BÁO CÁO NHÓM 9

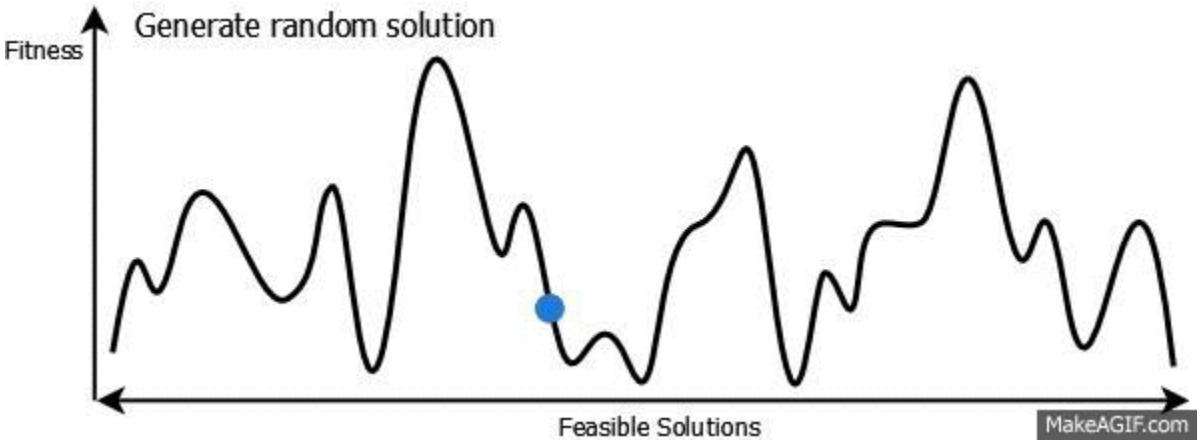
ĐỀ TÀI : LẬP LỊCH THI ĐẤU THỂ THAO

Bản báo cáo này trình bày một cách chi tiết hơn các giải thuật mà nhóm đã sử dụng trong bài tập lớn, bao gồm các giải thuật : Thuật toán ủ mô phỏng(SA), tìm kiếm leo đồi và Nhánh cận. Về phần sử dụng công cụ Ortools nhóm không trình bày ở đây.

I. Simulated annealing

1. Tổng quan

Thuật toán ủ mô phỏng (SA), được trình bày một cách độc lập như một thuật toán tìm kiếm cho các vấn đề tối ưu tổ hợp trong, là một thuật toán metaheuristic phổ biến được sử dụng rộng rãi để giải quyết các vấn đề tối ưu hóa rời rạc và liên tục. Tính năng chính của thuật toán SA nằm ở phương tiện để thoát khỏi tối ưu cục bộ bằng cách cho phép các động tác leo đồi tìm thấy tối ưu toàn cầu.

