Bài toán người du lịch và Phương pháp tối ưu bầy đàn

Ngô Minh Hải
Phan Xuân Phúc

Ngày 18 tháng 2 năm 2019

Mục lục

Giới thiệu về bài toán người du lịch

Bài toán người du lịch (the **T**raveling **S**alesman **P**roblem hay *TSP*) là một trong những bài toán tối ưu tổ hợp NP-khó được nghiên cứu nhiều nhất.

Bài toán có thể được mô tả đơn giản như sau: Tìm đường đi tốn ít chi phí nhất qua tất cả các thành phố cho trước.

Giới thiệu về bài toán người du lịch

Cụ thể: Một người mong muốn có thể thực hiện một chuyến đi qua n thành phố, sao cho người đó thăm mỗi thành phố đúng một lần và kết thúc tại thành phố đầu tiên người đó thăm.

Để đi từ thành phố này tới thành phố khác cần tiêu tốn một chi phí cụ thể. Bài toán yêu cầu phải tối thiểu hóa được tổng chi phí của chuyến đi.

Mô hình hóa bài toán

Bài toán có thể được mô hình hóa thành một đồ thị với n đỉnh tương ứng với n thành phố.

Khi đi từ thành phố i sang thành phố j thì người đó tiêu tốn một chi phí, ta gọi là c_{ij}.

Với phiên bản *TSP* quyết định ($DTSP$), yêu cầu của bài toán sẽ là có tồn tại một hành trình đi qua tất cả các thành phố đúng một lần với tổng chi phí không vượt quá k hay không?

Mô hình hóa bài toán

$$
\begin{aligned}
\text{tối thiểu hóa} \quad & \sum_{(i,j)\in E} c_{ij} x_{ij}, \\
\text{sao cho} \quad & \sum_{j\in V} x_{ij} = 2 \quad \forall i \in V, \\
& \sum_{i,j\in S, i\neq j} x_{ij} \le |S| - 1 \quad \forall S \subset V, S \neq \emptyset, \\
& x_{ij} \in \{0, 1\}. \\
\text{trong đó} \quad & V \text{ là tập đỉnh}, \\
& E \text{ là tập cạnh}, \\
& c_{ij} \text{ là chi phí đi từ nút } i \text{ tới nút } j.
\end{aligned}
$$

TSP là NP-khó

Để chứng minh *TSP* là NP-khó, ta cần thực hiện các bước sau:

- Chứng minh $DTSP \in$ NP (có thể dễ dàng chứng minh),
- Chứng minh $DTSP \in$ NP-đầy đủ (chứng minh dựa trên tính qui dẫn: $\textit{Ham-Cycle}$[1] $\leq_p DTSP$)
- Chứng minh $TSP \in$ NP-khó (chứng minh dựa trên tính qui dẫn: $DTSP \leq_p TSP$)

[1] Chu trình Hamilton: bài toán *TSP* chính là tìm ra chu trình Hamilton trên đồ thị với tổng chi phí ít nhất hoặc không vượt quá k.

Giới thiệu

Phương pháp tối ưu bầy đàn (**P**article **S**warm **O**ptimization hay PSO) là một thuật toán metaheuristic[2] dùng để giải quyết các bài toán tối ưu hóa.

Được lần đầu công bố bởi J. Kennedy, R. C. Eberhart, và Y. Shi năm 1995, thuật toán sử dụng khái niệm về trí tuệ bầy đàn để nhằm tìm ra được một phương án chấp nhận được cho bài toán tối ưu. Điều này mô phỏng lại hành vi của bầy chim tìm mồi hay bầy cá dưới nước nên mới có tên gọi như vậy.

[2] Metaheuristic là nhóm các thủ tục hay tri thức nhằm tìm ra, tạo ra, hoặc chọn lấy một tri thức có thể cho ta một phương án đủ tốt cho các bài toán tối ưu.

Mô tả thuật toán

Khởi tạo: PSO khởi tạo bằng một nhóm cá thể (particle swarm) ngẫu nhiên.

Yêu cầu: Thuật toán có nhiệm vụ tìm nghiệm tối ưu bằng cách tìm các cá thể tốt hơn qua từng thế hệ.

