The pseudocodes and flowcharts of GA, ALS, DHS+BT, BT+DHS, KA, HGWO, HHO, CSA, RDA, and SEO for solving the scheduling problem of FASs with blocking and deadlock constraints are presented in the following.

1. Genetic algorithm for scheduling FASs

Similar to the encoding and decoding methods designed in HPSO, a chromosome in genetic algorithm (GA) can be encoded as a permutation of randomly generated part numbers. Then, the chromosome can be converted into a transition sequence, and the repairing algorithm (RA) from HPSO is used in to ensure that a feasible transition sequence can be obtained. The pseudocode of GA is presented in Algorithm 1, and the flowchart is shown in Fig. 1.

Algorithm 1: GA

Input: An FAS data set;

Output: A feasible schedule α and its fitness value;

---

initialize: $T_{CPU} = 0$, $T_m = u \cdot C_R/5$, $N = 100$, $p_s = 6$, and $best = \infty$; /*T_{CPU} is the actual running time, T_m is the maximum running time, N is the number of chromosomes in the population, p_s is the number of the best chromosomes that are directly copied to the next generation, and *best* records the best result and is initialized to ∞. */

Randomly generate the initial population;

while($T_{CPU} < T_m$){

  for(each chromosome in the population){

    Decode chromosome S_i to obtain the transition sequence $\tau(S_i)$;

    RA(S_i, $\tau(S_i)$)

    Calculate the fitness value of S_i, i.e., $F(S_i) = C_{max}(\tau(S_i))$;

    if($F(S_i) < best$){$\alpha := S_i$ and $best := F(S_i)$;}

  }**end for**

    Copy the best p_s chromosomes to the next generation;

    while($i \le (N - p_s)/2$){/* initialize $i = 1$. */

      Select two parent chromosomes using binary tournament method;

      Perform the crossover operator to generate two child chromosomes;

      i++;

    }**end while**

    Perform the mutation operator on each child chromosome; /*The best p_s chromosomes and the newly generated child chromosomes constitute the next generation. */

    T_{CPU} := *cputime*(); /**cputime*() is the running time of the algorithm. */

}**end while**

Output α and *best*.

End

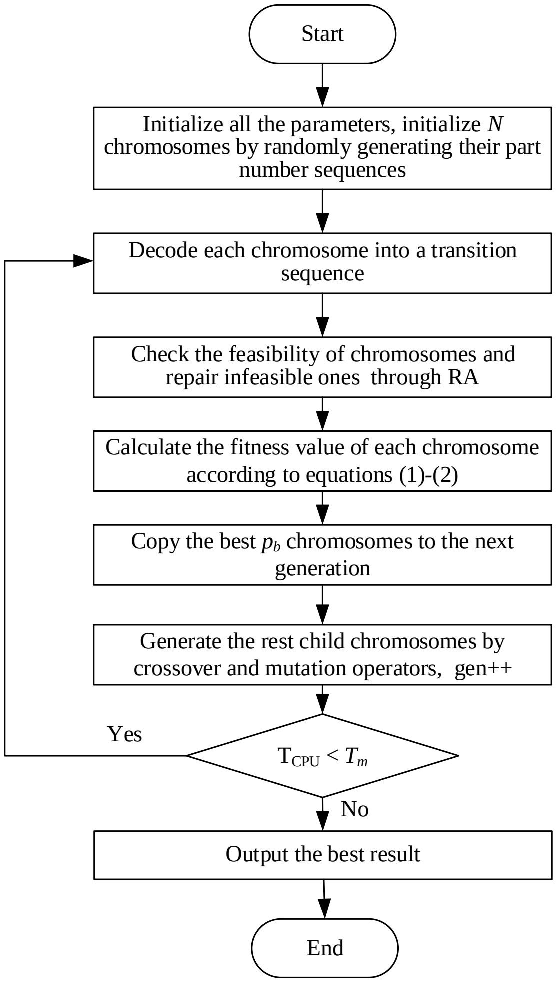

Fig. 1. Flowchart of GA for scheduling FASs.

2. Anytime layered search algorithm for scheduling FASs

In Algorithm 2, the pseudocode of anytime layered search (ALS) method is presented, where hUB is a hypothetical upper bound and initialized to $f(M_0, \varepsilon)$ ($f(M_0, \varepsilon)$ is the estimated remaining processing time calculated based on the heuristic function presented in [1]), sUB is a suboptimal upper bound and initialized to ∞, CPU_{limit} is the a predefined computation time limit, and *width* is used to limit the number of nodes to be expanded at each layer. CPU_{limit} is set to $u \times C_R/5$ seconds. The value of *width* is set to 10.

Algorithm 2: ALS

Input: An FAS model

Output: α and $C_{max}(\alpha)$

Initialize: OPEN0 = $\{(M_0, \varepsilon)\}$, OPEN = $\emptyset$, NODE = $\emptyset$, NODE1 = $\emptyset$, NODE2 = $\emptyset$, hUB = $f(M_0, \varepsilon)$, sUB = ∞, CPU_{limit}, $d_{frontier}$ = 0, *width*;

while(hUB < sUB and cputime() < CPU_{limit}) do{

  while(OPEN0 $\neq \emptyset$){

  select a node (M, π) from OPEN0;

  compute the set of enabled transitions for (M, π), denoted as $ET(M, \pi)$;/*MBA[2] is used to calculate $ET(M, \pi)$.*/

  OPEN0 := OPEN0\$\{(M, \pi)\}$;

while($ET(M, \pi) \neq \varnothing$) do{

    select a transition $t \in ET(M, \pi)$, $M[t > M_1$, $\pi_1 = \pi t$, and calculate $g(M_1, \pi_1)$, $f(M_1, \pi_1)$;

    $ET(M, \pi) := ET(M, \pi)\backslash\{t\}$;

    if(there is a node (M_1, π_2) in NODE satisfying $g(M_1, \pi_2) > g(M_1, \pi_1)$){NODE := (NODE$\backslash\{(M_1, \pi_2)\})\cup\{(M_1, \pi_1)\}$;}

    else{NODE := NODE$\cup\{(M_1, \pi_1)\}$;}/* (M_1, π_1) is a new node */

}/***end while**($ET(M, \pi) \neq \varnothing$) */

while(NODE $\neq \varnothing$) do{

    select a node (M', π') from NODE;

