A Discrete Artificial Bee Colony Algorithm for the Traveling Salesman Problem with Time Windows

Korhan Karabulut
Software Engineering Department
Yasar University
Selcuk Yasar Campus, Izmir, Turkey
korhan.karabulut@yasar.edu.tr

M. Fatih Tasgetiren
Industrial Engineering Department
Yasar University
Selcuk Yasar Campus, Izmir, Turkey
fatih.tasgetiren@yasar.edu.tr

Abstract—This paper presents a discrete artificial bee colony algorithm (DABC) for solving the traveling salesman problem with time windows (TSPTW) in order to minimize the total travel cost of a given tour. TSPTW is a difficult optimization problem arising in both scheduling and logistic applications. The proposed DABC algorithm basically relies on the destruction and construction phases of iterated greedy algorithm to generate neighboring food sources in a framework of ABC algorithm. In addition, it also relies on a classical 1-opt local search algorithm to further enhance the solution quality. The performance of the algorithm was tested on a benchmark set from the literature. Experimental results show that the proposed DABC algorithm is very competitive to or even better than the best performing algorithms from the literature.

Keywords—traveling salesman problem with time windows; artificial bee colony algorithm; iterated greedy algorithm; swarm intelligence; heuristic optimization.

I. INTRODUCTION

The TSPTW deals with finding a minimum cost tour starting and ending on a given depot and visiting a set of customers in such a way that each customer has a service time and a time window defining its ready time and due date. Each customer must be visited before its due date for a feasible tour otherwise tour is considered as infeasible. On the other hand, if the vehicle arrives before the customer ready time, it must wait. The TSPTW can be modeled as routing and scheduling tasks. As to routing tasks, its objective is to find an efficient route to visit a number of customers, starting and ending at a depot with the constraint that each customer must be visited in a time window. Typically, in this case, the objective function is the cost of a tour which is in fact the total distance traveled. In addition to above, the TSPTW is equivalent to the single machine scheduling with sequence dependent setup times where each job has a release and due date. In this case, the objective function is to minimize the makespan.

The TSPTW is shown to be $NP$-hard, as well as, even finding a feasible solution is an $NP$-complete problem [1]. The literature contains studies on makespan optimization [2, 3], as well as, optimization of total distance traveled. Langevin et al. [4] uses a two-commodity flow formulation that can be used for both makespan and travel-cost optimization. Dumas et al. [5] extended earlier dynamic programming approaches being able to solve instances up to 200 customers. Pesant et al. [6] and Focacci et al. [7] developed exact methods by applying constraint programming. More recently, Ascheuer et al. [8] considered a branch-and-cut algorithm that was designed for the asymmetric TSPTW. Balas and Simonetti [9] developed a dynamic programming algorithm for various TSP variants with including the TSPTW.

Due to the $NP$-hardness nature of the problem, heuristic and meta-heuristic approaches were also considered. Carlton and Barnes [10] developed a tabu search approach. Gendreau et al. [11] presented a insertion heuristic. Calvo [12] used a construction heuristic. Ohlmann and Thomas [13] proposed a compressed annealing (CA) algorithm. Most recently, [29] uses a Beam-ACO which is a hybrid method combining ant colony optimization with beam search and [30] proposes a general VNS heuristic composed of construction and optimization stages. These two studies are used for comparison with our results.

This remaining part of the paper is organized as follows: Section II gives the problem formulation. Section III is devoted to the details of the DABC algorithm. The computational results for the test problems are given in Section IV. Section V provides concluding remarks.

II. PROBLEM FORMULATION

Given an undirected complete graph $G = (N, A)$, $N = \{0, 1, \ldots, n\}$ is a set of nodes representing the depot (node 0) and $n$ customers; and $A = N \times N$ is a set of edges connecting the nodes where a solution is a tour represented as $\pi = \{\pi_0, \pi_1, \ldots, \pi_n, \pi_{n+1}\}$. $\pi_0 = \pi_{n+1} = 0$ since the tour starts and ends at the depot, and $\{\pi_1, \pi_2, \ldots, \pi_n\}$ is a permutation of the nodes in $N \setminus \{0\}$ in which $\pi_k$ denotes the index of the customer at the $k$th position of the tour.

$c(i, j)$ is the cost of travel time between customers $i$ and $j$, plus the service time at customer $i$. Each node $i \in N$ has a time window $[e_i, l_i]$ which specifies the ready time and the due date. Waiting time is permitted, that is, a node $i$ can be reached before the start of its time window $e_i$, but cannot be left before $e_i$. 