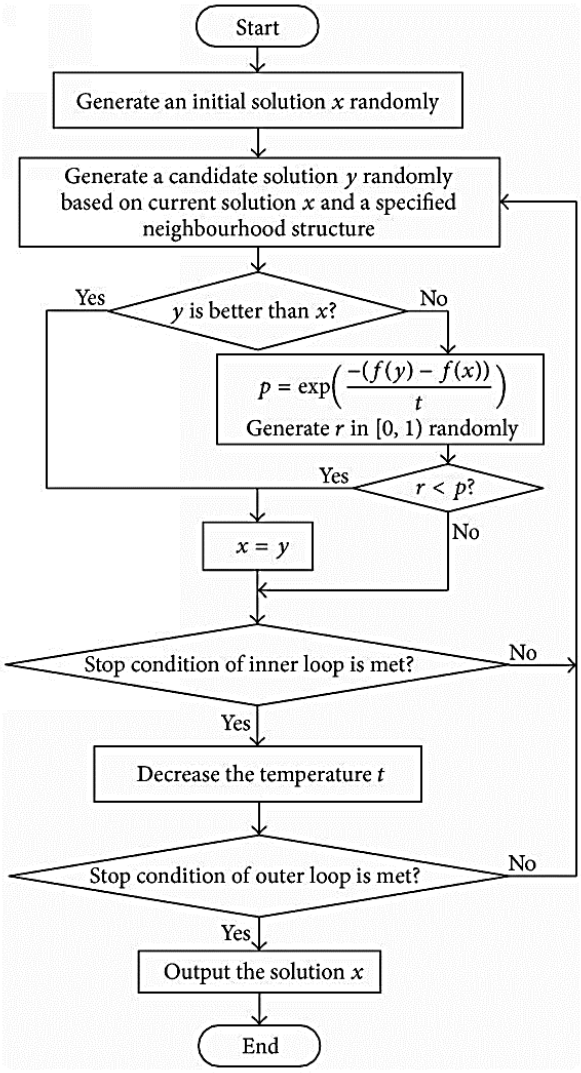


Thuật toán dựa trên nguyên lý của việc tôi ủ kim loại, đó là khi nhiệt độ cao ta có thể dễ dàng tùy chỉnh kim loại nóng chảy thành các hình dáng, ngược lại khi nhiệt độ giảm dần kim loại sẽ ít thay đổi hình thái hơn và cuối cùng nguội lạnh thành một hình dáng mong muốn. Thuật toán SA được biểu diễn theo 1 xác suất dựa trên biến "nhiệt độ" giảm dần theo thời gian, cho phép mở rộng không gian tìm kiếm, chấp nhận các lời giải tồi hơn hiện tại với xác suất giảm dần. Khi nhiệt độ rất thấp ta sẽ chỉ thấy thuật toán giống như leo đồi, luôn dao động tronmg một vùng lân cận nhỏ và hướng về điểm cực trị.

Thuật toán gồm 4 bước chính lặp lại:

- Bước 1: Khởi tạo lời giải ngẫu nhiên
- Bước 2: Tìm kiếm lân cận (*neighborhood*) bằng các dịch chuyển (*moves*)
- Bước 3: Đánh giá hàm mục tiêu của lời giải mới tìm được:
    - Chấp nhận nếu lời giải tốt hơn
    - Chấp nhận tồi hơn với xác suất ***p*** dựa vào **T**
- Bước 4: Giảm nhiệt độ **T** và lăp lại các bước tới khi gặp điều kiện dừng

Ý tưởng của thuật toán được mô tả qua lưu đồ giải thuật sau:



Các thành phần chính:

- Init solution:
    - Khởi tạo lời giải heuristic
    - Random
- Neighborhood:
    - Random

- Biến đổi lời giải hiện tại

- Acceptance:
  - Neighbor có hàm mục tiêu nhỏ hơn
  - Neighbor có hàm mục tiêu lớn hơn với xác suất *p*
- Stopping:
  - Max time, Max iterations
  - Lời giải có giá trị hàm mục tiêu nhỏ hơn ngưỡng θ
  - Max iterations without improvement

2. Init Solution

Việc mã hóa không gian lời giải vô cùng quan trọng, giúp ta dễ dàng biến đổi lời giải, đánh giá hàm mục tiêu của các lời giải,…Thông thường, ta sẽ sử dụng heuristic để xây dựng chiến lược lời giải tốt, tuy nhiên ở bài toán này để đơn giản ta sẽ khởi tạo random và mã hóa theo cơ chế dưới đây nhằm thỏa mãn rang buộc thi đấu vòng tròn.

Giả sử có n (chẵn) đội:

- Ma trận chi phí d(nxn) biểu diễn d_{ij} là khoảng cách đi từ đội T_i đến T_j
- Có 2n-2 round (tuần) diễn ra

Lời giải: Ma trận A (nxn):

A(i,k) = { **j** nếu đội **Ti** đấu với đội **Tj** ở round **k** tại sân nhà của **i**
- j nếu đội **Ti** đấu với đội **Tj** ở round **k** tại sân nhà của **i**

i, j ∈ 1,2,…n, k ∈ 1,2,…2n-2

Ví dụ: với n=4, ta có ma trận lời giải:

