best is modified with two facilities which are randomly chosen. Then flip operator is applied to existing solution, as long as it gets a better solution. If the produced alternative do not have a better solution, loop counter is incremented by 1. Local search algorithm allows maximum of $n$ unsuccessful trials in order to guarantee reasonable run time (Güner and Şevkli, 2008).

The same local search is applied to ABC, since bees in ABC algorithm have $n$-dimensional position vectors similar to CPSO. Application of ABC algorithm to UFL problem was described in subsection 3.2.

(DPSO_{LS}) $s_0 \leftarrow$ Global best open facility vector ($Y_g$)
(CPSO_{LS}) $s_0 \leftarrow$ Global best position vector ($X_g$)
(ABC_{LS}) $s_0 \leftarrow$ Global best food source ($X_g$)

Modify $s_0$ based on $\eta, \kappa$ and set to $s$
Artificial Bee Colony Optimization Algorithm for Uncapacitated Facility Location Problems

$\text{loop} \leftarrow 0$
repeat
    Apply Flip to $s$ and get $s_1$
    if $f(s_1) \leq f(s)$ then
        Replace $s$ with $s_1$
    else
        $\text{loop} \leftarrow \text{loop} + 1$
    end if
until $\text{loop} = n$
if $f(s) \leq f(s_0)$ then
    (DPSO,LS) Replace $Y_g$ with $s$
    (CPSO,LS) Replace $X_g$ with $s$
    (ABC,LS) Replace $X_g$ with $s$
end if

Figure 3: Pseudocode for Local Search Applied to DPSO, CPSO and ABC

4. Experimental Study

In this study, the proposed ABC and ABC,LS algorithms for UFL problem are tried on sample UFL problem data sets which are known as OR-Library by Beasley (2005). These problems have different sizes where $m$ is number of customers, $n$ is number of facilities. From this library, relatively larger ($m \times n = 1000 \times 100$) three problems entitled as CapA, CapB and CapC are selected. Optimal solutions of each problem are also given along with the problem files. Four new larger problem sets with sizes ($m \times n = 2000 \times 200$) and ($m \times n = 3000 \times 300$) are produced by combining two and three of these large problem sets together (CapA, CapB, CapC), as shown in Table 1. Experiments are executed on these seven large problem sets.
Table 1: Construction of Larger Problem Sets by Using CapA, CapB and CapC Files

(a) Combination of CapA and CapB (200x2000)

<table>
<thead>
<tr>
<th>Customers</th>
<th>Facility Locations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000...1</td>
<td>CapA</td>
</tr>
<tr>
<td>2000...1001</td>
<td>CapB</td>
</tr>
</tbody>
</table>

(b) Combination of CapB and CapC (200x2000)

<table>
<thead>
<tr>
<th>Customers</th>
<th>Facility Locations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000...1</td>
<td>CapB</td>
</tr>
<tr>
<td>2000...1001</td>
<td>CapC</td>
</tr>
</tbody>
</table>

(c) Combination of CapA and CapC (200x2000)

<table>
<thead>
<tr>
<th>Customers</th>
<th>Facility Locations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000...1</td>
<td>CapA</td>
</tr>
<tr>
<td>2000...1001</td>
<td>CapC</td>
</tr>
</tbody>
</table>

(d) Combination of CapA, CapB and CapC (300x3000)

<table>
<thead>
<tr>
<th>Customers</th>
<th>Facility Locations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000...1</td>
<td>CapA</td>
</tr>
<tr>
<td>2000...1001</td>
<td>CapB</td>
</tr>
<tr>
<td>3000...2001</td>
<td>CapC</td>
</tr>
</tbody>
</table>

The format of OR-Library uncapacitated location problem data files is shown below:

- number of potential facility locations \( m \) | number of customers \( n \)
Artificial Bee Colony Optimization Algorithm for Uncapacitated Facility Location Problems

- for each potential facility location \(i \ (i = 1, \ldots, m)\)
  - capacity | fixed cost
- for each customer \(j \ (j = 1, \ldots, n)\)
  - demand | cost of allocating all of the demand of \(j\) to facility \(i\) \((i = 1, \ldots, m)\)

Pseudocode for producing a new problem data set by combining two existing data sets is shown in Figure 4. Two or three data set files that will be combined are opened in parallel. Optimal solutions for the generated problems were not calculated and left as zero.