U.S. Government work not protected by U.S. copyright
In this paper we focus on the minimization of the travel cost. The objective is to find a Hamiltonian tour starting and ending at the depot and minimizing the total traveled distance while visiting each city within its time window.

Given this objective, the TSPTW can be formulated as follows:

\[
\begin{align*}
\min f(\pi) &= \sum_{k=0}^{n} c(\pi_k, \pi_{k+1}) \\
\text{St:} &\quad v(\pi) = \sum_{k=0}^{n+1} \omega(\pi_k) = 0 \\
&\quad \omega(\pi_k) = \begin{cases} 1 & \text{if } A_{\pi_k} > l_{\pi_k} \\ 0 & \text{otherwise} \end{cases}
\end{align*}
\]

where \( i = 1, \ldots, NP \) and \( j = 1, \ldots, n \); \( NP \) is the number of food sources; \( n \) is the number of decision variable to be optimized and \( r \) is a uniform random number between 0 and 1. In addition, counters storing the numbers of trials of solutions are set to 0 in this phase. After initialization, the population of the food sources (solutions) repeats cycles of the search processes of the employed bees, the onlooker bees and the scout bees. Termination criteria for the ABC algorithm might be reaching a maximum cycle number (MCN), meeting an error tolerance (\( e \)) or a pre-defined CPU time.

As mentioned before, each employed bee is associated with only one food source site. For this reason, the size of the employed bees is the number of food sources. An employed bee generates a neighboring solution by making a modification on the position of the food; finds a neighboring food source, and then evaluates its quality. In ABC, finding a neighboring food source is defined as follows:

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

Inside the neighborhood of every food source \( x_i \), a food source \( v_i \) is produced by modifying only a single parameter of \( x_i \). In equation (5), \( j \) is a random integer in the range \( [1, n] \) and \( k \in \{1, \ldots, NP\} \) is a randomly chosen index that must be different from \( i \). \( \Theta \) is a uniform random number in the range \([-1, 1]\).

The equation (5) implies that as the difference between the parameters of \( x_{ij} \) and \( x_{kj} \) decreases, the perturbation on the \( x_{ij} \) vector will be decreased too. For this reason, the step size will be adaptively reduced as the search gets closer to the optimal solution in the search space. If any parameter value generated is out of its pre-determined boundaries, the parameter can be set to an acceptable value by using equation (4) again.

After producing \( v_i \) within the boundaries, a fitness value for a minimization problem can be assigned to the solution \( v_i \) as follows:

\[
\text{fitness}_i = \begin{cases} 
\frac{1}{1 + f_i} & \text{if } f_i \geq 0 \\
1 + \text{abs}(f_i) & \text{if } f_i < 0 
\end{cases}
\]

where \( f_i \) is the objective function value of the solution \( v_i \). A greedy selection is employed between \( x_i \) and \( v_i \) such that the better one is selected depending on fitness values representing the nectar amount of the food sources at \( x_i \) and \( v_i \). If \( x_i \) cannot be improved, its counter is incremented by 1, otherwise, the counter is reset to 0.

Once all employed bees finish their searches, they share their information with the onlooker bees on the dance area. An onlooker bee considers the nectar information taken from all employed bees and selects a food source with a probability related to its nectar amount. This stochastic selection is based on the fitness values of the solutions in the population. A fitness-based selection scheme might be a roulette wheel, ranking based, stochastic sampling, tournament selection etc.
In the basic ABC, roulette wheel selection is employed where each food source has a probability proportional to its fitness value and the total fitness value as follows:

$$p_i = \frac{\text{fitness}_i}{\sum_{i=1}^{n} \text{fitness}_i} \quad (7)$$

By doing so, the onlooker bees are encouraged to visit the food sources that have higher nectar amounts. Briefly, food source selection by onlookers is based on the information provided by employed bees. A uniform random number within the range $[0, 1]$ is obtained for each source. If the probability value $p_i$ is greater than this random number then the onlooker bee generates a neighboring food source again by using equation (5) as in the case of the employed bee. After the source is evaluated, greedy selection is applied. If solution $x_i$ cannot be improved, its counter is incremented by 1, otherwise, the counter is reset to 0. This process is repeated until all onlookers are distributed onto food source sites.