T\R	1	2	3	4	5	6	7	8	9	10
1	6	-2	4	3	-5	-4	-3	5	2	-6
2	5	1	-3	-6	4	3	6	-4	-1	-5
3	-4	5	2	-1	6	-2	1	-6	-5	4
4	3	6	-1	-5	-2	1	5	2	-6	-3
5	-2	-3	6	4	1	-6	-4	-1	3	2
6	-1	-4	-5	2	-3	5	-2	3	4	1

Hàm mục tiêu với team T_i:

costT(i)=k=12n-2dDi,Di,k,Di,Di,k+1+di,Di, 0+dDi,2n-2,i

Với D(i,k) = {Ai,k nếu Ai,k<0 i nếu ngược lại

Hàm mục tiêu tổng quát:

cost= i=1ncostT(i)

3. The Neighborhood

Vùng lân cận của một lịch trình S là tập hợp các lịch trình (khả thi) có thể thu được bằng cách áp dụng một trong năm loại di chuyển. Ba loại di chuyển đầu tiên có ý nghĩa trực quan đơn giản, trong khi hai loại cuối cùng khái quát chúng.

1. SwapHomes(S, Ti, Tj)

Đổi chỗ sân khách/sân nhà thi đấu giữa team T_i vàT_j sdfds

T\R	1	2	3	4	5	6	7	8	9	10
1	6	-2	4	3	-5	-4	-3	5	2	-6
2	5	1	-3	-6	4	3	6	-4	-1	-5
3	-4	5	2	-1	6	-2	1	-6	-5	4
4	3	6	-1	-5	-2	1	5	2	-6	-3
5	-2	-3	6	4	1	-6	-4	-1	3	2
6	-1	-4	-5	2	-3	5	-2	3	4	1

SwapHomes(S, T2, T4)

T\R	1	2	3	4	5	6	7	8	9	10
1	6	-2	4	3	-5	-4	-3	5	2	-6
2	5	1	-3	-6	-4	3	6	4	-1	-5
3	-4	5	2	-1	6	-2	1	-6	-5	4
4	3	6	-1	-5	2	1	5	-2	-6	-3
5	-2	-3	6	4	1	-6	-4	-1	3	2
6	-1	-4	-5	2	-3	5	-2	3	4	1

2. SwapRounds(S, r_k, r_l)

Đổi chỗ round (tuần) thi đấu r_k và r_l

T\R	1	2	**3**	4	**5**	6	7	8	9	10
1	6	-2	4	3	-5	-4	-3	5	2	-6
2	5	1	-3	-6	4	3	6	-4	-1	-5
3	-4	5	2	-1	6	-2	1	-6	-5	4
4	3	6	-1	-5	-2	1	5	2	-6	-3
5	-2	-3	6	4	1	-6	-4	-1	3	2
6	-1	-4	-5	2	-3	5	-2	3	4	1

T\R	1	2	**3**	4	**5**	6	7	8	9	10
1	6	-2	-5	3	4	-4	-3	5	2	-6
2	5	1	4	-6	-3	3	6	-4	-1	-5
3	-4	5	6	-1	2	-2	1	-6	-5	4
4	3	6	-2	-5	-1	1	5	2	-6	-3
5	-2	-3	1	4	6	-6	-4	-1	3	2
6	-1	-4	-3	2	-5	5	-2	3	4	1

3. SwapTeams(S, Ti, Tj)

Đổi lịch thi đấu team i và j (ngoại trừ trận giữa T_i và T_j)

T\R	1	2	3	4	5	6	7	8	9	10
1	6	-2	4	3	-5	-4	-3	5	2	-6
2	5	1	-3	-6	4	3	6	-4	-1	-5
3	-4	5	2	-1	6	-2	1	-6	-5	4
4	3	6	-1	-5	-2	1	5	2	-6	-3
5	-2	-3	6	4	1	-6	-4	-1	3	2
6	-1	-4	-5	2	-3	5	-2	3	4	1

SwapTeams(S, T2, T5)

