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

Ngô Minh Hải
Phan Xuân Phúc

Ngày 17 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 $c(i, j)$.

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

$TSP = \{(G, c, k) : G = (V, E)$ với V là tập các đỉnh (vertices) và
E là tập các cạnh (edges),
c là hàm chi phí: $V \times V \to \mathbb{Z}$,
$k \in \mathbb{Z}$, với
k là số nguyên lớn nhất mà tổng chi phí
của chuyến đi không được vượt quá$\}$.

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: $Ham\text{-}Cycle$[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()

```
1  population = ∅
2  g_best = 0
3  for i = 1 to population.size
4        p_v = randomVelocity()
5        p_x = randomPosition(population.size)
6        p_best = p_x
7        if Cost(p_best) ≤ Cost(g_best)
8              g_best = p_best
```

Mã giả

```
ParticleSwarmOptimization()
 1  InitializeParticleSwarm()
 2  while !stopCondition
 3        for each p ∈ population
 4              p_v = updateVelocity(p_v, g_best, p_best)
 5              p_x = updatePosition(p_x, p_v)
 6              if Cost(p_x) ≤ Cost(p_best)
 7                    p_best = p_x
 8                    if Cost(p_best) ≤ Cost(g_best)
 9                          g_best = p_best
10  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.

Code Java:
`https://thekuam.github.io/pso/?fbclid=IwAR3SKXFOlumgquIG2NmTfY0S9gniFw6Wvzo4z56JkR1R2QTdRQBw0vp32QU`