In each cycle, once all employed bees and onlooker bees finish their searches, the algorithm determines whether or not any food source is to be abandoned by using the counters. If the value of the counter is greater than the control parameter ‘limit’, then the food source associated with this counter is abandoned. The food source abandoned is replaced with a new food source discovered by the scout, which provides diversification mechanism to the ABC algorithm. Assume that the abandoned source is $x_i$, then the scout randomly discovers a new food source to be replaced with $x_i$ which can be achieved by the equation (4). In the basic ABC, there is only a single food source which can be abandoned in each cycle, and only one employed bee can be a scout.

The above ABC algorithm, originally designed for the continuous nature of optimization problems, cannot directly be used for discrete/combinatorial problems. In this study, we implement some modifications to the above ABC algorithm to handle discrete decision variables. The details of the modifications are as discussed below

A. Initial Population

The permutation based representation is used in the DABC algorithm and a solution is represented by a permutation of nodes $\pi = \{\pi_0, \pi_1, \ldots, \pi_i, \ldots, \pi_n\}$. For the initial population in the DABC algorithm, solutions in the population are constructed randomly. Since we generate the neighboring food source by using the destruction and construction DC procedure of the iterated greedy (IG) algorithm [19], each food source in the initial population is assigned to a destruction size $d_i$ which is randomly generated within the range $[1, n - 1]$.

The key procedure in the DABC algorithm is the destruction and construction heuristic, which is given in Fig. 1. In the destruction step, a given number $d$ of nodes, randomly chosen and without repetition, are removed from the solution, thus resulting in two partial solutions. The first one with the size $d$ of nodes is denoted as $\pi^R$ and includes the removed nodes in the order in which they are removed. The second one with the size $n - d$ of nodes is the original solution without the removed nodes, which is denoted as $\pi^D$. Finally, the construction phase requires a constructive heuristic procedure. We employ the NEH insertion heuristic of [20]. In order to reinsert nodes in $\pi^R$ into the destructed solution $\pi^D$, the first nodes $\pi^D_k$ is inserted into all possible $n - d$ positions in the destructed solution $\pi^D$ generating $n - d$ partial solutions. Among these $n - d$ partial solutions including nodes $\pi^R_k$, the best partial solution with the minimum total travel cost is chosen and kept for the next iteration. Then the second node $\pi^D_k$ is considered and so on until $\pi^D$ is empty or a final solution is obtained. Hence $\pi^D$ is again of size $n$. For the details of the IG algorithm, we refer to [19] for details. Any food source can represent a feasible solution if the equation (2) is satisfied. In case of infeasibility, some sophisticated constraint handling methods are used to handle the infeasibility as explained in the next sections.

Procedure DC($\pi, d_i$)

$\pi^D \leftarrow \text{Destruct}(\pi, d_i, \pi^R)$

$\pi \leftarrow \text{Construct}(\pi^D, \pi^R)$

Return $\pi$

Figure 1. Destruction and construction procedure.

B. Employed bee phase

According to the basic ABC algorithm, the employed bees generate food sources in the neighborhood of their current positions. Since each food source has its own destruction size of $d_i$, the neighboring food source is generated by using the DC procedure with an associated $d_i$ value. After generating a neighboring food source, a $1_{opt}$ local search is applied to further improve the solution quality (nectar amount). As for the selection, new food source is always accepted if it is better than the current food source (according to SF, NFT and $\epsilon$-constraint level that will be described later) which is the same as in the basic ABC algorithm which carries out a greedy selection procedure. The local search procedure will be explained in detail in next sections.

C. Onlooker bee phase

In the basic ABC algorithm, an onlooker bee selects a food source $\pi_k$ depending on its winning probability value which is similar to the wheel selection in GAs. However, the tournament selection is widely used in GA applications due to its simplicity and ability to escape from local optima [22]. For this reason, we determine the food source by a tournament selection with the size of $2$ where an onlooker bee selects a food source $\pi_k$ in such a way that two food sources are randomly picked up from the population, and compared to each other, allowing the better one to be chosen. An onlooker bee utilizes its own destruction size $d_k$ to produce a new neighboring solution. Then a $1_{opt}$ local search is employed to further improve the nectar amount of the onlooker bee. If the new food source obtained is better than or equal to the current one, the new food source will replace the current one.
and become a new member in the population (according to SF, NFT and \(\varepsilon\)-constraint level that will be described later). The size of the onlooker bees is NP individuals. The local search procedure will be explained in detail in the next sections.