T\R	1	2	3	4	5	6	7	8	9	10
1	6	-5	4	3	-2	-4	-3	2	5	-6
2	5	-3	6	4	1	-6	-4	-1	3	-5
3	-4	2	5	-1	6	-5	1	-6	-2	4
4	3	6	-1	-2	-5	1	2	5	-6	-3
5	-2	1	-3	-6	4	3	6	-4	-1	2
6	-1	-4	-2	5	-3	2	-5	3	4	1

4. PartialSwapRounds(S, Ti, rk, rl)

Team T_i đổi chỗ round r_l và r_k

T\R	1	**2**	3	4	5	6	7	8	**9**	10
1	6	-2	2	3	-5	-4	-3	5	4	-6
2	5	1	-1	-5	4	3	6	-4	-6	-3
3	-4	5	4	-1	6	-2	1	-6	-5	2
4	3	6	-3	-6	-2	1	5	2	-1	-5
5	-2	-3	6	2	1	-6	-4	-1	3	4
6	-1	-4	-5	4	-3	5	-2	3	2	1

PartialSwapRounds(S, T2, r2,r9)

T\R	1	**2**	3	4	5	6	7	8	**9**	10
1	6	4	2	3	-5	-4	-3	5	-2	-6
2	5	-6	-1	-5	4	3	6	-4	1	-3
3	-4	5	4	-1	6	-2	1	-6	-5	2
4	3	-1	-3	-6	-2	1	5	2	6	-5
5	-2	-3	6	2	1	-6	-4	-1	3	4
6	-1	2	-5	4	-3	5	-2	3	-4	1

Việc sửa đổi thành phần như vậy gây ra vi phạm về ràng buộc đấu vòng tròn, do đó tại bước dịch chuyển ta cần cập nhật tuần tự lại các trận đấu vị vi phạm, điều này luôn có duy nhất một cách cập nhật và tối đa $O(n^3)$ phép di chuyển.

5. PartialSwapTeams(S, Ti, Tj, rk)

Đổi chỗ Team T_i và Team T_i tại round r_k

T\R	1	2	3	4	5	6	7	8	**9**	10
1	6	-2	4	3	-5	-4	-3	5	2	-6
2	5	1	-3	-6	4	3	6	-4	-1	-5
3	-4	5	2	-1	6	-2	1	-6	-5	4
4	3	6	-1	-5	-2	1	5	2	-6	-3
5	-2	-3	6	4	1	-6	-4	-1	3	2
6	-1	-4	-5	2	-3	5	-2	3	4	1

PartialSwapRounds(S, T2, T4, r9)

T\R	1	2	3	4	5	6	7	8	**9**	10
1	6	-2	2	3	-5	-4	-3	5	4	-6
2	5	1	-1	-5	4	3	6	-4	-6	-3
3	-4	5	4	-1	6	-2	1	-6	-5	2
4	3	6	-3	-6	-2	1	5	2	-1	-5
5	-2	-3	6	2	1	-6	-4	-1	3	4
6	-1	-4	-5	4	-3	5	-2	3	2	1

Tương tự như PartialSwapRounds, ta sẽ cập nhật các vi phạm trên round với chi phí $O(n^3)$

4. Local Search

Như thường lệ, thuật toán bắt đầu từ một cấu hình ban đầu. Bước di chuyển (moves) của nó di chuyển từ cấu hình con hiện tại **c** sang một cấu hình trong vùng lân cận của **c**.

SA bắt đầu từ một lịch trình ban đầu ngẫu nhiên thu được bằng cách sử dụng tìm kiếm quay lui đơn giản. Không có sự chú ý đặc biệt nào được dành cho thuật toán này và lịch trình khả thi đã dễ dàng thu được.