    NODE := NODE$\backslash(M', \pi')$;

    if($f(M', \pi')$ > hUB){OPEN := OPEN$\cup(M', \pi')$, $d_{frontier}$ = depth(M', π');}

    else{OPEN0 := OPEN0$\cup(M', \pi')$;}

}/***end while**(NODE $\neq \varnothing$) */

}/***end while**(OPEN0 $\neq \varnothing$) */

/* sBFHS starts */

$i := d_{frontier} - 1$;

UB := hUB;

while(OPEN[i] $\neq \varnothing$) do{/* OPEN[i] denote the set of nodes in OPEN that are in layer i */

    for all nodes in OPEN[i], select the best *width* nodes satisfying $f(M_3, \pi_3) \leq$ UB and put in NODE1;/* according to the heuristic evaluation function values of nodes */

    while(NODE1 $\neq \varnothing$) do{

    select a node (M_3, π_3) from NODE1;

    compute the set of enabled transitions for (M_3, π_3), denoted as $ET(M_3, \pi_3)$;/*MBA[2] is used to calculate $ET(M_3, \pi_3)$.*/

    NODE1 := NODE1$\backslash\{(M_3, \pi_3)\}$, OPEN := OPEN $\backslash\{(M_3, \pi_3)\}$;

    while($ET(M_3, \pi_3) \neq \varnothing$) do{

        select a transition $t \in ET(M_3, \pi_3)$, $M_3[t_3 > M_4$, $\pi_4 = \pi_3 t$, and calculate $g(M_4, \pi_4)$, $f(M_4, \pi_4)$;

        if($M_4 = M_f$){ if($C_{max}(\pi_4) < C_{max}(\alpha)$){$\alpha = \pi_4$, $C_{max}(\alpha) = C_{max}(\pi_4)$, sUB = $C_{max}(\alpha)$}

        else{

            if(there is a node (M_4, π_5) in NODE2 satisfying $g(M_4, \pi_5) > g(M_4, \pi_4)$){NODE2 := NODE2$\backslash\{(M_4, \pi_5)\} \cup\{(M_4, \pi_4)\}$;}

            else{NODE2 := NODE2$\cup\{(M_4, \pi_4)\}$;}}

        }/***end while**($ET(M_3, \pi_3) \neq \varnothing$)*/

    }/***end while**(NODE1 $\neq \varnothing$)*/

    while(NODE2 $\neq \varnothing$) do{

    select a node (M_6, π_6) from NODE2;

    if(there is a node (M_6, π_7) in OPEN satisfying $g(M_6, \pi_7) > g(M_6, \pi_6)$){OPEN := (OPEN$\backslash\{(M_6, \pi_7)\})\cup\{(M_6, \pi_6)\}$;}

    else{ OPEN := OPEN$\cup\{(M_6, \pi_6)\}$;}

    NODE2 :=NODE2$\backslash\{(M_6, \pi_6)\}$;

    }/***end while**(NODE2 $\neq \varnothing$)*/

    $i = i + 1$;

    if(f_{min}(OPEN[i]) > hUB){ UB $\leftarrow$ sUB -1; $d_{frontier} \leftarrow i$-1;}

}/***end while**(OPEN[i] $\neq \varnothing$) */

hUB ← minimum f-cost of OPEN[k], $\forall k \in [0, d_{frontier}]$;
backtrack to the node with hUB in the deepest layer and update $d_{frontier}$, put the node into OPEN0;
delete nodes in OPEN satisfying $f(M, \pi) >$ sUB;
cputime();/* running time of the search procedure*/
if(OPEN = $\emptyset$){break;}
}/* **end while**(hUB < sUB and cputime() < CPU_{limit}) */
End

[1] Baruwa, T., Piera, M. A., Guasch, A., 2015. Deadlock-free scheduling method for flexible manufacturing systems based on timed colored Petri nets and anytime heuristic search. IEEE Trans. Syst., Man, and Cybern.: Syst. 45(5), 831-846.

[2] Luo, J. C., Liu, Z. Q., Zhou, M. C., 2019. A Petri Net-based Deadlock Avoidance Policy for Flexible Manufacturing Systems with Assembly Operations and Multiple Resource Acquisition, IEEE Trans. Ind. Informat. 15(6), 3379-3387.

3. DHS+BT and BT+DHS algorithm for scheduling FASs

The pseudocodes of different combinations of deadlock-free heuristic search (DHS) and backtracking (BT) strategy are given in Algorithm 3 and Algorithm 4. In Algorithm 3, DHS is used until a depth-bound d is reached, then BT is applied. While in Algorithm 4, BT is used until a depth-bound d is reached, then DHS is applied. OPEN and CLOSED are two lists in DHS, and OPEN0 is the list in BT. The parameter d is set to the half of depth(M_f, α).

Algorithm 3: DHS+BT

Input: An FAS model
Output: α and $C_{max}(\alpha)$

Initialize: OPEN = $\{(M_0, \varepsilon)\}$, CLOSED = $\emptyset$, OPEN0 = $\emptyset$, depth(M_0, ε) = 0, depth-bound = d, Flag = 1, Flag0 = 1;/* depth(M, π) records the depth of node, which equals to the length of the transition sequence */
while(OPEN $\neq \emptyset$) do{
  while(Flag = 1) do{
    select a node (M, π) from OPEN with minimum $f(M, \pi)$;
    if(depth(M, π) > d){ Flag = 0, OPEN0 : = $\{(M, \pi)\}$; If(Flag0 = 0){Flag0 = 1;}break}
    compute the set of enabled transitions for (M, π), denoted as $ET(M, \pi)$;/*MBA[2] is used to calculate $ET(M, \pi)$.*/
    if($ET(M, \pi) = \emptyset$){OPEN := OPEN\$\{(M, \pi)\}$; }/* ($M$, π) is a deadlock state. */
    else{
      OPEN := OPEN\$\{(M, \pi)\}$;
      CLOSED := CLOSED$\cup\{(M, \pi)\}$;
      while($ET(M, \pi) \neq \emptyset$) do{
        select a transition $t \in ET(M, \pi)$, $M[t > M_1$, $\pi_1 = \pi t$, and calculate $f(M_1, \pi_1)$;
        $ET(M, \pi) := ET(M, \pi)\backslash\{t\}$;
        if(there is a node (M_1, π_2) in OPEN satisfying $f(M_1, \pi_2) > f(M_1, \pi_1)$){
          OPEN := (OPEN\$\{(M_1, \pi_2)\})\cup\{(M_1, \pi_1)\}$;}
        else if(there is a node (M_1, π_2) in CLOSED satisfying $f(M_1, \pi_2) > f(M_1, \pi_1)$){