D. Scout bee phase

In the basic ABC algorithm, a scout bee produces a food source randomly in the predefined search space. This will decrease the search efficacy, since the best food source in the population often carries better information than others during the search process, and the search space around it could be the most promising region. Therefore, in the DABC algorithm, a tournament selection with the size of two is again used to discard a solution in such a way that two random food sources are picked up from the population and the worst one is selected. Then the scout generates a food source by performing a DC with the best destruction size \(d_{\text{best}}\) to the best solution in the population and replaced with the food source determined by tournament selection. The size of the scout bees is 0.2 \(\times\) NP individuals.

E. Local Search

The local search employed in this work is based on the 1-opt neighborhood. In the 1-opt local search, a single customer is removed from the tour and reinserted in different positions in a forward and backward manner sequentially. In both procedures, a customer is removed and reinserted into \(n-1\) positions in a way that if the insertion point is feasible. In other words, local search phase of the algorithm only considers the feasible movements. The outline of the 1-opt local search is given in Figure 1.

**Procedure 1_opt**

**Input:** \((\pi = (\pi_1, \pi_2, \ldots, \pi_n))\)

//Backward pass

for \(i \leftarrow n\) to 2 do

remove \(\leftarrow d(\pi_{i-2}, \pi_i) + d(\pi_i, \pi_{i+1}) - d(\pi_{i-1}, \pi_{i+1})\)

for \(j \leftarrow i-1\) to 1 do

add \(\leftarrow d(\pi_{j-1}, \pi_i) + d(\pi_i, \pi_j) - d(\pi_j, \pi_{j+1})\)

if (remove > add) and \((is\ feasible(\pi_i, \pi_j))\)

\(\pi^* \leftarrow Insert(\pi_i, \pi_j, \pi)\)

if \(f(\pi^*) < f(\pi)\) then

\(\pi \leftarrow \pi^*\)

end if

end for

//Forward pass

for \(i \leftarrow 1\) to \(n-1\) do

remove \(\leftarrow d(\pi_{i-1}, \pi_i) + d(\pi_i, \pi_{i+1}) - d(\pi_{i-2}, \pi_{i+1})\)

for \(i \leftarrow i+1\) to \(n\) do

add \(\leftarrow d(\pi_{i-1}, \pi_i) + d(\pi_i, \pi_j) - d(\pi_j, \pi_{j+1})\)

if (remove > add) and \((is\ feasible(\pi_i, \pi_j))\)

\(\pi^* \leftarrow Insert(\pi_i, \pi_j, \pi)\)

if \(f(\pi^*) < f(\pi)\) then

\(\pi \leftarrow \pi^*\)

end if

end if

end for

end for

Figure 2. 1_opt local search.

F. Constraint handling

Swarm intelligence operators may generate infeasible solutions. In this case, care must be taken with them violating the constraints. There exist different approaches to handle the constraints [21]. In this paper, three sophisticated approaches are employed to handle the constraints which are summarized below:

Deb [22] was the first to propose the superiority of feasible solutions (SF) for constrained optimization based on lexicographic ordering where constraint violation and objective function value are distinguished. The aim is to optimize both constraint violation and objective function by a lexicographic order where constraint violation precedes objective function value. In SF, when two solutions \(\pi_a\) and \(\pi_b\) are evaluated, \(\pi_a\) is deemed to be superior to \(\pi_b\) under the following conditions for a minimization problem:

- \(\pi_a\) is feasible and \(\pi_b\) is not.
- \(\pi_a\) and \(\pi_b\) are both feasible and \(\pi_a\) has a smaller objective function value than \(\pi_b\).
- \(\pi_a\) and \(\pi_b\) are both infeasible, but \(\pi_a\) has a smaller overall constraint violation \(v(\pi)\) as computed by using the equation (2).

The adaptive penalty approach is presented in [23] where the notion of a “near feasibility threshold” (NFT) corresponds to a “promising region” beyond the feasible region. The NFT is defined as a threshold distance from feasible region such that the search within feasible region and the NFT-neighborhood of the feasible region is encouraged whereas it is discouraged beyond that threshold. Additionally, an adaptive term is added to the penalty function to consider the gap between the best feasible value and best infeasible value found so far. Then the adaptive penalty function is defined for the \(m\) number of constraints as:

\[
\begin{align*}
    f_p(\pi) = f(\pi) + (f_{\text{feas}} - f_{\text{all}}) \sum_{i=1}^{m} \left(\frac{v(\pi)}{NFT}\right)^\alpha \\
\end{align*}
\]

where \(f_{\text{all}}\) denotes the un-penalized value of the best solution found yet and \(f_{\text{feas}}\) denotes the value of the best feasible solution yet found. As noted in [24], the adaptive term may lead to zero or over-penalty. For this reason, only the dynamic part of the above penalty function with NFT threshold is used in this paper as follows:

\[
\begin{align*}
    f_p(\pi) = f(\pi) + \left(\frac{v(\pi)}{NFT}\right)^\alpha \\
\end{align*}
\]

The general form of the NFT is given by \(NFT = \frac{NFT_0}{1 + t^\gamma}\) where \(NFT_0\) is an upper bound for the NFT; \(\gamma\) is a user-defined positive parameter; and \(t\) is the generation counter.