On these benchmark UFL problems, the performances of the proposed algorithms (ABC and ABC\_LS) are compared with CPSO, DPSO, CPSO\_LS, DPSO\_LS algorithms. In order to make a fair comparison, all these six algorithms are programmed in the same platform, in conformance with the definition of the researchers [Karaboğa (2005), Şevkli and Güner (2006), Güner and Şevkli (2008)]. Algorithms were developed as a Windows application with Microsoft Visual C++ 2008 Express Edition. The computer that was used during test runs has the following configuration: Intel Core i5-430M processor at 2.26 GHz with 4 GB RAM.

Open files A and B (like CapA.txt and CapB.txt)
Read \(m_A, n_A, m_B, n_B\) from first lines
Write \(m_A + m_B, n_A + n_B\)

for all facility \(i \ (i = 1, \ldots, mA)\) do
  Read capacity and cost of facility \(i\) from file A
  Write capacity, fixed cost of facility \(i\)
end for

for all facility \(i \ (i = 1, \ldots, mB)\) do
  Read capacity and cost of facility \(i\) from file B
  Write capacity, fixed cost of facility \(i\)
end for

for all customer \(j \ (j = 1, \ldots, nA)\) do
  Read demand from file A and file B
  Write sum of demand A and demand B
  for all facility \(i \ (i = 1, \ldots, mA)\) do
    Read cost \((j,i)\) from dataset A
    Write cost \((j,i)\)
  end for
end for

for all customer \(j \ (j = 1, \ldots, nB)\) do
  Read demand from file A and file B
  Write sum of demand A and demand B
  for all facility \(i \ (i = 1, \ldots, mB)\) do
    Read cost \((j,i)\) from dataset B
    Write cost \((j,i)\)
  end for
end for
Rewind to first customer line in file A and file B
Read demand from file A and file B
Write sum of demand A and demand B
For all facility i \((i = 1, \ldots, m_B)\) do
Read cost \((j, i)\) from dataset B
Write cost \((j, i)\)
End for
End for

Figure 4: Pseudocode for Combining Two Data Sets

In order to test the algorithms on problem data sets, a test application with a user interface which its screen shot is shown in Figure 5 is developed. On this form, for six different algorithms, PSO and ABC parameters are set to some default values. These parameters may be changed by the user. It is possible to load problem data sets from TXT files which have the format of OR-library (Beasley, 2005) files or from Microsoft Excel files with similar format.

Figure 5: User Interface of the Test Application

The performances are evaluated by three factors: average relative percent error (ARPE), hit to optimum rate (HR) and average computational processing time (ACPU) in milliseconds. ARPE is the percentage of deviation from the optimum and defined as follows:

\[
ARPE = \frac{\sum_{i=1}^{R} \left( \frac{H_i - U}{U} \right)}{R} \times 100
\]  

(13)
where \( H \) represents result of the \( i^{th} \) replication, \( U \) is value of optimal solution and \( R \) is the number of replications. \( HR \) shows the ratio between the number of successful runs and the total numbers of runs. If an experimental run resulted with the optimum value or 1% close to the optimum value, it is counted as successful.

Due to stochastic nature of the algorithms, many samples were produced in experiments. For each algorithm totally 21,000 trials which can be calculated as 7 experiment groups \( \times \) 3 replications \( \times \) 1000 iterations have been made.

