ĐẠI HỌC QUỐC GIA TPHCM

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

BỘ MÔN CÔNG NGHỆ TRI THỨC

Đồ án 1: Search Algorithms

Đề tài: Các thuật toán trí tuệ bầy đàn

Môn học: Cơ sở trí tuệ nhân tạo

Sinh viên thực hiện:

Nguyễn Tấn Hùng (23122007)

Đoàn Hải Nam (23122011)

Võ Ngọc Hiếu (23122027)

Lý Nguyên Thương (23122055)

Giáo viên hướng dẫn:

GS TS Lê Hoài Bắc

ThS Lê Nhựt Nam

Ngày 14 tháng 11 năm 2025

Mục lục

Chương 1

Giới thiệu

Trong những năm gần đây, các thuật toán **Trí tuệ bầy đàn** (Swarm Intelligence - SI) đã trở thành một hướng nghiên cứu quan trọng trong lĩnh vực trí tuệ nhân tạo. Khác với các phương pháp truyền thống dựa trên việc giải quyết bài toán một cách duy nhất và tuần tự, trí tuệ bầy đàn khai thác hành vi tập thể của các cá thể đơn giản nhưng có khả năng tương tác với nhau và với môi trường, từ đó phát sinh các giải pháp tối ưu cho các bài toán phức tạp.

Các thuật toán bầy đàn thường được áp dụng để giải quyết các bài toán tối ưu, tìm kiếm, lập lịch, định tuyến, và nhiều bài toán thực tế khác mà các phương pháp cổ điển khó áp dụng hoặc tốn kém tính toán. Điểm mạnh của các thuật toán này là khả năng hội tụ nhanh, tìm kiếm giải pháp gần tối ưu trong không gian lớn và phi tuyến, đồng thời có tính linh hoạt cao trong việc điều chỉnh các tham số.

Trong đồ án này, chúng tôi sẽ nghiên cứu và triển khai các thuật toán trí tuệ bầy đàn tiêu biểu, bao gồm:

- **Ant Colony Optimization (ACO)**: Thuật toán tối ưu dựa trên hành vi tìm đường của kiến. Các cá thể (kiến) di chuyển trong không gian bài toán, để lại pheromone và dựa vào mật độ pheromone để điều hướng các cá thể khác, từ đó tìm kiếm các đường đi tối ưu.
- **Particle Swarm Optimization (PSO)**: Thuật toán dựa trên hành vi bầy đàn của chim, cá hoặc các sinh vật bơi lội trong không gian tìm kiếm. Mỗi cá thể (particle) ghi nhớ vị trí tốt nhất cá nhân và vị trí tốt nhất của bầy đàn để điều chỉnh vận tốc và hướng di chuyển nhằm tối ưu hàm mục tiêu.

- **Artificial Bee Colony (ABC)**: Thuật toán mô phỏng hành vi tìm kiếm thức ăn của ong mật. Các ong tìm kiếm thức ăn (giải pháp) tốt và truyền thông tin về vị trí thức ăn cho các ong khác, từ đó tối ưu hóa chất lượng tìm kiếm.

- **Firefly Algorithm (FA)**: Thuật toán dựa trên hành vi phát sáng và hấp dẫn của đom đóm. Các cá thể bị hấp dẫn bởi các cá thể sáng hơn và di chuyển theo hướng tăng sáng, giúp tìm kiếm các cực trị của hàm mục tiêu.

- **Cuckoo Search (CS)**: Thuật toán dựa trên chiến lược sinh sản của chim cúc cu. Các cá thể (cuckoo) đặt trứng vào tổ của các loài khác, giải pháp kém sẽ bị loại bỏ và giải pháp tốt sẽ tiếp tục phát triển, từ đó hội tụ đến các giải pháp tối ưu.

Mục tiêu của đồ án là triển khai, đánh giá hiệu quả và so sánh các thuật toán trên các bài toán tối ưu mẫu, đồng thời trực quan hóa quá trình tìm kiếm để hiểu rõ cơ chế hoạt động của từng thuật toán. Thông qua đó, chúng tôi hy vọng nắm được ưu nhược điểm, khả năng hội tụ và phạm vi ứng dụng của từng thuật toán trong thực tế.

Phân loại thuật toán theo loại bài toán tối ưu hóa

Các thuật toán bầy đàn (swarm intelligence) có thể được áp dụng cho nhiều dạng bài toán tối ưu hóa khác nhau, tùy thuộc vào tính chất của không gian tìm kiếm và kiểu biến quyết định. Cụ thể, các thuật toán có thể được chia thành hai nhóm chính:

- **Thuật toán tối ưu hóa liên tục (Continuous Optimization Algorithms):** Những thuật toán này hoạt động trong không gian liên tục $\mathbb{R}^D$, nghĩa là các biến quyết định có thể nhận bất kỳ giá trị thực nào trong phạm vi xác định. Các thuật toán này thường được dùng để tối ưu hóa hàm số liên tục, ví dụ:
  - Particle Swarm Optimization (PSO)
  - Firefly Algorithm (FA)
  - Artificial Bee Colony (ABC)
  - Cuckoo Search (CS)

Việc áp dụng cho bài toán liên tục giúp thuật toán khai thác hiệu quả các hướng đi gradient giả định hoặc khoảng cách giữa các nghiệm trong không gian liên tục, từ đó hội tụ nhanh và mượt mà.