The \(\varepsilon\)-constrained method (EC) was developed in [25-28] based on lexicographic ordering with relaxation of the constraints. In the \(\varepsilon\)-constraint handling method, the relaxation of the constraints is controlled by using the \(\varepsilon\) parameter. The
proper control of the $\epsilon$ parameter is essential to obtain high quality solutions for problems with equality constraints [25-28]. The $\epsilon$ level is updated until the generation counter $t$ reaches the control generation $t_C$. After the generation counter exceeds $t_C$, the $\epsilon$ level is set to zero to obtain solutions with no constraint violation. The idea behind EC method is that solutions with the total violation less that $\epsilon(t)$ are treated as feasible solutions when comparing two solutions. The $\epsilon$ - constraint handling method can be summarized as follows:

\[
\epsilon(0) = v(\pi_0) \quad (10)
\]

\[
\epsilon(t) = \begin{cases} 
\epsilon(0) \left(1 - \frac{t}{t_C}\right)^{cp} & 0 < t < t_C \\
0 & t > t_C 
\end{cases} \quad (11)
\]

Where $\pi_0$ is the top $\theta$-th individual and $\theta = 0.25 \times NP$. The recommended parameter ranges are $\theta = 0.25N$, $t_C = (0.1 t_{\text{max}}, 0.8 t_{\text{max}})$ and $cp = (2, 10)$. The complete computational procedure of the ABC algorithm is given in Figure 3.

Step 1. Set parameters
Step 2. Establish initial population randomly
Step 3. Assign a destruction size $d_i$ to each food source in the population randomly between 1 and $n - 1$
Step 4. Evaluate population and find $\pi_{\text{best}}$ and $d_{\text{best}}$
Step 5. Repeat the following for each employed bee $\pi_i$
   (Employed Bee Phase)
   a. Generate a new food source by $\pi_{\text{new}} = DC(\pi_i, d_i)$
   b. Apply 1_opt local search to $\pi_{\text{new}}$
   c. if $f(\pi_{\text{new}}) < f(\pi_i)$, $\pi_i = \pi_{\text{new}}$
   d. if $f(\pi_{\text{new}}) < f(\pi_{\text{best}})$, $\pi_{\text{best}} = \pi_{\text{new}}$ and $d_{\text{best}} = d_i$
Step 6. Repeat the following for each onlooker bee $\pi_k$
   (Onlooker Bee Phase)
   a. Select a food source by $\pi_k = TS_{\text{worst of } z}(\pi_k \in \text{NP})$
   b. Generate a new food source by $\pi_{\text{new}} = DC_k(\pi_k, d_k)$
   c. Apply 1_opt local search to $\pi_{\text{new}}$
   d. if $f(\pi_{\text{new}}) < f(\pi_k)$, $\pi_k = \pi_{\text{new}}$
   e. if $f(\pi_{\text{new}}) < f(\pi_{\text{best}})$, $\pi_{\text{best}} = \pi_{\text{new}}$ and $d_{\text{best}} = d_k$
Step 7. Repeat the following for each scout bee $\pi_k$
   (Scout Bee Phase)
   a. Select a food source by $\pi_k = TS_{\text{worst of } z}(\pi_k \in \text{NP})$
   b. Generate a new food source by $\pi_{\text{new}} = DC_k(\pi_k, d_k)$
   c. Apply 1_opt local search to $\pi_{\text{new}}$
   d. if $f(\pi_{\text{new}}) < f(\pi_k)$, $\pi_k = \pi_{\text{new}}$
   else $d_k = \text{rand}(\%)(n - 1)$
   e. if $f(\pi_{\text{new}}) < f(\pi_{\text{best}})$, $\pi_{\text{best}} = \pi_{\text{new}}$
   f. If the stopping criterion is not met, go to Step 5, else stop and return $\pi_{\text{best}}$

It is important to note that we used these three constraint handling methods in the proposed algorithm as follows: If a food source is infeasible, it is penalized by NFT. Before making a selection between two food sources, each food source is treated to be either feasible or infeasible according to the $\epsilon$ level (i.e. if the overall violation of the individual is less than $\epsilon(t)$, it is treated to be feasible). Then, SF is used when making a selection between two food sources. In the case of local search, only NFT is used.