OPEN := OPEN∪{(M_1, π_1)}, CLOSED := CLOSED\{(M_1, π_2)};}
else if(there is no node with marking M_1 in OPEN∪CLOSED){
OPEN := OPEN∪{(M_1, π_1)};}}
}/***end while**($ET(M, \pi) \neq \emptyset$) */
} /***end while**(Flag = 1)*/
while(Flag0 = 1) do{/* BT starts.*/
if(OPEN0 = ∅){Flag1 = 0; if(Flag = 0){Flag = 1;}, break;}
select the topmost node (M', π') from OPEN0;
compute the set of enabled transitions for (M', π'), denoted as $ET(M', \pi')$;/* MBA[2] is used to calculate $ET(M', \pi')$.*/
let COUNT(M', π') denote the number of fired transitions at (M', π'); /* if (M', π') is a new generated node, initialize COUNT(M', π') = 0 */
if(COUNT(M', π') = $|ET(M', \pi')|$){OPEN0 := OPEN0\{(M', π')};}/* $|ET(M', \pi')|$ is the number of enabled transitions at (M', π').*/
else{
for($t' \in ET(M', \pi')$){
if(flag(M', t') = 0){$M'[t' > M''$, $\pi'' = \pi' t'$, flag(M', t') = 1; calculate $f(M'', \pi'')$; COUNT(M', π') = COUNT(M', π') + 1; break;} /* flag(M', t') = 1 means transition t' at M' has been fired. */
}/* **End for**($t' \in ET(M', \pi')$)*/
if($M'' = M_f$){$\alpha = \pi''$, $C_{max}(\alpha) = C_{max}(\pi'')$, output α and $C_{max}(\alpha)$; Exit;}
else{
if($ET(M'', \pi'') \neq \emptyset$){OPEN0 := OPEN0 ∪{($M''$, π'')}; }/* (M'', π'') is put on the top of OPEN0.*/}}
}/***end while**(Flag0 = 1)*/
}/* **end while**(OPEN ≠ ∅)*/
End

Algorithm 4: BT+DHS

Input: An FAS model

Output: α and $C_{max}(\alpha)$

Initialize: OPEN = ∅, CLOSED = ∅, OPEN0 = {(M_0, ε)}, depth(M_0, ε) = 0, depth-bound = d, Flag = 1, Flag0 = 1;
while(OPEN0 ≠ ∅) do{
while(Flag0 = 1) do{
select the topmost node (M, π) from OPEN0;
if(depth(M, π) > d){ Flag0 = 0, OPEN : = {(M, π)}; If(Flag = 0){Flag = 1;}break}
compute the set of enabled transitions for (M, π), denoted as $ET(M, \pi)$;/* MBA[2] is used to calculate $ET(M, \pi)$.*/
let COUNT(M, π) denote the number of fired transitions at (M, π); /* if (M, π) is a new generated node, initialize COUNT(M, π) = 0 */
if(COUNT(M, π) = $|ET(M, \pi)|$){OPEN0 := OPEN0\(M, π);}/* $|ET(M, \pi)|$ is the number of enabled transitions at (M, π).*/
else{

10: **for**($t \in ET(M, \pi)$){

11: if(flag(M, t) = 0){$M[t > M_1$, $\pi_1 = \pi t$, flag(M, t) = 1; calculate $f(M_1, \pi_1)$; COUNT(M, π) = COUNT(M, π) + 1; break;} /* flag(M, t) = 1 means transition t at marking M has been fired */

12: }/***end for**($t \in ET(M, \pi)$)*/

13: if($ET(M_1, \pi_1) \neq \varnothing$){OPEN0 := OPEN0 $\cup$ {(M_1, π_1)}; }/* (M_1, π_1) is put on the top of OPEN0*/

14: }

15: }/***end while**(Flag0 = 1)*/

16: **while**(Flag = 1) do{/* A* algorithm starts */

17: if(OPEN = $\varnothing$){ Flag = 0; if(Flag0 = 0){Flag0 = 1;}, break;}

18: select a node (M', π') from OPEN with minimum $f(M', \pi')$;

19: compute the set of enabled transitions for (M', π'), denoted as $ET(M', \pi')$;/* MBA[2] is used to calculate $ET(M', \pi')$.*/

20: if($ET(M', \pi') = \varnothing$){OPEN := OPEN\{($M'$, π')}; }/* (M', π') is a deadlock state. */

21: else{

22: OPEN := OPEN\{(M', π')};

23: CLOSED := CLOSED$\cup${(M', π')};

24: **while**($ET(M', \pi') \neq \varnothing$) do{

25: select a transition $t' \in ET(M', \pi')$, $M'[t' > M''$, $\pi'' = \pi' t'$, and calculate $f(M'', \pi'')$;

26: $ET(M', \pi') := ET(M', \pi')\backslash\{t'\}$;

27: **if**($M'' = M_f$){ $\alpha = \pi''$, $C_{max}(\alpha) = C_{max}(\pi'')$, output α and $C_{max}(\alpha)$; Exit;}

28: else{

29: if(there is a node (M'', π_2) in OPEN satisfying $f(M'', \pi_2) > f(M'', \pi'')$){

30: OPEN := (OPEN\{(M'', π_2)})$\cup${(M'', π'')};}

31: else if(there is a node (M'', π_2) in CLOSED satisfying $f(M'', \pi_2) > f(M_1, \pi'')$){

32: OPEN := OPEN$\cup${(M'', π'')}, CLOSED := CLOSED\{(M'', π_2)};}

33: else if(there is no node with marking M'' in OPEN$\cup$CLOSED){

34: OPEN := OPEN$\cup${(M'', π'')};}

35: } /***end if**($M'' = M_f$)-else */

36: }/***end while**($ET(M', \pi') \neq \varnothing$) */

37: } /***end while**(Flag = 1)*/

38: }/***end while**(OPEN0 $\neq \varnothing$)*/

End

4. Keshtel algorithm for scheduling FASs

The pseudocode of keshtel algorithm (KA) is shown in Algorithm 5, and the corresponding flowchart is presented in Fig. 2.