CPSO, DPSO, CPSO_LiS, DPSO_LiS algorithms were executed 3 times up to 1000 iterations with 10 particles, where for all dimensions, initial particle positions were randomly set to a value between -10 and 10, initial particle velocities were randomly set to a value between -4 and 4. Cognitive parameter \( (c_1) \) was taken as 0.5 and social parameter \( (c_2) \) was taken as 0.5. Inertia weight was 0.9 and deceleration factor decay was 0.03.

ABC and ABC_LiS algorithms were executed, 3 times up to 1000 iterations with 10 employed bees, 10 onlooker bees and 1 scout bee. Food limit was taken as 250, which is one fourth of total number of iterations. Initially, artificial bee positions are randomly set to a value between -10 and 10 for each dimension.

For these three experiment groups which are shown in Table 2, best results achieved by each tested algorithm are shown on a chart in Figure 6. On the chart, it is observed that, CPSO, DPSO and ABC algorithms were not able to achieve the known optimal solutions. For OR-LIB problem CapA, three algorithms with local search have achieved known optimal result. For OR-LIB problem CapB, DPSO_LiS and ABC_LiS algorithms have achieved known optimal result. For OR-LIB problem CapC, only ABC_LiS algorithm has achieved known optimal result.
Table 2: Results for Files CapA, CapB and CapC

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>ARP E %</th>
<th>Hit rate</th>
<th>Best result</th>
<th>Average CPU time (msec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPSO</td>
<td>98.5%</td>
<td>0</td>
<td>26,650,917</td>
<td>5,127</td>
</tr>
<tr>
<td>DPSO</td>
<td>164.9%</td>
<td>0</td>
<td>34,303,823</td>
<td>6,861</td>
</tr>
<tr>
<td>ABC</td>
<td>25.6%</td>
<td>0</td>
<td>20,361,866</td>
<td>10,159</td>
</tr>
<tr>
<td>CPSO_L</td>
<td>2.41%</td>
<td>0</td>
<td>17,156,454</td>
<td>47,561</td>
</tr>
<tr>
<td>DPSO_L</td>
<td>1.26%</td>
<td>0</td>
<td>17,156,454</td>
<td>37,909</td>
</tr>
<tr>
<td>ABC_L</td>
<td>0.0%</td>
<td>1</td>
<td>17,156,454</td>
<td>49,309</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>ARP E %</th>
<th>Hit rate</th>
<th>Best result</th>
<th>Average CPU time (msec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPSO</td>
<td>54.4%</td>
<td>0</td>
<td>18,104,827</td>
<td>5,870</td>
</tr>
<tr>
<td>DPSO</td>
<td>71.1%</td>
<td>0</td>
<td>19,350,308</td>
<td>7,467</td>
</tr>
<tr>
<td>ABC</td>
<td>7.99%</td>
<td>0</td>
<td>13,939,214</td>
<td>9,594</td>
</tr>
<tr>
<td>CPSO_L</td>
<td>0.40%</td>
<td>1</td>
<td>12,979,071.58</td>
<td>4,188</td>
</tr>
<tr>
<td>DPSO_L</td>
<td>0.30%</td>
<td>0.6</td>
<td>12,979,071.58</td>
<td>49,477</td>
</tr>
</tbody>
</table>

Experiment group #1 - Test results for OR-LIB problem capA (100x1000)
Known optimal result: 17,156,454

Experiment group #3 - Test results for OR-LIB problem capC (100x1000)
Known optimal result: 11,505,594.33

Experiment group #2 - Test results for OR-LIB problem capB (100x1000)
Known optimal result: 12,979,071.58
The following four tests are made on artificially combined large problems. Since the optimal solutions were not calculated for these problems, it was not possible to calculate ARPE and HR measures.

For these four experiment groups which are shown in Table 3, best results achieved by each tested algorithm is shown on a chart in Figure 7.

