The following sections present and explain the pseudo-codes and genetic operators in detail.

**Listing 1.** Pseudo code of proposed GA with random forest algorithm

**input:** population size , iteration number , mutation rate , crossover rate , number of scenarios (), features, increment percentage , number of trees , maximum depth of the tree , minimum samples split , minimum samples leaf , elimination percentage

**begin**

Initialize population randomly;

Compute initial fitness function values;

Extract features and train initial Random Forest model;

; (Iteration counter)

**while** **do**

Perform **elitism**;

Perform **roulette wheel** **selection** to generate mating pool ;

Perform **crossover** with ;

Predict fitness values using **Random Forest** model;

Filter out worst-performing individuals () based on predicted fitness;

Update population with filtered individuals;

Perform **mutation** with ;

Compute fitness function values;

Retrain **Random Forest** model with the best () individuals; (**for Incremental learning**)

Train **Random Forest** model with the updated population; (**for Non-incremental learning**)

;

**end while**

Return best solution;

**end**

**output:** Best solution

**Listing 2.** Pseudo code of random forest training

**input:** training data (), training fitness (), number of trees , maximum depth of the tree , minimum samples split , minimum samples leaf

**begin**

Initialize forest as an empty list;

**for** **do**

Sample data with replacement to create bootstrap sample (TD, TF);

Train a decision tree on (TD, TF);

Add the trained tree to the forest;

**End for**

Return forest;

**end**

**output:** Forest

**Listing 3.** Pseudo code of random forest predictions

**input:** training data (), training fitness (), number of trees , maximum depth of the tree , minimum samples split , minimum samples leaf , Forest

**begin**

Initialize predictions as a zero vector;

**for** **do**

Predict the output using tree;

Train a decision tree on (TD, TF);

Add the predictions to the predictions vector;

**end for**

Average the predictions by dividing by the number of trees;

Return predictions;

**end**

**output:** Predictions

The pseudo code for the decoding procedure is presented as follows:

**Listing 4.** Pseudo code of decoding procedure

**input:** , , , , , , , , , ,

**begin**

(Total cost of collection centers establishment); (Total expected cost of warehouses establishment); (Total distribution cost);(Total scenario cost); (Total distributed product from all warehouses); (Total shortage cost); (Total cost)

**for** **do**

**if** **then**

;

**end if**

**end for**

**for** **do**

**for** **do**

**if** **then**

;

**end if**

**end for**

**for** **do**

**for** **do**

**if** and **then**

;

**end if**

**end for**

**end for**

**for** **do**

**for** **do**

**if** **then**

;

**end if**

**end for**

**end for**

;

**for** **do**

**for** **do**

;

**end for**

;

;

;

**end for**

;

**end for**

;

**end**

**output:** Fitness function value

The pseudo code for the initial population generation is presented as follows:

**Listing 5.** Pseudo code of initial population generation

**input:** , , , , , ,

; ; (length of the chromosome)

**begin**

**for** **do**

**for** **do**

(the values in chromosome)

**end for**

**for** **do**

generate a random number between [0-1];

**if**  **then**

=1;

**end if**

**end for**

**for**  **do**

generate a random number between [0-1];

**for**  **do**

**if**  **then**

=1;

**end if**

**end for**

; (Total amount of products distributed to warehouse )

**for** **do**

**if** =1 **then**

**for**  **do**

**if** =1 **then**

generate a random integer between [0-Min(,];

;

=+;

**end if**

**end for**

**end if**

**end for**

**for** **do**

**if** =1 **then**

**for** **do**

generate a random integer between [0-Min(,];

;

=;

**end for**

;

**end if**

**end for**

**end for**

**end for**

**end**

**output:** Population ()

**Listing 6.** Pseudo code of the proposed crossover operator

**input:**, , , , , , ,

; ; (length of the chromosome)

**begin**

generate a random integer between [0-];

**for** **do**

; ( is the offspring)

;

**end for**

**for** **do**

;

**end for**

**for** **do**

generate a random integer between [0-];

**for** **do**

;

;

**end for**

**for** **do**

;

;

**end for**

**end for**

call the following procedure for and respectively;

**for do**

;

**for** **do**

**if**  **then**

**for**

**if** **then**

;

;

**else**

;

**end if**

**end**

**end if**

**end**

**for** **do**

**if**  **then**

**for**

;

=;

**end for**

;

**else**

;

**end if**

**end for**

**end for**

**Output:** Offspring

**Listing 7.** Pseudo code of the proposed mutation operator

**input:**, , , , ,

; ; (length of the chromosome)

**begin**

**for do**

;

generate a random integer between [0-];

generate random integers , between [0-];

;

;

;

;

**for**

;

**end for**

**for** **do**

**if**  **then**

**for**

=;

**end**

;

**else**

;

**end if**

**end for**

**end for**

**Output:** Middle Population ()

**Listing 8.** Pseudocode of ABCA

**input:** population size , iteration number , the number of employed bees , the number of onlooker bees , total number of iterations allowed for an unimproved solution

**begin**

; (iteration counter)

; (increment number)

Initialize population randomly;

Compute initial fitness function values;

**while** **do**

**for** (**Employed bee stage**)

select a food source randomly;

find the neighborhood solution with *crossover operator*;

compute the fitness function value;

apply *mutation operator* to improve the solution;

**if** the solution is improved **then**

update the employed bee;

**else**

;

**end if**

**end for**

**for** (**Onlooker bee stage**)

choose a solution with *roulette wheel selection* operator;

generate a new solution with *crossover operator*;

evaluate the fitness function value;

**if** the solution is improved **then**

update the onlooker bee;

**else**

;

**end if**

**end for**

**for** (**Scout bee stage**)

**if**  **then**

abandon the solution;

generate a new solution randomly;

**else**

continue with the existing solution;

**end if**

**end for**

**end while**

return best solutions at each iteration;

**end**

**output:** the best solution