Cụ thể: Với mỗi thế hệ (tương ứng với một vòng lặp), mỗi cá thể sẽ được cập nhật hai giá trị tốt nhất.
Giá trị thứ nhất là nghiệm tốt nhất đạt được của cá thể đó tới thời điểm hiện tại, gọi là p_{best}.
Giá trị thứ hai là nghiệm tốt nhất toàn cục, gọi là g_{best}.

Mô tả thuật toán

Hai giá trị p_{best} và g_{best} càng tiến gần tới tiêu chuẩn hội tụ thì ta có thể coi chúng chính là giá trị nghiệm tốt nhất.

Để làm được điều đó, ta cần thay đổi vị trí của các cá thể trong bầy sao cho chúng tiến gần tới nghiệm tối ưu nhất.

Mô tả thuật toán

PSO đưa ra công thức tính vận tốc và vị trí của mỗi cá thể qua từng thế hệ như sau:

$$v_i^{k+1} = w \cdot v_i^k + c_1 \cdot rand_1() \cdot (p_{\text{best}} - x_i^k) + c_2 \cdot rand_2() \cdot (g_{\text{best}} - x_i^k),$$

$$x_i^{k+1} = x_i^k + v_i^{k+1}.$$

Trong đó: v_i^{k+1}, v_i^k: vận tốc của cá thể thứ i trong thế hệ thứ $k+1$ và k,
x_i^{k+1}, x_i^k: vị trí của cá thể thứ i trong thế hệ thứ $k+1$ và k,
w: trọng số quán tính—tự định nghĩa, phụ thuộc từng bài toán cụ thể,
c_1, c_2: các hệ số gia tốc—tự định nghĩa, phụ thuộc từng bài toán cụ thể,
$rand_1(), rand_2()$: các số ngẫu nhiên từ 0 tới 1, tăng tính ngẫu nhiên của thuật toán.

Mã giả

INITIALIZEPARTICLESWARM()

$population = \emptyset$
$g_{\text{best}} = 0$
for $i = 1$ **to** $population.size$
    $p_v = randomVelocity()$
    $p_x = randomPosition(population.size)$
    $p_{\text{best}} = p_x$
    if $Cost(p_{\text{best}}) \leq Cost(g_{\text{best}})$
        $g_{\text{best}} = p_{\text{best}}$

Mã giả

$\textsc{ParticleSwarmOptimization}()$

$\quad\textsc{InitializeParticleSwarm}()$
$\quad$ **while** $!stopCondition$
$\quad\quad$ **for** each $p \in population$
$\quad\quad\quad p_v = updateVelocity(p_v, g_{\text{best}}, p_{\text{best}})$
$\quad\quad\quad p_x = updatePosition(p_x, p_v)$
$\quad\quad\quad$ **if** $Cost(p_x) \leq Cost(p_{\text{best}})$
$\quad\quad\quad\quad p_{\text{best}} = p_x$
$\quad\quad\quad\quad$ **if** $Cost(p_{\text{best}}) \leq Cost(g_{\text{best}})$
$\quad\quad\quad\quad\quad g_{\text{best}} = p_{\text{best}}$
$\quad$ **return** g_{best}

Ứng dụng PSO trong TSP

Với bài toán TSP, ta định nghĩa mỗi cá thể của bầy là một hành trình qua các thành phố.

Ban đầu, các hành trình này được khởi tạo ngẫu nhiên và đưa vào bầy.

Chi phí của mỗi hành trình sẽ là tiêu chí so sánh để quyết định vị trí, vận tốc, và vị trí tốt nhất của từng cá thể (hay hành trình) trong bầy.