SA sau đó tuân theo lược đồ thuật toán ủ mô phỏng truyền thống. Đưa ra một temperature **T,** thuật toán chọn ngẫu nhiên một trong các động tác dịch chuyển (moves) trong lân cận và tính toán biến thiên **Δf** của hàm mục tiêu được tạo ra bởi sự di chuyển. Nếu **Δ < 0**, TTSA áp dụng di chuyển. Ngược lại, nó áp dụng di chuyển với xác suất **exp(−Δ/t).**

Như điển hình trong việc ủ mô phỏng, xác suất chấp nhận một động thái không cải thiện giảm dần theo thời gian. Hành vi này có được bằng cách giảm nhiệt độ như sau: TTSA sử dụng một bộ đếm biến *counter* được tăng lên cho mỗi động tác di chuyển không cải tiến (**Δ > 0**) và đặt lại về 0 khi giải pháp tốt nhất được tìm thấy cho đến nay được cải thiện. Khi bộ đếm đạt đến một giới hạn trên cụ thể, nhiệt độ được cập nhật thành **T · β** (trong đó β là một hằng số cố định nhỏ hơn 1) và *counter* được đặt lại về 0. Hình 1 mô tả thuật toán ủ mô phỏng chi tiết hơn. Thuật toán giữ một đại diện ngầm của lân cận như một tập hợp các cặp và bộ ba, vì tất cả các động tác có thể được thể

hiện theo cách này. Chẳng hạn, PartialSwapteam (S, T_i, T_j, r_k) được đặc trưng bởi các bộ ba của Mẫu <T_i, T_j, r_k>.

5. Reheats

Tính năng cuối cùng của TTSA là việc sử dụng reheats, một sơ đồ tiêu chuẩn chung mô phỏng một làm mát đã được đề xuất bởi một số tác giả (ví dụ, xem, [8]). Ý tưởng cơ bản là, một khi mô phỏng ủ đạt đến nhiệt độ rất thấp, nó gặp khó khăn trong việc ES CAPE từ cực tiểu cục bộ, bởi vì xác suất chấp nhận các động tác không giảm là rất thấp. Đ nóng lại là ý tưởng tăng nhiệt độ một lần nữa để thoát khỏi mức tối thiểu cục bộ hiện tại. TTSA sử dụng một phương pháp hâm nóng tương đối đơn giản. Ý tưởng là hâm nóng sau khi hoàn thành vòng lặp ngoài cùng bằng cách tăng nhiệt độ lên gấp đôi giá trị của nó khi tìm thấy giải pháp tốt nhất. TTSA hiện chấm dứt khi số lượng hâm nóng liên tiếp mà không cải thiện giải pháp tốt nhất đạt đến giới hạn nhất định. Thuật toán bao gồm tất cả các sửa đổi này được hiển thị trong hình

6. Cài đặt & Đánh giá

Giải thuật SA cho bài toán giải đấu thể thao được mô tả qua mã giả dưới đây:

```
1. find random schedule S;
2. bestSoFar ← C(S); nbf ← ∞;
3. bestInfeasible ← ∞; nbi ← ∞;
4. reheat ← 0; counter ← 0;
5. while reheat ≤ maxR do
6.  phase ← 0;
7.  while phase ≤ maxP do
8.     counter ← 0;
9.     while counter ≤ maxC do
10.       select a random move m from neighborhood (S);
11.       //let S_new be the schedule obtained from S with m;
12.       if C(S_new) < C(S)
16.          accept ← true;
17.       else
18.          accept ← true with probability exp(−ΔC/T ),
19.                false otherwise;
20.       end if
21.       if accept then
22.          S ← S_new;
23.          if nbv(S) == 0 then
24.             nbf ← min(C(S), bestFeasible);
25.          else
26.             nbi ← min(C(S), bestInfeasible);
27.          end if
28.          if C(S_new) < bestSoFar then
29.             reheat ← 0; counter ← 0; phase ← 0;
30.             bestTemperature ← T ;
31.             bestSoFar ← C(S_new) ;
34.          else
35.             counter++;
36.          end if
```

```
    end while
  phase++;
  T ← T · β;
end while
reheat++;
T ← 2 · bestTemperature;
end while
```

Các tham số mặc định:

maxP	1250
maxC	1000
maxR	10
β	0.955
T	max(distance_matrix) * n/2

Đánh giá:

Đem lại kết quả chính xác với số đội nhỏ (4-6), với số đội lớn kết quả có xu hướng cải thiện tốt, gần đúng hàm mục tiêu.