IV. COMPUTATIONAL RESULTS

We implemented DABC algorithm in C++ and conducted all experiments on a computer with an Intel Core 2 Quad processor with 2.66 GHz CPU. We compare the performance of DABC with the results of two recent best performing algorithms from literature. The first one is based on the beam search hybridized with the ant colony algorithm (BACO) in [29] and the second one is using a global variable neighborhood search (GVNS) in [30]. We restricted the time limit to 60 seconds per run and per instance. We carried out 15 independent runs for each instance and provide the average objective function value as well as the CPU time. We consider two available sets of benchmark instances where $n$ represents the number of customers; $w$ is the time window width and $Best$ is the best known or optimal value:

1. The first benchmark set consists of 27 classes of five instances each. All instances were proposed and solved to optimality by Dumas et al. [5]. Instance size ranges from 20 to 200 customers.
2. Gendreau et al. [11] provided the second benchmark set consisting of 120 instances grouped into 26 classes with equal number of customers and time window width. These instances were obtained from the instances proposed by Dumas et al. [5] by extending the time windows by 100 units, resulting in time windows in the range from 120 to 200 time units.

The computational results of the DABC algorithm for Dumas’s instances are given in Table I. The statistics over 15 replications to each instance are averaged over the five instances of each instance class. In terms of solution quality,
the DABC algorithm. The computational results of the DABC algorithm for Gendreau et al.’s instances are given in Table II. These instances have wide time windows and exact algorithms have problems finding even feasible solutions in a reasonable computation time. As can be seen in Table II, the performance of the BACO algorithm degrades as the problem size gets larger. Both DABC and GVNS algorithm were superior to the BACO algorithm for instances starting from 80x100 instances to 100x200 instances since they were able to further improve the best known solutions for this difficult class of the problem.

It is clear from Table II that both DABC and GVNS algorithms were superior to the BACO algorithm. Ultimately, the best known solutions for the instances of 80x100, 80x120, 80x140, 80x160, 80x180, 80x200, 100x120, 100x140, 100x160, 100x180, 100x200 are further improved by the DABC algorithm. Note that GVNS algorithm was able to do that too. DABC was able to find a better solution than GVNS in the 100x100 instance. The extensive evaluation of 5 other benchmark sets is also given by the BACO algorithm in [29]. We would like to mention that the DABC algorithm was also very competitive to or even better than the BACO algorithm for those 5 other benchmark instances.

For the future work, the challenging new benchmark suite developed in [30] and solved by the GVNS algorithm will be considered. These instances range from 200 nodes to 400 nodes as well as from the time window width from 100 to 500. Authors work on this benchmark suite for a future work.

V. CONCLUSIONS

This paper presents a discrete artificial bee colony algorithm for solving the traveling salesman problem with time windows in order to minimize the total travel cost of a given tour. TSPTW is a difficult optimization problem arising in both scheduling and logistics applications. The proposed DABC algorithm basically relies on the destruction and construction phases of iterated greedy algorithm to generate neighboring food sources in a framework of ABC algorithm. In addition, it also relies on a classical 1-opt local search algorithm to further enhance the solution quality. The performance of the algorithm was tested on a benchmark set from the literature. Experimental results show that the proposed DABC algorithm is very competitive to or even better than the best performing algorithms from the literature. Ultimately, the best known solution for the instance of 100x100 from Gendreau et al.’s data set is further improved by the DABC algorithm.


