**Listing 1.** Pseudocode of the proposed ABCA based algorithms

**Input:** population size , iteration number , number of employed bees , the number of onlooker bees , scout bee limit , K-means clusters,

**Begin**

; (iteration counter)

; (increment number)

Initialize population by applying the *initial population generation* scheme;

Compute the fitness function (FF) values for the initial population;

**While** **do**

**For** // Employed bee phase

Choose a food source randomly;

Generate a neighboring solution by applying the *crossover operator*;

Compute the FF value;

Employ the *mutation operator* to improve the existing solution;

**If** the solution is improved **then**

Update the employed bee;

**Else**

;

**End if**

E**nd for**

**For** // Onlooker bee phase

Select a solution using the *roulette wheel selection* operator;

Generate a new solution by applying the *crossover operator*;

Compute the FF value;

**If** the solution is improved **then**

Update the onlooker bee;

**Else**

;

**End if**

**End for**

**For** // (Scout bee phase)

**If**  **then**

Abandon the solution;

Generate a new solution by applying the *initial population generation* scheme;

**Else**

Preserve the existing solution;

**End if**

**End for**

;

**End while**

Return the best solutions at each iteration;

**End**

**Output:** The best solution and corresponding fitness values and metrics for

**Listing 2.** Pseudo code of chromosome decoding

**Input:** , , K-means clusters,

**Begin**

**For** =1: **do**

; (Supplier utilization cost)

**For** =1: **do**

**If** the supplier is utilized **then**

;

**Else**

;

**End if**

**End for**

; (Storage area utilization cost)

**For** =1: **do**

**If** the storage area is utilized **then**

;

**Else**

;

**End if**

**End for**

; (Distribution cost)

; (Shortage cost)

; (Order receiving and picking cost)

**For** =1: **do**

Compute corresponding , , for the scenario ;

;

;

;

**End for**

Compute corresponding , , and for the robust part;

;

**End for**

Return FF values;

**Output:** FF values for all population

**Listing 3.** Pseudo code of initial population generation

**Input**: , , K-means clusters

**Begin**

Define for the initial population;

**For** =1: **do**

Generate an empty chromosome ;

// Generate first-stage decisions

Determine utilization decisions for suppliers randomly;

**For** =1: **do**

**If** or is applied **then**

Assign a cluster to storage area randomly;

**Else**

Generate a random integer between ;

Determine clusters for storage area randomly;

**End if**

**End for**

// Second-Stage Decisions Subprocedure

**For** =1: **do**

**For** =1: **do**

Find the assigned clusters to storage area ;

**For** =1: **do**

Find the products in cluster ;

**For** =1: **do**

**If** the capacity of storage area is not exceeded **then**

Assign the minimum of demand or supplier’s capacity for product to storage area ;

Update the capacity of storage area ;

**Else**

Assign the minimum of area or supplier’s capacities for product to storage area ;

**End if**

**End for**

**End for**

**End for**

**End for**

Repeat the subprocedure for the robust part of the chromosome;

Return first and second stage decisions for the chromosome ;

**End for**

Return ;

**End**

**Output**: Initial population

**Listing 4.** Pseudocode of the proposed one-point crossover operator

**Input:** , , K-means clusters, Parents (parent-1 and parent-2)

**Begin**

// Perform single-point crossover for first stage decisions

Exchange the supplier utilization decisions between parents;

Generate a random crossover point in the range [0, ];

Extract segment from parent-1’s first-stage decisions up to ;

Extract segment from parent-2’s first-stage decisions after ;

Create new solution-1 by concatenating segments and ;

Create new solution-2 by concatenating segments and ;

Perform feasibility analysis on new solution-1 based on parent-1 and ;

Perform feasibility analysis on new solution-2 based on parent-2 and ;

// Perform single-point crossover for the scenarios

**For** =1: **do**

Generate a random crossover point in the range [0, ];

Extract segment from parent-1’s second-stage decisions up to ;

Extract segment from parent-2’s first-stage decisions after ;

Create new solution-1 by concatenating segments and ;

Create new solution-2 by concatenating segments and ;

Perform feasibility analysis on new solution-1 considering parent-1, demand, capacities, and ;

Perform feasibility analysis on new solution-2 considering parent-2, demand, capacities, and ;

**End for**

Exchange the robust parts between parents;

Preserve the new feasible solutions;

**End**

**Output**: The new solution(s) or neighboring solution(s)

**Listing 5.** Pseudocode of the proposed mutation operator

**Input:** Solution, Fitness function (), , K-means clusters,

**Begin**

**While do**

**For** =1: **do**

**//**Swapping subprocedure

Generate two random integers and within the range [0-];

Swap the values at position and in the solution;

Adjust the second stage of solution for scenario according to demand, capacities, and ;

**End for**

Apply the swapping subprocedure for the robust part of the solution;

Compute new fitness function (NFF) value for the new solution;

**End while**

Return the improved solution;

**End**

**Output**: The improved solution