Ứng dụng PSO trong TSP

Cụ thể:
Đầu tiên, ta khởi tạo ngẫu nhiên các thành phố dựa trên tọa độ của chúng trên một mặt phẳng tọa độ.
Mỗi thành phố sẽ có tọa độ (x, y) riêng và dễ dàng tính khoảng cách bằng công thức:

$$c_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \ \forall i, j \in V.$$

Sau khi khởi tạo tọa độ cho các thành phố, ta tiến hành tạo ngẫu nhiên các hành trình và tính tổng chi phí đi trên từng hành trình ấy. Ta sẽ gán cho từng hành trình p_{best} chính là tổng chi phí của từng hành trình.

$$p_{\text{best}} = \sum_{i,j \in V} c_{ij}.$$

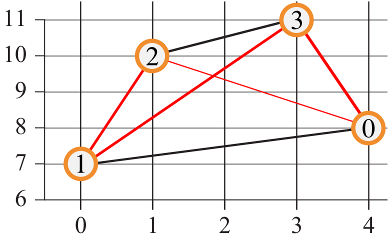
Ứng dụng PSO trong TSP

Từ các hành trình đó, ta xây dựng các cá thể gồm ba thuộc tính:

- *data* là dữ liệu của cá thể, với bài toán *TSP* chính là hành trình thực hiện.
- *pBest* là vị trí tốt nhất hiện tại, bằng tổng chi phí của hành trình.
- *velocity* là vận tốc của cá thể, là mức độ thay đổi hành trình (hay *data*) của cá thể.

Ứng dụng PSO trong TSP

Ví dụ về khởi tạo một cá thể, hành trình được sinh ra ngẫu nhiên:



$\Rightarrow$ particle 0:

$$\text{data}[] = \{0, 3, 1, 2, 0\},$$

$$\text{pBest} = \sum_{i,j \in 0,1,2,3} c_{ij}$$

$$= 14.93010659580074,$$

$$\text{velocity} = 0.0.$$

Đường đi [0, 3, 1, 2, 0]

$$\begin{aligned} c_{03} &= \sqrt{(x_0 - x_3)^2 + (y_0 - y_3)^2} \\ &= \sqrt{(4-3)^2 + (8-11)^2} \\ &= \sqrt{10} \\ &= 3.16227766016837 \\ c_{31} &= 5 \\ c_{12} &= 3.16227766016837 \\ c_{20} &= 3.60555127546398 \end{aligned}$$

Ứng dụng PSO trong TSP

Sau phần khởi tạo mô hình, ta tiến hành thực hiện giải thuật:

B1: Tiến hành sắp xếp danh sách các cá thể theo thứ tự p_{best} giảm dần. g_{best} sẽ bằng p_{best} của cá thể đứng dầu.

B2: Tính *velocity* của từng cá thể bằng công thức:

$$velocity = velocity + V\_WEIGHT \times rand() \times (g_{\text{best}} - p_{\text{best}}),$$

trong đó V_WEIGHT là trọng số gia tốc.

Trong giải thuật này vì mỗi lần sang thế hệ mới, các cá thể luôn cập nhật với p_{best} tốt hơn nên công thức có khác đôi chút với giải thuật gốc.

Ứng dụng PSO trong TSP

B3: Sau khi tính xong *velocity*, ta cập nhật các cá thể trong danh sách.
Để cập nhật, ta lấy phần nguyên của *velocity* từng cá thể làm tham số chỉ số lần thay đổi của cá thể. Ta cũng lợi dụng danh sách đã được sắp xếp sao chép một số vị trí thành phố trong hành trình của các cá thể tốt hơn (đứng trước cá thể đang xét) để có được hành trình tốt hơn.

Ứng dụng PSO trong TSP

B4: Giải thuật sẽ dừng khi đạt một trong ba điều kiện sau:

- Đã đạt tới số vòng lặp tối đa cho phép $= MAX\_ITERATIONS$,
- Đã tìm được hành trình có p_{best} thỏa hàm mục tiêu $f1$,
- Giá trị g_{best} không thay đổi trong một khoảng thời gian thỏa mãn hàm mục tiêu $f2$.