Table 3: Experimental Results for Larger Files that are Combination of Files CapA, CapB, CapC

<table>
<thead>
<tr>
<th>Experiment group #4 - Test results for CapAB (200×2000) which is the combination of OR-LIB problems CapA (100×1000) and CapB (100×1000)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
</tr>
<tr>
<td>CPSO</td>
</tr>
<tr>
<td>DPSO</td>
</tr>
<tr>
<td>ABC</td>
</tr>
<tr>
<td>CPSO,LS</td>
</tr>
<tr>
<td>DPSO,LS</td>
</tr>
<tr>
<td>ABC,LS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Experiment group #5 - Test results for CapAC (200×2000) which is the combination of OR-LIB problems CapA (100×1000) and CapC (100×1000)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
</tr>
<tr>
<td>CPSO</td>
</tr>
<tr>
<td>DPSO</td>
</tr>
<tr>
<td>ABC</td>
</tr>
<tr>
<td>CPSO,LS</td>
</tr>
<tr>
<td>DPSO,LS</td>
</tr>
<tr>
<td>ABC,LS</td>
</tr>
</tbody>
</table>
Experiment group #6 - Test results for
CapBC (200×2000)
which is the combination of OR-LIB problems
CapB (100×1000) and CapC (100×1000)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Best result</th>
<th>Average CPU time (msec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPSO</td>
<td>35,803,861.78</td>
<td>30.902</td>
</tr>
<tr>
<td>DPSO</td>
<td>19,248,327.62</td>
<td>22.966</td>
</tr>
<tr>
<td>ABC</td>
<td>17,788,155.82</td>
<td>38.874</td>
</tr>
<tr>
<td>CPSO_LS</td>
<td>17,183,290.55</td>
<td>369.479</td>
</tr>
<tr>
<td>DPSO_LS</td>
<td>17,184,044.08</td>
<td>368.121</td>
</tr>
<tr>
<td>ABC_LS</td>
<td>17,166,151.78</td>
<td>377.760</td>
</tr>
</tbody>
</table>

Experiment group #7 - Test results for
CapABC(300×3000)
which is the combination of OR-LIB problems
CapA (100×1000), CapB (100×1000) and CapC (100×1000)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Best result</th>
<th>Average CPU time (msec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPSO</td>
<td>81,027,351.28</td>
<td>73.364</td>
</tr>
<tr>
<td>DPSO</td>
<td>23,578,024.35</td>
<td>50.809</td>
</tr>
<tr>
<td>ABC</td>
<td>23,055,375.35</td>
<td>95.845</td>
</tr>
<tr>
<td>CPSO_LS</td>
<td>22,123,623.68</td>
<td>1,299.781</td>
</tr>
<tr>
<td>DPSO_LS</td>
<td>22,139,760.80</td>
<td>1,361.905</td>
</tr>
<tr>
<td>ABC_LS</td>
<td>22,120,588.87</td>
<td>1,389.623</td>
</tr>
</tbody>
</table>

On the chart, it is observed that, performance of CPSO algorithm is far behind optimal value (or best result of other algorithms). Performance of ABC algorithm is always close to solutions of algorithms with local search. Three algorithms with local search produced best results which are very close to each other. In all experiments, except experiment group #4, ABC\_LS produced best results.

Figure 7: Best Result Achieved by Different Algorithms on CapAB, CapAC, CapBC and CapABC
5. Case Study

A fertilizer manufacturer company in Turkey sells own products and imported products. Production is made in one factory and imported products are received by marine lines at four different ports. Distribution from these locations to 768 different demand points throughout whole country is operated by company itself. Company needs to decide to introduce port locations among eleven alternative ports. Alternative port locations are at Yarımcıa, İzmir, İskenderun, Samsun, Tekirdağ, Bandırma, Çanakkale, Antalya, Mersin, Giresun and Rize. Totally 11 locations will be considered against 768 demand points.

From the company’s information system, recent data is extracted. Yearly demand quantities of each demand location and transportation costs between locations are given. ABC algorithm for UFL problem that was described in this paper is used to find the best possible solution which will result in least total transportation cost.

