**Pseudocode M: Mutation Operator**

FUNCTION Mutation :

1. FOR
   1. FOR :

IF discrete Uniform [0,1] mutation rate:

Call function Mutate

END IF

* 1. END FOR

1. END FOR
2. RETURN Mutated Child

FUNCTION Mutate :

1. SET
2. UPDATE residual demand
3. SET
4. REPEAT WHILE :
   1. IF is not NULL:
      1. DETERMINE. lengths of the sections where
      2. DETERMINE and where
      3. SET previous value
      4. SET
   2. ELSE:
      1. CREATE new section
      2. SET
      3. IF
         1. SET
      4. ELSE:
         1. SET
      5. END IF
   3. END IF
   4. UPDATE residual demand
   5. IF Then STOP
   6. ELSE Go To Step 4
5. RETURN Mutated Child

**Pseudocode C: Crossover Operator**

FUNCTION Crossover (*Parent1, Parent2*, *crossover rate* = **1** by default):

1. IF discrete Uniform [0,1] *crossover rate*:

RETURN *Parent1* and STOP

1. SET of *Parent1*; and of Parent 2
2. SET Randomly chosen values from the discrete Uniform [], and where = number of sections of Parent 1.
3. SET the first sections of *Child* by COPYING sections through from *Parent1*
4. UPDATE residual demand of Child
5. SET , where is the index number of sections from Parent2.
6. REPEAT WHILE :
   1. CREATE a new section of Child
   2. IF where = number of sections of Parent2; AND
      1. DETERMINE
      2. SET , where is the number of layers from Parent2 section
      3. FOR :
         1. IF AND THEN SET
         2. ELSE SET
      4. SET
   3. ELSE:
      1. SET
      2. FOR
         1. IF THEN SET
         2. ELSE SET
      3. STOP
   4. UPDATE residual demand of Child
   5. IF Then STOP
   6. ELSE Go To Step 7
7. RETURN Child

**Pseudocode F: Fitness Operator**

FUNCTION Fitness :

1. DETERMINE Calculated cost from Objective Function equation for solution
2. SET
3. UPDATE Residual demand for solution with considering original given demand per size as demand and in demand constraint Equation 8
4. FOR :
   1. IF Then
      1. SET s\_penalty =
      2. SET
5. SET for
6. UPDATE Residual demand for solution with considering as demand and in demand constraint Equation 8
7. FOR :
   1. IF Then
      1. SET e\_penalty = [considering double penalty for excess production]
      2. SET
8. RETURN

**Pseudocode P: Discrete PSO Algorithm**

1. Generate Initial population that has ‘swarmsize’ amount of feasible solutions (particles) by applying H1,H3, & H5 and SET them to variable *‘Swarm’* where each particle is *XP*
2. For p in swarmsize: SET *PbestP = XP*
3. Generate Initial velocity *VP*in range [a,b] for each particle *XP*
4. SET *Gbest* by local search method where Fitness(Gbest) is minimum between all particles.
5. FOR in iteration:
   1. FOR in swarmsize:
      1. Calculate by the Equation (1)
      2. Update velocity by the Equation (2)
      3. Calculate by the Equation (3)
      4. Update by the Equation (4)
      5. Update by calling function
      6. IF Fitness () < Fitness ():
         1. SET =
      7. IF Fitness () < Fitness ():
         1. SET
6. RETURN

FUNCTION :

1. DETERMINE minimum number of sections in
2. INITIALIZE with 0 section.
3. : UPDATE residual demand for solution
4. FOR ):
   1. IF :
      1. : Copy section of (both ) and CREATE a new section for
   2. ELSE IF :
      1. Copy section of (both ) and CREATE a new section for
   3. ELSE:
      1. : Copy section of (both ) and CREATE a new section for
   4. : UPDATE residual demand for solution
   5. IF :
      1. STOP and RETURN
5. SET Total number of sections in
6. FOR :
   1. : Copy section of (both ) and CREATE a new section for
   2. : UPDATE residual demand for solution
   3. IF :
      1. STOP and RETURN
7. IF :
   1. Randomly USE or algorithm to pack all remain demands
8. RETURN

FUNCTION :

1. SET
2. IF Random Uniform [0,1] < *perturbation rate:*
   1. =
   2. SET = Random Integer
   3. IF , then SET
3. FOR :
   1. IF Random Uniform [0,1] < *perturbation rate:*
      1. SET
      2. DETERMINE length of the section
      3. DETERMINE
      4. SET
4. RETURN

**Example of applying PSO Algorithm**

A dummy variable called is employed to enable the change from the continuous state to the discrete state, and vice versa. stands for the m-dimensional vector connected to the solution , which has a value between depending on the particle's state of solution at iteration . Here, m is the total number of sections in particle (solution) .

|  |  |  |
| --- | --- | --- |
|  |  | (1) |

Now, represent the distance between the current solution, , and the best solution found by the particle, . And represent the distance between and the best solution found by the swarm after iteration, .

The velocity term update equation in this discrete PSO is then:

|  |  |  |
| --- | --- | --- |
|  |  | (2) |

Here are two randomly generated number in uniform [0,1] and parameter for velocity update.

