$s \leftarrow initialsolution$;
$bestSolution \leftarrow s$;
$t \leftarrow maxTemperature$;
repeat
  $iter \leftarrow 0$;
  repeat
    $s' \leftarrow$ *select a random solution* $s' \in N(s)$;
    $\Delta \leftarrow f(s') - f(s)$;
    if $\Delta < 0$ **then**
      $s \leftarrow s'$;
      if $f(s') < f(bestSolution)$ **then**
        $bestSolution \leftarrow s'$;
      end
    else if $rand(0,1) < \exp(\delta/T)$ **then**
      $s \leftarrow s'$;
    end
    $iter \leftarrow iter + 1$;
  until $iter \geq maxIterations$;
  $t \leftarrow \alpha \times t$
until $t \leq minTemperature$;
return bestSolution;

Algorithm 1: Pseudo-code of Simulated Annealing

$\mu \leftarrow initialCenter$;
$r_0 \leftarrow initialRadius$;
$best \leftarrow bestSolution$;
$f_{best} \leftarrow \infty$;
$iter \leftarrow 0$;
repeat
  $C_{iter}(s) \leftarrow$
   generate random solutions with the radius r_{iter};
  $s' \leftarrow$ *choose the best solution* (s) *in* $C_{iter}(s)$;
  if $f(s') < f_{best}$ **then**
    $best \leftarrow s'$;
    $f_{best} \leftarrow f(s')$;
  $r_{iter+1} \leftarrow$ *reduce the radius for next iteration*;
  $iter \leftarrow iter + 1$;
until $iter \geq maxIterations$;
return best;

Algorithm 2: Pseudo-code of Vortex Search

$P \leftarrow Initialize\ the\ population$;
$f_P \leftarrow Evaluate\ the\ population$;
repeat
  foreach *individual* $\in P$ **do**
    $x \leftarrow choose\ one\ individual\ with\ respect\ to\ f(P)$;
    $y \leftarrow choose\ one\ individual\ with\ respect\ to\ f(P)$;
    $child \leftarrow apply\ crossover\ to\ generate\ a\ new\ child$;
    $child_{mut} \leftarrow apply\ mutation\ on\ child\ to\ ensure\ diversity$
     $P \leftarrow add\ child_{mut}\ into\ the\ population\ [P \cup child_{mut}]$;
  end
  $r \leftarrow rank\ individuals\ with\ respect\ to\ F(P)$;
  $P \leftarrow$
until termination criterion is satisfied;
return best-of-run individual

Algorithm 3: Pseudo-code of Genetic Algorithm

$CR \leftarrow crossover\ rate$;
$P \leftarrow Initialize\ the\ population\ (i.e., x_j \in P \mid j = \{1, 2, ..., N\})$;
$f(P) \leftarrow evaluate\ the\ fitness\ of\ population$;
$best \leftarrow bestSolution$;
repeat
  foreach individual $j \in P$ **do**
    $n_1, n_2, n_3 \leftarrow choose\ three\ numbers\ such\ that\ 1 \leq n_1, n_2, n_3 \leq N\ and\ n_1 \neq n_2 \neq n_3\ \neq j$;
    $i_{rand} \leftarrow generate\ a\ random\ integer\ i_{rand} \in (1, N)$;
    foreach parameter i **do**

$$x'_{i,j} = \begin{cases} x_{i,n_3} + F \times (x_{i,n_1} - x_{i,n_2}) & rand(0,1) < CR \vee i = i_{rand} \\ x_{i,j} & otherwise \end{cases}$$

    end
    $f(x'_j) \leftarrow evaluate\ fitness$;
    if $f(x'_j) < f(x_j)$ **then**
      $x_j \leftarrow x'_j$;
      $f(x_j) \leftarrow f(x'_j)$;
      if $f(x'_j) < f(best)$ **then**
        $best \leftarrow x'_j$;
      end
    end
  end
until termination criterion is satisfied;
return best

Algorithm 4: Pseudo-code of Differential Evolution

$P \leftarrow Initialize\ the\ particle\ population$;
$pbest \leftarrow P$;
$gbest \leftarrow best\ particle\ initially\ found$;
repeat
  foreach *particle with position* x_p **do**
    $f_(x_p) \leftarrow evaluate\ the\ fitness\ of\ x_p$;
    if $f(x_p) < f(pbest_p)$ **then**
      $pbest \leftarrow x_p$;
    end
  end
  $gbest \leftarrow find\ the\ current\ best\ particle$;
  foreach *particle with position* x_p **do**
    $v_p \leftarrow compute\ velocity\ with\ respect\ to\ x_p,\ pbest_p,\ gbest$;
    $x_p \leftarrow update\ position\ with\ respect\ to\ x_p\ and\ v_p$;
  end
until termination criterion is satisfied;
return gbest

Algorithm 5: Pseudo-code of Particle Swarm Optimization

$P \leftarrow Initialize\ the\ population (i.e., x_i \in P \mid i = \{1, 2, ..., N\})$;
$f(P) \leftarrow Evaluate\ the\ population$;
repeat
  foreach *employed bee* $\in P$ **do**
    $v_i \leftarrow produce\ new\ solution$;
    $f(v_i) \leftarrow evaluate\ the\ fitness\ of\ v_i$;
    if $f(v_i) < f(x_i)$ **then**
      $x_i \leftarrow v_i$;
      $f(x_i) \leftarrow f(v_i)$;
    end
  end
  $p_i \leftarrow calculate\ the\ probability\ for\ solution\ x_i$;
  foreach *onlooker bee* $\in P$ **do**
    $x_i \leftarrow select\ a\ solution\ depending\ on\ p_i$;
    $v_i \leftarrow produce\ new\ solution$;
    $f(v_i) \leftarrow evaluate\ the\ fitness\ of\ v_i$;
    if $f(v_i) < f(x_i)$ **then**
      $x_i \leftarrow v_i$;
      $f(x_i) \leftarrow f(v_i)$;
    end
  end
  if *abadoned solution* (x_j) *is found* **then**
    $x_j \leftarrow randomly\ generate\ a\ new\ solution$;
  end
  $best \leftarrow memorize\ the\ best\ solution\ so\ far$;
until termination criterion is satisfied;
return best

Algorithm 6: Pseudo-code of Artificial Bee Colony