Trong đó,
$f1$ bằng chi phí mục tiêu mà tổng chi phí hành trình không được vượt quá (mặc định $f1 = TARGET = 0.0$). Nếu $\exists\ p_{\text{best}} \leq f1$ thì hành trình có p_{best} đó sẽ là nghiệm tối ưu của bài toán.
$f2$ là số vòng lặp tối đa mà giá trị tốt nhất toàn cục g_{best} không thay đổi. Vượt quá thời gian đó (mặc định $f2 = MAX\_LAPSE = 10$), hành trình có $p_{\text{best}} = g_{\text{best}}$ sẽ là nghiệm tối ưu của bài toán.
Nếu không thỏa cả ba điều kiện, quay lại B1.

Ví dụ minh họa

Ví dụ với trường hợp số thành phố = 15, các giá trị khởi tạo:

- Số cá thể trong bầy: $PARTICLE\_COUNT = 10$,
- Trọng số gia tốc: $V\_WEIGHT = 10$,
- Số vòng lặp tối đa: $MAX\_ITERATIONS = 10000$,
- Số vòng lặp g_{best} không đổi tối đa ($f2$): $MAX\_LAPSE = 10$,
- Miền tọa độ: $DOMAIN = 50$ (các tọa độ sẽ dao động trong đoạn $[0, 50]$),
- Giá trị hàm mục tiêu ($f1$): $TARGET = 0.0$,
- Chi phí tốt nhất toàn cục mặc định: $globalBest = Double.MAX\_VALUE$.

Ví dụ minh họa

Hình 1 cho thấy chương trình khởi tạo tọa độ các thành phố và tính toán các cá thể tương ứng với các hành trình — Route:

```
Number of Cities: 15
City 0: [12, 15]
City 1: [36, 39]
City 2: [45, 47]
City 3: [32, 18]
City 4: [49, 47]
City 5: [46, 4]
City 6: [38, 3]
City 7: [34, 12]
City 8: [9, 47]
City 9: [16, 26]
City 10: [22, 22]
City 11: [43, 15]
City 12: [1, 32]
City 13: [10, 6]
City 14: [47, 7]
Route: 10, 1, 5, 14, 3, 9, 0, 7, 2, 12, 4, 11, 6, 13, 8, 10, Distance: 408.3551544157211
Route: 1, 3, 0, 9, 4, 5, 6, 7, 8, 13, 2, 11, 12, 10, 14, 1, Distance: 454.99250521635986
Route: 3, 9, 10, 5, 2, 4, 6, 14, 8, 7, 13, 11, 12, 1, 0, 3, Distance: 449.6103256732552
Route: 3, 4, 2, 10, 6, 7, 12, 1, 0, 9, 11, 5, 8, 13, 14, 3, Distance: 420.1116271285374
Route: 5, 0, 10, 3, 4, 9, 2, 13, 8, 7, 12, 11, 6, 1, 14, 5, Distance: 475.1321290174883
Route: 2, 4, 0, 9, 12, 1, 6, 3, 10, 5, 7, 14, 8, 13, 11, 2, Distance: 400.2559123732567
Route: 9, 0, 8, 10, 1, 4, 12, 6, 2, 13, 3, 11, 5, 7, 14, 9, Distance: 417.64389156918594
Route: 3, 1, 5, 4, 6, 2, 13, 7, 11, 0, 12, 10, 8, 14, 9, 3, Distance: 491.029114900692
Route: 10, 6, 14, 1, 2, 5, 0, 7, 8, 13, 11, 3, 12, 9, 4, 10, Distance: 437.2311393154689
Route: 12, 6, 0, 2, 4, 5, 11, 10, 7, 8, 1, 3, 9, 13, 14, 12, Distance: 438.5630489850572
```

Hình: 1 Khởi tạo mô hình PSO

Ví dụ minh họa

Hình 2 cho thấy thuật toán đã chạy tới Bước 21, chưa đạt tới hàm mục tiêu $f1$ ($= 0.0$ hay hành trình không tốn chi phí) nhưng đã đạt tới thời gian g_{best} không đổi $f2$:

```
Step: 21
Target not reached
Shortest Route: 1, 2, 8, 3, 6, 5, 13, 10, 0, 9, 12, 11, 7, 14, 4, 1, Distance: 329.43469110363594
```

Hình: 2 Kết quả thuật toán PSO

Mã nguồn

Mã nguồn Java:
`https://thekuam.github.io/pso/`