### Table I. Results for Instances by Dumas

<table>
<thead>
<tr>
<th>$n$</th>
<th>$w$</th>
<th>Best ABC</th>
<th>CPU</th>
<th>GVNS CPU</th>
<th>BACO CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>20</td>
<td>361.2</td>
<td>0.0</td>
<td>361.2</td>
<td>0.0</td>
</tr>
<tr>
<td>20</td>
<td>40</td>
<td>316.0</td>
<td>0.0</td>
<td>316.0</td>
<td>0.0</td>
</tr>
<tr>
<td>20</td>
<td>60</td>
<td>309.8</td>
<td>0.0</td>
<td>309.8</td>
<td>0.0</td>
</tr>
<tr>
<td>20</td>
<td>80</td>
<td>311.0</td>
<td>0.0</td>
<td>311.0</td>
<td>0.0</td>
</tr>
<tr>
<td>20</td>
<td>100</td>
<td>275.2</td>
<td>0.0</td>
<td>275.2</td>
<td>0.0</td>
</tr>
<tr>
<td>20</td>
<td>200</td>
<td>486.6</td>
<td>0.0</td>
<td>486.6</td>
<td>0.0</td>
</tr>
<tr>
<td>20</td>
<td>400</td>
<td>461.0</td>
<td>0.0</td>
<td>461.0</td>
<td>0.0</td>
</tr>
<tr>
<td>20</td>
<td>600</td>
<td>416.4</td>
<td>0.0</td>
<td>416.4</td>
<td>0.0</td>
</tr>
<tr>
<td>20</td>
<td>800</td>
<td>399.8</td>
<td>0.0</td>
<td>399.8</td>
<td>0.0</td>
</tr>
<tr>
<td>20</td>
<td>1000</td>
<td>377.0</td>
<td>0.0</td>
<td>377.0</td>
<td>0.0</td>
</tr>
<tr>
<td>20</td>
<td>2000</td>
<td>581.6</td>
<td>0.0</td>
<td>581.6</td>
<td>0.0</td>
</tr>
<tr>
<td>20</td>
<td>4000</td>
<td>590.2</td>
<td>0.0</td>
<td>590.2</td>
<td>0.0</td>
</tr>
<tr>
<td>20</td>
<td>6000</td>
<td>560.0</td>
<td>0.0</td>
<td>560.0</td>
<td>0.0</td>
</tr>
<tr>
<td>20</td>
<td>8000</td>
<td>508.0</td>
<td>0.0</td>
<td>508.0</td>
<td>0.0</td>
</tr>
<tr>
<td>20</td>
<td>10000</td>
<td>514.8</td>
<td>1.8</td>
<td>514.8</td>
<td>1.3</td>
</tr>
<tr>
<td>20</td>
<td>20000</td>
<td>676.6</td>
<td>0.4</td>
<td>676.6</td>
<td>0.9</td>
</tr>
<tr>
<td>20</td>
<td>40000</td>
<td>630.0</td>
<td>1.8</td>
<td>630.0</td>
<td>1.3</td>
</tr>
<tr>
<td>20</td>
<td>60000</td>
<td>606.4</td>
<td>1.8</td>
<td>606.4</td>
<td>1.8</td>
</tr>
<tr>
<td>20</td>
<td>80000</td>
<td>593.8</td>
<td>5.8</td>
<td>593.8</td>
<td>2.1</td>
</tr>
<tr>
<td>20</td>
<td>100000</td>
<td>757.6</td>
<td>2.8</td>
<td>757.6</td>
<td>1.4</td>
</tr>
<tr>
<td>20</td>
<td>200000</td>
<td>701.8</td>
<td>1.8</td>
<td>701.8</td>
<td>1.9</td>
</tr>
<tr>
<td>20</td>
<td>400000</td>
<td>696.6</td>
<td>2.0</td>
<td>696.6</td>
<td>2.7</td>
</tr>
<tr>
<td>20</td>
<td>800000</td>
<td>868.4</td>
<td>6.6</td>
<td>868.4</td>
<td>3.6</td>
</tr>
<tr>
<td>20</td>
<td>1600000</td>
<td>834.8</td>
<td>8.2</td>
<td>834.8</td>
<td>5.3</td>
</tr>
<tr>
<td>20</td>
<td>3200000</td>
<td>805.0</td>
<td>22.6</td>
<td>810.6</td>
<td>7.4</td>
</tr>
<tr>
<td>20</td>
<td>6400000</td>
<td>809.0</td>
<td>56.0</td>
<td>809.0</td>
<td>18.0</td>
</tr>
<tr>
<td>20</td>
<td>12800000</td>
<td>984.2</td>
<td>984.2</td>
<td>12.6</td>
<td>984.2</td>
</tr>
</tbody>
</table>