Algorithm 5: KA

Input: An FAS data set;

Output: A feasible schedule α and its fitness value;

1: **initialize:** $T_{CPU} = 0$, $T_m = u \times C_R/5$, $N = 50$, $LK = 5$, $WK = 5$, $S_{max} = 5$, and $best = \infty$; /*T_{CPU} is the actual running time, T_m is the maximum value of running time, N is the number of keshtels in the swarm, LK is the number of lucky keshtels, WK is the number of the worst keshtels, and $best$ records the best result and is initialized to ∞.*/

2: Randomly generate the initial swarm of keshtels;
3: **while**($T_{CPU} < T_m$){
4: **for**(each keshtel in the swarm){
5: Decode individual S_i to obtain the transition sequence $\tau(S_i)$;
6: RA(S_i, $\tau(S_i)$)
7: Calculate the fitness value of S_i, i.e., $F(S_i) = C_{max}(\tau(S_i))$;
8: if($F(S_i) < best$){$\alpha := S_i$ and $best := F(S_i)$;}
9: }**end for**
10: Select the best LK individuals as the lucky keshtels, and let them remain in the swarm;
11: **while**($i \leq LK$){/* initialize i = 1. */
12: Find the nearest neighbor keshtel of the i-th lucky keshtel, denoted as k;
13: **while**($s < S_{max}$){ /* initialize s = 0. */
14: Let the keshtel k swirl around i to obtain a new position vector, denoted the new individual by S_k';
15: Decode S_k' to obtain the transition sequence $\tau(S_k')$;
16: if($F(S_k') < F(S_i)$){ $S_k := S_i$ and $S_i := S_k'$;}
17: else if($F(S_k') < F(S_k)$ and $F(S_k') > F(S_i)$){ $S_k := S_k'$;}
18: s++;
19: }**end while**
20: i++;
21: }**end while**
22: For the worst WK keshtels, randomly generated new part number sequences and position vectors to replace them;
23: **for**(each keshtel S_j in the rest keshtels of the swarm){
24: Randomly select two keshtels, move the keshtel S_j toward an empty place between them; /* Update the keshtel's position vector and part number sequence. */}
25: T_{CPU} := *cputime*(); /**cputime*() is the running time of the algorithm. */
26: }**end while**
27: Output α and *best*.
End

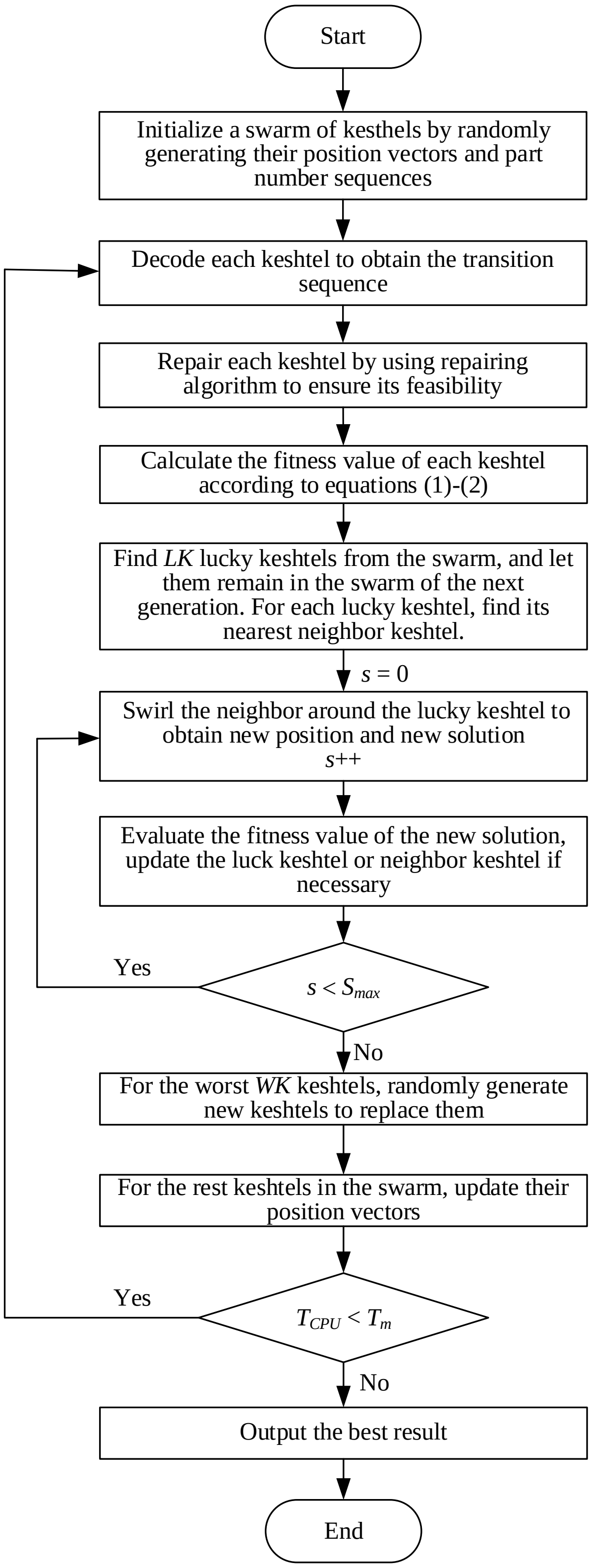

Fig. 2. Flowchart of KA for scheduling FASs.