ABC and ABC\textsubscript{LS} algorithms were executed same number of times, 3 times up to 1000 iterations with 10 employed bees, 10 onlooker bees and 1 scout bee. Food limit was taken as 250, which is one fourth of total number of iterations. Initially, artificial bee positions are randomly set to a value between -10 and 10 for each dimension.

The best results found by these executions are shown in Table 4 below. Results are represented as yearly total transportation costs in Turkish Liras. It is observed that total transportation and operational costs constantly decreases as the allowed maximum number of port locations increases. This is natural, since distances gets smaller if there are closer facilities to demand points.

Results show that, best solution is achieved by a model with eight clusters. By considering these calculations, the best alternative is to operate 8 ports (Bandırma, İskenderun, Rize, Yarımcıa, Antalya, Tekirdağ, Çanakkale, İzmir). In this case, yearly total transportation cost would be decreased to 57,656,414 TRY.
Table 4: Test Results for Case Study (11×768)

<table>
<thead>
<tr>
<th>Max # opened facilities / Algorithm</th>
<th>Best result (TRY/year) / Avg. CPU time (msec)</th>
<th>Offered new port locations</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 ABC</td>
<td>62,643,706 / 286</td>
<td>İskenderun, Yarımca, Çanakkale, Giresun</td>
</tr>
<tr>
<td>4 ABC&lt;sub&gt;LS&lt;/sub&gt;</td>
<td>60,624,460 / 552</td>
<td>İskenderun, Yarımca, Antalya, Çanakkale</td>
</tr>
<tr>
<td>5 ABC</td>
<td>60,595,775 / 495</td>
<td>Rize, Yarımca, Antalya, Tekirdağ, İzmir</td>
</tr>
<tr>
<td>5 ABC&lt;sub&gt;LS&lt;/sub&gt;</td>
<td>58,985,248 / 853</td>
<td>İskenderun, Yarımca, Antalya, Tekirdağ, Çanakkale</td>
</tr>
<tr>
<td>6 ABC</td>
<td>58,499,273 / 727</td>
<td>İskenderun, Yarımca, Antalya, Tekirdağ, Çanakkale, İzmir</td>
</tr>
<tr>
<td>6 ABC&lt;sub&gt;LS&lt;/sub&gt;</td>
<td>58,305,613 / 1,293</td>
<td>Bandırma, Yarımca, Antalya, Tekirdağ, Çanakkale, İzmir</td>
</tr>
<tr>
<td>7 ABC</td>
<td>57,776,377 / 948</td>
<td>Bandırma, Mersin, Yarımca, Antalya, Tekirdağ, Çanakkale, İzmir</td>
</tr>
<tr>
<td>7 ABC&lt;sub&gt;LS&lt;/sub&gt;</td>
<td>57,749,368 / 1,511</td>
<td>Bandırma, İskenderun, Yarımca, Antalya, Tekirdağ, Çanakkale, İzmir</td>
</tr>
<tr>
<td>8 ABC</td>
<td>57,740,019 / 1,112</td>
<td>Bandırma, Mersin, Rize, Samsun, Yarımca, Tekirdağ, Çanakkale, İzmir</td>
</tr>
<tr>
<td>8 ABC&lt;sub&gt;LS&lt;/sub&gt;</td>
<td>57,656,414 / 1,653</td>
<td>Bandırma, İskenderun, Rize, Yarımca, Antalya, Tekirdağ, Çanakkale, İzmir</td>
</tr>
</tbody>
</table>
6. Conclusion

In this paper, we compared performances of CPSO, DPSO and ABC algorithms in solving uncapacitated facility location (UFL) problems. We have also applied local search along with these algorithms. Experiments were done on medium OR-Library problem data sets (100×1000) and large (200×2000, 300×3000) problem data sets that are produced in this study. Since the algorithms have a stochastic nature, average values for identical replications are shown in experiment results. Totally 126,000 runs have been conducted for six algorithms. For medium problem data sets, ABC produced better results both with or without local search. For large problem data sets, ABC without local search, still produced better results. But with local search, an attractive observation was that results of all three algorithms on large problem data sets were close to each other.