### Table II. Comparison on Gendreau Benchmarks

<table>
<thead>
<tr>
<th>$n$</th>
<th>$w$</th>
<th>Best ABC</th>
<th>CPU</th>
<th>GVNS CPU</th>
<th>BACO CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>120</td>
<td>265.6</td>
<td>0.0</td>
<td>265.6</td>
<td>0.3</td>
</tr>
<tr>
<td>20</td>
<td>140</td>
<td>232.8</td>
<td>0.0</td>
<td>232.8</td>
<td>0.3</td>
</tr>
<tr>
<td>20</td>
<td>160</td>
<td>218.2</td>
<td>0.0</td>
<td>218.2</td>
<td>0.3</td>
</tr>
<tr>
<td>20</td>
<td>180</td>
<td>236.6</td>
<td>0.0</td>
<td>236.6</td>
<td>0.4</td>
</tr>
<tr>
<td>20</td>
<td>200</td>
<td>241.0</td>
<td>0.0</td>
<td>241.0</td>
<td>0.4</td>
</tr>
<tr>
<td>20</td>
<td>220</td>
<td>360.0</td>
<td>0.0</td>
<td>377.8</td>
<td>0.8</td>
</tr>
<tr>
<td>20</td>
<td>240</td>
<td>348.4</td>
<td>0.0</td>
<td>364.4</td>
<td>0.8</td>
</tr>
<tr>
<td>20</td>
<td>260</td>
<td>326.8</td>
<td>0.0</td>
<td>326.8</td>
<td>0.9</td>
</tr>
<tr>
<td>20</td>
<td>280</td>
<td>326.8</td>
<td>0.0</td>
<td>330.4</td>
<td>1.0</td>
</tr>
<tr>
<td>20</td>
<td>300</td>
<td>313.8</td>
<td>0.6</td>
<td>313.8</td>
<td>1.0</td>
</tr>
<tr>
<td>20</td>
<td>320</td>
<td>451.0</td>
<td>0.8</td>
<td>451.0</td>
<td>1.5</td>
</tr>
<tr>
<td>20</td>
<td>340</td>
<td>452.0</td>
<td>0.6</td>
<td>452.0</td>
<td>1.7</td>
</tr>
<tr>
<td>20</td>
<td>360</td>
<td>448.6</td>
<td>0.8</td>
<td>464.0</td>
<td>1.7</td>
</tr>
<tr>
<td>20</td>
<td>380</td>
<td>421.2</td>
<td>3.4</td>
<td>421.2</td>
<td>2.2</td>
</tr>
<tr>
<td>20</td>
<td>400</td>
<td>427.4</td>
<td>2.4</td>
<td>427.4</td>
<td>2.4</td>
</tr>
<tr>
<td>20</td>
<td>420</td>
<td>578.6</td>
<td>3.8</td>
<td>578.6</td>
<td>2.3</td>
</tr>
<tr>
<td>20</td>
<td>440</td>
<td>541.4</td>
<td>3.0</td>
<td>541.4</td>
<td>2.7</td>
</tr>
<tr>
<td>20</td>
<td>460</td>
<td>506.0</td>
<td>6.4</td>
<td>506.0</td>
<td>3.2</td>
</tr>
<tr>
<td>20</td>
<td>480</td>
<td>504.8</td>
<td>3.6</td>
<td>504.8</td>
<td>3.3</td>
</tr>
<tr>
<td>20</td>
<td>500</td>
<td>489.0</td>
<td>8.0</td>
<td>500.6</td>
<td>3.7</td>
</tr>
<tr>
<td>20</td>
<td>520</td>
<td>481.4</td>
<td>8.8</td>
<td>481.4</td>
<td>4.2</td>
</tr>
<tr>
<td>20</td>
<td>540</td>
<td>666.4</td>
<td>3.0</td>
<td>666.4</td>
<td>3.1</td>
</tr>
<tr>
<td>20</td>
<td>560</td>
<td>642.0</td>
<td>11.8</td>
<td>642.0</td>
<td>3.7</td>
</tr>
<tr>
<td>20</td>
<td>580</td>
<td>597.2</td>
<td>15.8</td>
<td>597.2</td>
<td>4.1</td>
</tr>
<tr>
<td>20</td>
<td>600</td>
<td>548.4</td>
<td>10.0</td>
<td>548.4</td>
<td>4.4</td>
</tr>
<tr>
<td>20</td>
<td>620</td>
<td>555.0</td>
<td>11.8</td>
<td>555.0</td>
<td>5.1</td>
</tr>
<tr>
<td>20</td>
<td>640</td>
<td>561.6</td>
<td>12.2</td>
<td>561.6</td>
<td>6.3</td>
</tr>
<tr>
<td>20</td>
<td>660</td>
<td>550.2</td>
<td>37.4</td>
<td>550.2</td>
<td>6.8</td>
</tr>
</tbody>
</table>