5. Hyperbolic grey wolf optimizer for scheduling FASs

The pseudocode of Hyperbolic grey wolf optimizer (HGWO) for scheduling FASs is shown in Algorithm 6, and the corresponding flowchart is presented in Fig. 3.

Algorithm 6: HGWO

Input: An FAS data set;

Output: A feasible schedule π and its fitness value;

initialize: $T_{CPU} = 0$, $T_m = u \times C_R/5$, $N = 50$, $a_0 = 2.0$, and $best = \infty$; /*T_{CPU} is the actual running time, T_m is the maximum value of running time, N is the number of wolfs in the group, and *best* records the best result and is initialized to ∞.*/

Randomly generate the initial group of wolfs;

while($T_{CPU} < T_m$){

    for(each wolf in the group){

        Decode individual S_i to obtain the transition sequence $\tau(S_i)$;

        RA(S_i, $\tau(S_i)$)

        Calculate the fitness value of S_i, i.e., $F(S_i) = C_{max}(\tau(S_i))$;

        if($F(S_i) < best$){$\pi := S_i$ and $best := F(S_i)$;}

    }**end for**

    Select the best wolf, the second best wolf, and the third best wolf from the group, and denote them as S_α, S_β, and S_δ, respectively;

    Retain S_α, S_β, and S_δ in the next generation;

    for(each wolf in the group except S_α, S_β, and S_δ){/* initialize $i = 1$. */

        $a := a_0*(maxgen - \text{gen})/max\text{-}gen$;

        $r_1 = \text{rand}(0, 1)$;

        if($r_1 < 0.5$){

            Update its position vector according traditional GWO;}

        else{

            Use hyperbolic acceleration strategy to update its position vector;}

    }**end for**

    gen++;

    T_{CPU} := *cputime*(); /**cputime*() is the running time of the algorithm. */

}**end while**

Output π and *best*.

End

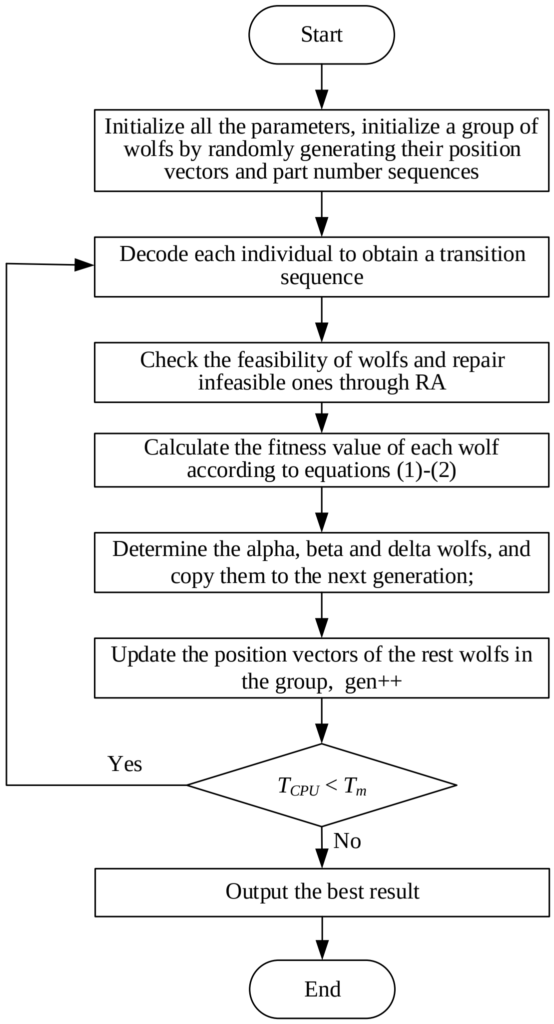

Fig. 3. Flowchart of HGWO for scheduling FASs.

6. Harries hawks optimizer for scheduling FASs

The pseudocode of Harries hawks optimizer (HHO) for scheduling FASs is shown in Algorithm 7, and the corresponding flowchart is presented in Fig. 4.

Algorithm 7: HHO

Input: An FAS data set;

Output: A feasible schedule α and its fitness value;

initialize: $T_{CPU} = 0$, $T_m = u \times C_R/5$, $N = 50$, E_0, $\beta = 1.5$, and *best* = ∞; /*T_{CPU} is the actual running time, T_m is the maximum value of running time, N is the number of hawks in the group, E_0 is the initial energy, β is the parameter in levy flight function, and *best* records the best result and is initialized to ∞.*/

Randomly generate the initial group of Harris hawks;

while($T_{CPU} < T_m$){

  for(each hawk in the group){

    Decode solution S_i to obtain the transition sequence $\tau(S_i)$;

    RA(S_i, $\tau(S_i)$)

    Calculate the fitness value of S_i, i.e., $F(S_i) = C_{max}(\tau(S_i))$;

if($F(S_i) < best$){$\pi := S_i$ and $best := F(S_i)$, let $S_{rabbit} := S_i$;}/* S_{rabbit} denote the rabbit.*/
}**end for**
for(each hawk in the group){
r_1 = rand(0, 1), $E_0 = 2r_1 - 1$, $E = 2E_0(max\text{-}gen - \text{gen})/max\text{-}gen$;
r_2 = rand(0, 1), $J = 2(1 - r_2)$;
r_3 = rand(0, 1)
According to the value of E and r_3, perform exploration process or exploitation process based on the equations in [3] to update the position vector and obtain new individual;
}**end for**
gen++;
T_{CPU} := *cputime*(); /**cputime*() is the running time of the algorithm. */
}**end while**
Output α and *best*.
End

[3] Heidari A. A., Mirjalili S., Faris H., Aljarah I., Mafarja M., Chen H., 2019. Harris hawks optimization: Algorithm and applications. Future Gener. Comput. Syst. 97, 849-872.