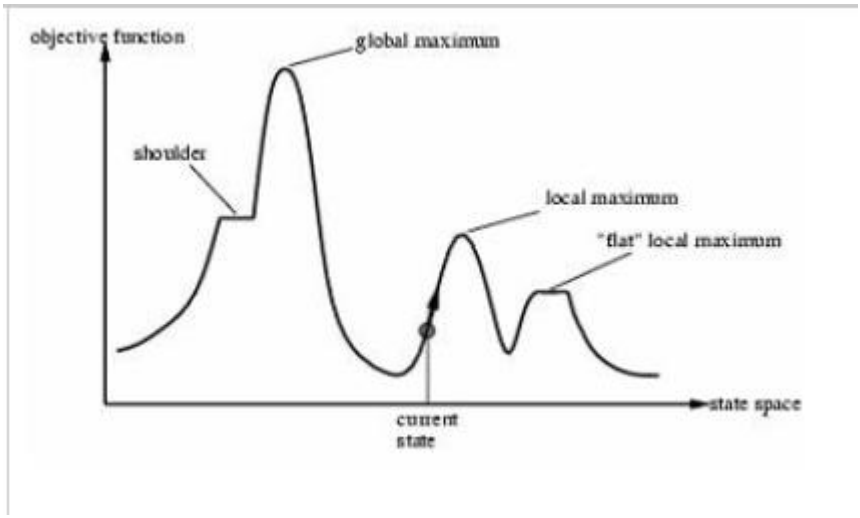
n	Init T	Init cost	Phase	Reheat	t(s)	Result Cost
4	2000	10257	2	0	0.2187	**8276**
6	8000	33538	312	1	4.6875	**19900**
10	10000	81273	696	15	19.1406	**48110**
12	20000	146443	996	10	597.0312	**93790**
14	40000	329475	715	7	283.3906	**170394**
16	60000	463200	927	13	195.5	**218317**

- Ưu điểm:
    - Tính đơn giản
    - Khả năng thoát khỏi tối ưu cục bộ
    - Có tính mở rộng cao (sử dụng thêm meta-heuristics)
- Nhược điểm:
    - Cần turning tham số nhiều

- Lựa chọn T và chiến lược giảm T ảnh hưởng tới hiệu quả
- Thời gian hội tụ lâu với số lượng đội lớn.

II. THUẬT TOÁN LEO ĐỒI

Ý tưởng chung của thuật toán: Gỉa sử hàm số của ta là f(x) với x là số thực, đầu tiên ta chọn một điểm rơi x, sau đó di chuyển x tới vùng lân cận mà có f(x) cao hơn (hoặc thấp hơn) và cuối cùng dừng lại ở một cực trị.



Khởi tạo các trận đấu:

Bước 1: Trả về giải pháp ngẫu nhiên bằng cách ghép đầu cuối và di chuyển phần tử khác từ trái qua phải.

Bước 2: Đưa ra giải pháp cho lượt về giống lượt đi.

Bước 3: Đổi kết quả lượt đi dạng (a,b) về a<b.

Bước 4: Tương tự với lượt đi dạng (a,b) về a>b.

Mỗi lần lặp ta sẽ chọn ngẫu nhiên tráo đổi 2 round hay swap sân nhà, sân khách.

Nếu chọn tráo đổi 2 round: Duyệt tất cả các cặp (i,j) là cá round đấu.

Bước 1: Thử swap 2 round đấu, tính kết quả khi swap.

Bước 2: Nếu cách này tốt hơn kết quả hiện tại, cập nhật kết quả.

Nếu chọn tráo đổi 2 game: Duyệt tất cả các cặp trận đấu (u,v)

Bước 1: Swap sân nhà và khách của 2 trận đấu (u,v) và (v,u), tính kết qủa khi thay đổi.