In experiment groups #1, #2 and #3, problem data sets (CapA, CapB and CapC) with size 100×1000 were used. Optimal solutions for these data sets were known. Hence it was possible to compare results of algorithms (CPSO, DPSO, ABC) in terms of Average Relative Percent Error (ARPE) which indicates how close the achieved solutions are to best known solution. In all three experiments, it is clearly observed that, ABC algorithm produced better results than CPSO and DPSO in terms of ARPE.

For experiment group #1 (CapA), CPSO has 98.58% and DPSO has 164.99%, where ABC has ARPE 25.67%. These results can be interpreted in a different way by dividing ARPE with each other, like, ABC produced 3.84 times closer results to the best know solution than result of CPSO. Result of ABC was 6.43 times better than result of DPSO.

For experiment group #2 (CapB), ARPE results of CPSO was 54.43%, DPSO was 71.18% and ABC was 7.99%. In other words, result of ABC was 6.81 times better than result of CPSO, 8.91 times better than result of DPSO.

For experiment group #3 (CapC), ARPE results of CPSO was 40.52%, DPSO was 45.54% and ABC was 13.44%. To express the same results as multiples, result of ABC was 3.01 times better than result of CPSO, 3.39 times better than result of DPSO.

In experiment groups #1, #2 and #3, problem data sets CapA, CapB and CapC were also tested by same algorithms augmented with local search
(CPSO$_{LS}$, DPSO$_{LS}$ and ABC$_{LS}$). In all cases, local search has improved the results. ABC$_{LS}$ found best known solution in all runs for CapA in experiment group #1. ARPE results for experiment group #1 of CPSO$_{LS}$ was 2.41%, DPSO$_{LS}$ was 1.26% and ABC$_{LS}$ was 0%. ARPE results for experiment group #2 of CPSO$_{LS}$ was 1.04%, DPSO$_{LS}$ was 0.40% and ABC$_{LS}$ was 0.30%. ARPE results for experiment group #3 of CPSO$_{LS}$ was 0.18%, DPSO$_{LS}$ was 0.89% and ABC$_{LS}$ was 0.16%.

In order to see the performance of the algorithms at huge size problems, four new problem data sets, three with size 200x2000 and one with size 300x3000 are generated by combining data sets CapA, CapB and CapC. Optimal solutions for combined data sets are not calculated in this study. For that reason, results are not comparable in terms of ARPE, but best results achieved by algorithms show that ABC still produces better results compared to CPSO and DPSO. Results of DPSO and ABC were much better than results of CPSO, like for experiment group #4 with CapAB, best result of CPSO was 61,314,297, best result of DPSO was 25,504,345 and best result of ABC was 20,958,644.

In experiment groups #4, #5, #6 and #7 with large problems, effect of local search is analysed. Interestingly, for each experiment, results of all three algorithms were close to each other. In experiment group #4, DPSO$_{LS}$ was better, in experiment group #5, all results were the same, in experiment group #6 and #7 ABC$_{LS}$ was better.

As a case study, ABC and ABC$_{LS}$ algorithms were used for solving an UFL problem of a fertilizer manufacturing company in Turkey. This company needs to decide which of 11 alternative port locations to use for distributing imported goods to 768 demand points throughout the country. The best solution is achieved by a model with eight clusters.

As a final note, most attracting conclusion is that in four experiment groups (#4, #5, #6 and #7) with large data sets, results produced by plain ABC were very close to results produced by CPSO$_{LS}$, DPSO$_{LS}$ and ABC$_{LS}$ with local search. This may be interpreted as power of ABC algorithm at large problems.

One drawback to underline is that CPU time of ABC was usually greater. In cases of implementing ABC algorithm to real world problems, programming tricks should be applied to reduce CPU time.
Artificial Bee Colony Optimization Algorithm 
for Uncapacitated Facility Location Problems

References