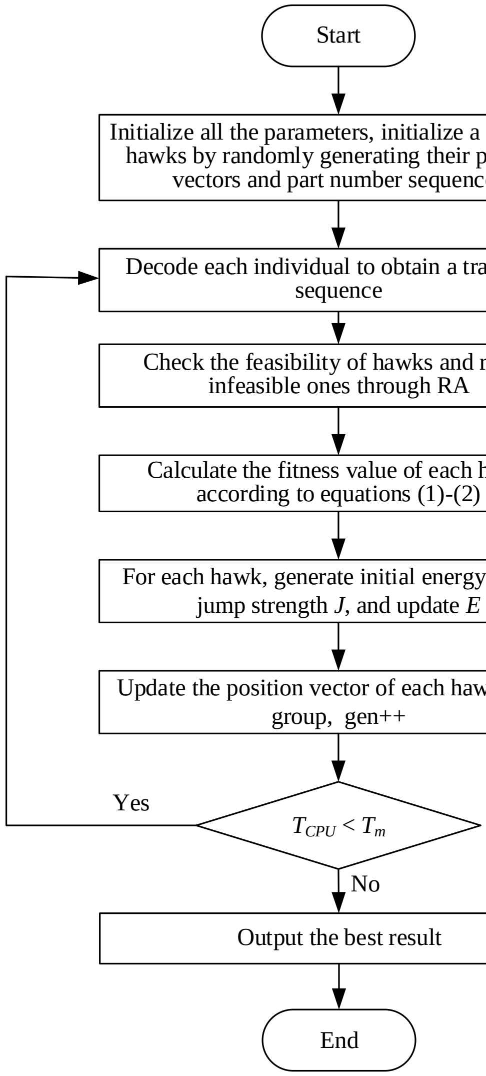

Fig. 4. Flowchart of HHO for scheduling FASs.

7. Cooperation search algorithm for scheduling FASs

The pseudocode of Cooperation search algorithm (CSA) for scheduling FASs is shown in Algorithm 8, and the corresponding flowchart is presented in Fig. 5.

Algorithm 8: CSA

Input: An FAS data set;

Output: A feasible schedule π and its fitness value;

initialize: $T_{CPU} = 0$, $T_m = u \times C_R/5$, $N = 100$, $M = 5$, $\alpha = \beta = 2.0$, GLOBAL, $pbest_i$, gen = 0, and $best = \infty$; /*T_{CPU} is the actual running time, T_m is the maximum value of running time, N is the number of staffs in the group, M is the number of global best-known solutions, GLOBAL is a list for storing the M global best-known individuals, α and β are the learning coefficients, $pbest_i$ is the personal best solution of staff i, and $best$ records the best result and is initialized to ∞.*/
Randomly generate the initial group of staffs;
while($T_{CPU} < T_m$){
  if(gen = 0){
    for(each staff in the group){
      Decode solution S_i to obtain the transition sequence $\tau(S_i)$;
      RA(S_i, $\tau(S_i)$)
      Calculate the fitness value of S_i, i.e., $F(S_i) = C_{max}(\tau(S_i))$;
    }**end for**
}
  for(each staff in the group){
    if($F(S_i) < F(pbest_i)$){$pbest_i := S_i$;}
    if($F(S_i) < best$){$\pi := S_i$ and $best := F(S_i)$;}
    if($F(S_i)$ is better than the worst fitness value in GLOBAL){ Update GLOBAL;}
  }**end for**
  for(each staff in the group){
    Generate a new individual NI_{1i}; /* Team communication operator. */
    Decode individual NI_{1i} to obtain the transition sequence $\tau(NI_{1i})$;
    RA(NI_{1i}, $\tau(NI_{1i})$);
    Calculate the fitness value of NI_{1i}, i.e., $F(NI_{1i}) = C_{max}(\tau(NI_{1i}))$;
    Generate a new individual NI_{2i}; /* Reflective learning operator. */
    Decode solution NI_{2i} to obtain the transition sequence $\tau(NI_{2i})$;
    RA(NI_{2i}, $\tau(NI_{2i})$);
    Calculate the fitness value of NI_{2i}, i.e., $F(NI_{2i}) = C_{max}(\tau(NI_{2i}))$;
    if($F(NI_{1i}) > F(NI_{2i})$){$S_i := NI_{2i}$;}/* Internal competition operator. */
    else{ $S_i := NI_{1i}$;}
  }**end for**
  gen++;
  T_{CPU} := *cputime*(); /**cputime*() is the running time of the algorithm. */
}**end while**
Output π and *best*.
End

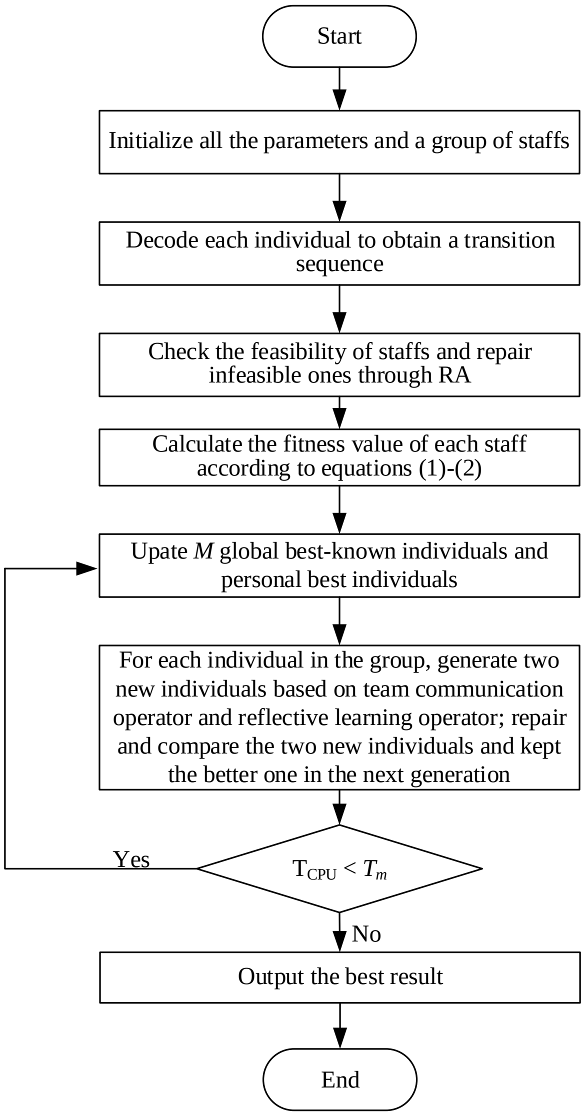

Fig. 5. Flowchart of CSA for scheduling FASs.