Bước 2: Nếu cách này tốt hơn kết quả hiện tại, cập nhật lại kết quả.

Thuật toán kết thúc khi tối ưu cục bộ, không tìm thấy lân cận tốt hơn.

III. Mô hình nhánh cận

1. Tổ chức dữ liệu

- n: Số lượng đội tuyển cần lập lịch thi đấu.
- path[0…n][0…n]: Khoảng cách giữa sân của các đội tuyển.

  VD: Khoảng cách từ sân của đội tuyển i đến sân của đội tuyển j là path[i][j].
- curPos[0…n]: Vị trí hiện tại của từng đội trong khi duyệt quay lui.
- inMatch[0…n][0…n]: Trận đấu giữa đội i và j đã được xếp lịch hay chưa trong khi duyệt quay lui. Nếu có inMatch[i][j] = 1, nếu không inMatch[i][j] = 0.
- hasMatchInRound[0…2n-2][0…n]: Trận đấu j đã được xếp lịch trong round i hay chưa trong khi duyệt quay lui. Nếu có hasMatchInRound[i][j] = 1, nếu không hasMatchInRound[i][j] = 0.
- curPath: Tổng khoảng cách di chuyển hiện tại của các đội trong quá trình duyệt quay lui.
- finalResult: Tổng khoảng cách di chuyển nhỏ nhất của các đội.
- traceMatch: Một ngăn xếp lưu lại trận đấu giữa các đội trong quá trình quay lui.
- traceResult: Một ngăn xếp lưu lại một kế hoạch thi đấu của phương án có tổng khoảng cách di chuyển nhỏ nhất của các đội.

2. Thuật toán nhánh cận

- Hàm **solve**(round): Dùng để đệ qui các vòng trong lịch thi đấu. Điểm dừng: khi round = 2*n-1 sẽ cập nhật lại tổng khoảng cách di chuyển nhỏ nhất giữa các đội, kế hoạch thi đấu cho phương án đó và return.

```
solve(round) {
    if(round == 2*n - 1) {
        [update finalResult, traceResult]
        return
    }
    calcEachRound(round, 1, 1, 1)
}
```

- Hàm **calcEachRound**(round, prevHome, prevGuest, match): Dùng để đệ quy các trận đấu trong 1 vòng của lịch thi đấu. Điểm dừng: khi match = n/2 + 1 sẽ gọi hàm **solve**(round + 1) để tính toán cho vòng tiếp theo.

```
calcEachRound(round, prevHome, prevGuest, match) {
    if(match == n/2 + 1) {
       solve(round + 1)
    }
    Foreach(i from prevHome to n) {
       Foreach(j from prevGuest + 1 if i == prevHome else 1 to n) {
          if(checkExistence(round, i, j) && curPath + travelPath < finalResult)  {
             [update curPos, curPath, hasMatchInRound, inMatch, traceMatch]
             calcEachRound(round, i, j, match + 1)
             [update curPos, curPath, hasMatchInRound, inMatch, traceMatch]
          }
       }
    }
}
```

- Hàm **checkExistence**(round, i, j): Dủng để kiểm tra các điều kiện ràng buộc khi xem xét 1 trận đấu với kế hoạch hiện tại.

```
checkExistence(round, i, j) {
   return i != j && !hasMatchInRound[round][i] && !hasMatchInRound[round][j] &&
!inMatch[i][j]
}
```

- Hàm **main**(): Hàm chính của chương trình

```
main() {
   [take input n, path]
   finalResult =∞
   Foreach(i from 1 to n) curPos[i] = i
   solve(1)
   [print traceResult and finalResult]
}
```

IV. THAM KHẢO

Zhan, S. H., Lin, J., Zhang, Z. J., & Zhong, Y. W. (2016). List-based simulated annealing algorithm for traveling salesman problem. *Computational intelligence and neuroscience*, *2016*.

Anagnostopoulos, A., Michel, L., Hentenryck, P. V., & Vergados, Y. (2006). A simulated annealing approach to the traveling tournament problem. *Journal of Scheduling*, *9*(2), 177-193.