The velocity is the used in the next equation:

|  |  |  |
| --- | --- | --- |
|  |  | (3) |

The following equation is then used to update the value of :

|  |  |  |
| --- | --- | --- |
|  |  | (4) |

Lastly the particle will be updated according to this function:

|  |  |  |
| --- | --- | --- |
|  |  | (5) |

Let’s consider an order with the following demands:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Size1 | Size2 | Size3 | Size4 | Size5 |  |  |  |
|  | 7 | 23 | 26 | 17 | 13 |  |  |  |

Now after iteration, lets consider a random particle solution, , it’s personal best solution, and the Global best solution found in swarm is :

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  | |  | |  | |
|  |  | | | | | |  | |  | |  | |
|  | 0 | 0 | 0 | 1 | 1 | 9 | |  | | -0.48712 | |
| 0 | 1 | 1 | 0 | 0 | 22 | |  | | -0.34965 | |
| 1 | 0 | 1 | 0 | 0 | 5 | |  | | -0.09995 | |
| 0 | 0 | 0 | 1 | 0 | 7 | |  | | 0.37745 | |
| 0 | 1 | 0 | 0 | 1 | 4 | |  | | 0.27231 | |
| 0 | 0 | 0 | 1 | 0 | 4 | |  | | 0.22360 | |
| 1 | 0 | 0 | 0 | 0 | 4 | |  | | 0.30020 | |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | | | | |  |  |  |
|  |  | | | | |  |  |  |
|  | 0 | 6 | 0 | 4 | 0 | 4 |  |  |
| 2 | 0 | 3 | 1 | 4 | 4 |  |  |
| 0 | 0 | 4 | 0 | 0 | 4 |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | | | | | |  |  | |  | |
|  |  | | | | | |  |  | |  | |
|  | 1 | 3 | 3 | 2 | 1 | 7 | |  |  | |
| 0 | 1 | 1 | 1 | 1 | 4 | |  |  | |
| 0 | 0 | 1 | 0 | 1 | 4 | |  |  | |

From Equation (1) we can calculate the Y value for our next iteration. has 3 values because we could only compare first 3 sections of with and .

|  |
| --- |
|  |
| 0 |
| 0 |
| 0 |

From Equation (2) we calculate the velocity of the particle at iteration:

|  |
| --- |
|  |
| 0.68173 |
| 0.61888 |
| 0.11383 |
| 0.37745 |
| 0.27231 |
| 0.22360 |
| 0.30020 |

Velocity after 3rd section remains the same since has only 3 values. Now, from Equation (3) we get the .

|  |
| --- |
|  |
| 0.68173 |
| 0.61888 |
| 0.11383 |

Then the Y values are updated with equation (4) for the iteration.

|  |
| --- |
|  |
| 1 |
| 1 |
| 0 |

We then update our particle solution with function which is mainly based on equation (5). First, we create a solution, with 0 section in it. Then we check the value. Since it is 1, we copy a whole section from and add it to our new particle solution. Then we update the residual demands. The demands are positive, so we go to the next value of . Now, , so we copy another section from Gbest and add it to . So far, our new particle looks like this.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | | | | |  |
|  |  | | | | |  |
|  | 1 | 3 | 3 | 2 | 1 | 7 |
|  | 0 | 1 | 1 | 1 | 1 | 4 |

We again update the residual demands.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| Size1 | Size2 | Size3 | Size4 | Size5 |
| 0 | -2 | 1 | -1 | 2 |

Since there are still positive numbers, we will go to the next value of . This time . We copy 3rd section of perticle and temporary add it to .

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 0 | 1 | 0 | 0 | 5 |

is then subject to a small perturbation. First, we consider perturbating the value. We generate a random number from range uniform [0,1] and compare it with our given perturbation rate (). Let's say that our random generated number is less than . We will therefore change the value. We then determine two integers by the following way:

=

In our case Thus, we get our Therefore, we set = random integer between [4,5] = 4

Next, we consider perturbating the value. Suppose that value of Sizel 1 is randomly chosen to be moved. First, we set . Then, we determine . In our case residual demand for Size l, which gives us . Therefore, we update . After perturbation we then add this section permanently to and update the residual demands.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | | | | |  |
|  |  | | | | |  |
|  | 1 | 3 | 3 | 2 | 1 | 7 |
|  | 0 | 1 | 1 | 1 | 1 | 4 |
|  | 0 | 0 | 1 | 0 | 0 | 4 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| Size1 | Size2 | Size3 | Size4 | Size5 |
| 0 | -2 | -3 | -1 | 2 |

Since there is no value left and there is still residual demand exist, we keep adding perturbated sections from until there is no residual demands or no section left in . If there are still residual demands after adding all the sections from we would randomly use H1, or H3 algorithm to pack all the demands. However, in our case after adding 2 more sections from , solution completed.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | | | | |  |
|  |  | | | | |  |
|  | 1 | 3 | 3 | 2 | 1 | 7 |
|  | 0 | 1 | 1 | 1 | 1 | 4 |
|  | 0 | 0 | 1 | 0 | 0 | 4 |
|  | 0 | 0 | 0 | 1 | 0 | 6 |
|  | 0 | 0 | 0 | 0 | 1 | 4 |