8. Red deer algorithm for scheduling FASs

The pseudocode of red deer algorithm (RDA) for scheduling FASs is shown in Algorithm 9, and the corresponding flowchart is presented in Fig. 6.

Algorithm 9: RDA

Input: An FAS data set;

Output: A feasible schedule π and its fitness value;

1: **initialize:** $T_{CPU} = 0$, $T_m = u \times C_R/5$, $N = 100$, $N_{male} = 10$, $N_{hind} = 90$, $N_{com} = 5$, $N_{stag} = 5$, and $best = \infty$; /*T_{CPU} is the actual running time, T_m is the maximum value of running time, N is the number of deer in the population, N_{male} is the number of male deer in the population, N_{com} is the number of commanders, N_{stag} is the number of stags, and *best* records the best result and is initialized to ∞.*/

2: Randomly generate the initial population of red deers;

3: **while**($T_{CPU} < T_m$){

4: if(gen = 0){

for(each deer in the population){
Decode individual S_i to obtain the transition sequence $\tau(S_i)'(i \in [1, N])$;
RA(S_i, $\tau(S_i)$)
Calculate the fitness value of S_i, i.e., $F(S_i) = C_{max}(\tau(S_i))$;
if($F(S_i) < best$){$\pi := S_i$ and $best := F(S_i)$;}
}**end for**
}
Select the best N_{male} individual as male deer;
For(each male deer in the population){/*Roar male red deers. */
Update the position vector and part number sequence of each male deer, denoted the new individual by $S_i'(i \in [1, N_{male}])$;
Decode S_i' to obtain the transition sequence $\tau(S_i')$;
Calculate the fitness value of S_i', i.e., $F(S_i') = C_{max}(\tau(S_i'))$;
if($F(S_i') < F(S_i)$){ $S_i := S_i'$;}
}**end for**
Select the best N_{com} individuals from male deers as commanders;
for(each commander in the population){/*Fight between commanders and stags. */
Randomly select a stag, denoted by S_k ($k \in [6, N_{male}]$);
Update the position vector and part-number sequence of the commander, denoted the new individual by S_i', $i \in [1, N_{com})$;
Decode S_i' to obtain the transition sequence $\tau(S_i')$;
Calculate the fitness value of S_i', i.e., $F(S_i') = C_{max}(\tau(S_i'))$;
if($F(S_i') < F(S_i)$){ $S_i := S_i'$;}
else if($F(S_i') > F(S_i)$ and $F(S_i') < F(S_k)$){$S_k := S_i'$;}
}**end for**
for(each commander in the population){Form its harem, denoted by Harem[i], $i \in [1, N_{com}]$;}/*Harem[i] is a list storing the hinds in the harem of commander S_i.*/
for(each commander in the population){
α = rand(0, 1); $mate_i$ = round(α|Harem[i]|), $i \in [1, N_{com}]$;/* $mate_i$ is the number of hinds commander i choose to mate, |Harem[i]| is the number of hinds for commander i.*/
Randomly select $mate_i$ different deers in i-th commander's harem, and generated $mate_i$ offspring, record the offspring in OFFSPRING; }/*OFFSPRING is a list for storing offspring.*/
for(each commander in the population){
Randomly select another commander j;
β = rand(0, 1); $MATE_i$ = round(β|Harem[j]|), $i \in [1, N_{com}]$;/*$MATE_i$ is the number of hinds commander i choose to mate. */
Randomly select $MATE_i$ different deers in j-th commander's harem, and generated $MATE_i$ offspring, record the offspring in OFFSPRING; }
for(each stag in the population){
Mate with the nearest hind, generate an offspring, record the offspring in OFFSPRING;}
Repair each offspring in OFFSPRING and calculate its fitness value;
From the individuals in the current population and the individuals in OFFSPRING, select the best N individuals to form the population of the next generation;
gen++;

T_{CPU} := *cputime*(); /**cputime*() is the running time of the algorithm. */
}**end while**
Output π and *best*.
End

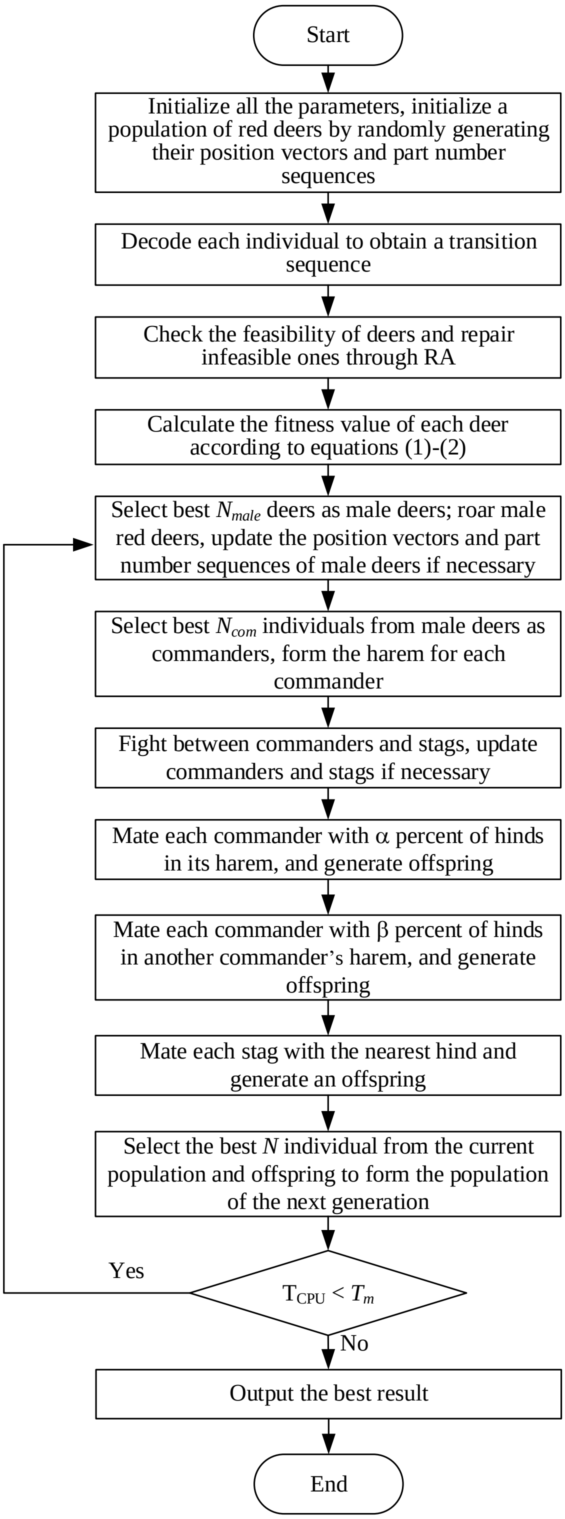

Fig. 6. Flowchart of RDA for scheduling FASs.

9. Social engineering optimizer for scheduling FASs

The pseudocode of social engineering optimizer (SEO) for scheduling FASs is shown in Algorithm 10, and the corresponding flowchart is presented in Fig. 7.

Algorithm 10: SEO

Input: An FAS data set;

Output: A feasible schedule α and its fitness value;

initialize: $T_{CPU} = 0$, $T_m = u \times C_R/5$, $N = 2$, $N_{attack} = 10$, $\beta_0 = 0.5\pi$, and $best = \infty$; /*T_{CPU} is the actual running time, T_m is the maximum value of running time, N is the number of individuals, N_{attack} is the number of attacks, and *best* records the best result and is initialized to ∞. */

Randomly generate two individuals;

while($T_{CPU} < T_m$){

  if(gen = 0){

    for(each individual){

      Decode individual S_i to obtain the transition sequence $\tau(S_i)$ ($i \in [1, N]$);

      RA(S_i, $\tau(S_i)$)

    }**end for**

  }

  Calculate the fitness value of each individual S_i, i.e., $F(S_i) = C_{max}(\tau(S_i))$;

  Set the individual with better fitness value as the attacker, and the other one as the defender;

  r_1 = rand(0, 1), $N_{train} = r_1 * K$;/* N_{train} is the number of times performing training and retraining, K is the length of a feasible solution.*/

  location = rand()%(K- N_{train});/* Select a starting point for performing training and retraining process.*/

  while($k < N_{train}$){ /* k is initialized as 0. */

    Replace the location-th element in the defender's array with the corresponding element in the attacker's array to form a new individual;

    Repair the newly generated individual, if its fitness value is better than the defender, update the defender;

    k++;

  }**end while**

  while($m \leq N_{attack}$){ /* m is initialized as 1. */

    $\beta = (m/N_{attack})\beta_0$;

    Spot an attack;

    Update defender if necessary;

    m++;

  }**end while**

  If the fitness value of defender is better than that of attacker, switch them;/*Respond to attack.*/

  Calculate the fitness value of each individual S_i, i.e., $F(S_i) = C_{max}(\tau(S_i))$;

  if(F(attacker) < *best*){α := attacker and *best* := F(attacker);}

  T_{CPU} := *cputime*(); /**cputime*() is the running time of the algorithm. */

}**end while**

Output α and *best*.

End

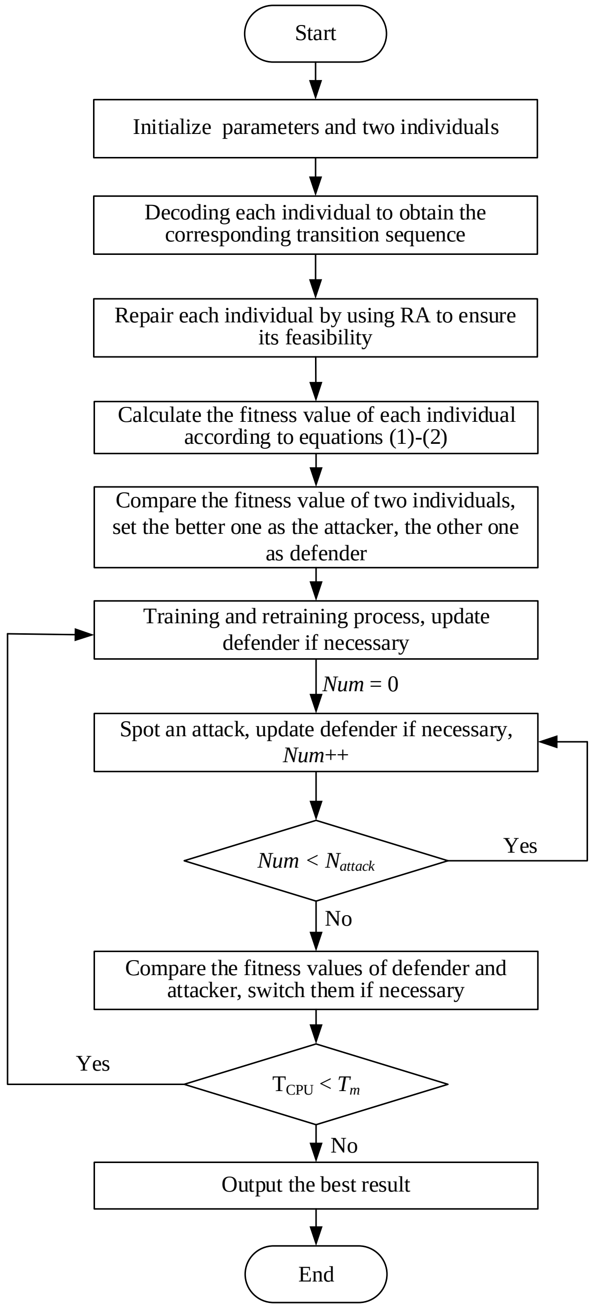

Fig. 7. Flowchart of SEO for scheduling FASs.