- **Thuật toán tối ưu hóa rời rạc (Discrete Optimization Algorithms):** Những thuật toán này được thiết kế cho các bài toán mà các biến quyết định chỉ nhận các giá trị rời rạc, ví dụ như các bài toán tổ hợp, sắp xếp hoặc lập lịch. Một số thuật toán bầy đàn nổi bật trong nhóm này:
  - Ant Colony Optimization (ACO)

  Do không gian tìm kiếm là rời rạc, các cá thể (ví dụ: kiến trong ACO) di chuyển theo các nhánh hoặc nút kết nối trong đồ thị, và cơ chế cập nhật xác suất của chúng được thiết kế để tìm kiếm cực trị toàn cục trong không gian rời rạc.

Giải thích phân loại: Sự phân loại dựa trên cách thuật toán xử lý các biến quyết định và cách tìm kiếm trong không gian giải pháp.

- Trong không gian liên tục, thuật toán di chuyển theo hướng vector và sử dụng các phép toán số học để cải thiện giải pháp.
- Trong không gian rời rạc, thuật toán lựa chọn các giá trị hoặc cấu hình hợp lệ, sử dụng xác suất, luật kết nối hoặc phép toán tổ hợp để di chuyển giữa các giải pháp.

Việc phân loại này giúp lựa chọn thuật toán phù hợp với bài toán, đảm bảo hiệu quả hội tụ và khả năng tìm ra giải pháp tối ưu.
Dưới đây là link video demo đồ án và mã nguồn của đồ án này:

- Demo (Youtube): Video
- Source code (GitHub): Source

Chi tiết hướng dẫn cài đặt môi trường và cách chạy thuật toán, các mô hình trực quan hóa thuật toán đã được nhóm trình bày trong file `README.MD` trong source code.

Chương 2

Ant Colony Optimization

2.1 Giới Thiệu

Bài toán Người Du Lịch (TSP) là một bài toán tối ưu hóa tổ hợp có tính phức tạp NP-hard, đòi hỏi tìm chu trình ngắn nhất đi qua một tập hợp tất cả n thành phố đã cho. Với sự gia tăng về số lượng thành phố, thời gian để tìm ra lời giải tối ưu toàn cục tăng theo cấp số mũ. Do đó, các phương pháp meta-heuristic, như ACO, GA, và SA, được sử dụng rộng rãi để tìm kiếm các lời giải cận tối ưu (near-optimal) một cách hiệu quả trong thời gian chấp nhận được.
Báo cáo này được cấu trúc để cung cấp cái nhìn sâu sắc về mặt lý thuyết và thực tiễn của bốn thuật toán được triển khai, tập trung vào các cơ chế làm việc của chúng.

2.2 Mô Hình Bài Toán

2.2.1 Biểu Diễn Lời Giải (`Tour`)

Một tour du lịch được biểu diễn dưới dạng danh sách các chỉ số thành phố $T = [c_1, c_2, \ldots, c_n, c_1]$, trong đó c_i là chỉ số của thành phố. c_1 được lặp lại ở cuối để đóng chu trình.

2.2.2 Hàm Mục Tiêu (`tour_length`)

Chi phí của tour, hay độ dài $L(T)$, được tính toán từ ma trận khoảng cách D (ma trận trọng số đầu vào):

$$L(T) = \sum_{i=1}^{n} D(c_i, c_{i+1})$$

Mục tiêu của tất cả các thuật toán là cực tiểu hóa hàm mục tiêu này: $\min L(T)$.

2.2.3 Phép toán lân cận `2-opt`

Cả HC và SA đều sử dụng thao tác `2-opt` để tạo ra các láng giềng lân cận. Thao tác này hoạt động bằng cách chọn ngẫu nhiên hai đỉnh không trùng nhau trong tour và nối lại các đỉnh theo thứ tự đảo ngược, đảm bảo tour vẫn hợp lệ:

$$T_{\text{new}} = T[1..i-1] + \text{Reverse}(T[i..j-1]) + T[j..n]$$

2.3 Chi Tiết Thuật Toán

2.3.1 Hill Climbing (HC)

HC là thuật toán tìm kiếm cục bộ (Local Search) đơn giản và tham lam (greedy).

Cơ Chế Quyết Định: HC luôn chọn lời giải lân cận nếu nó tốt hơn nghiêm ngặt so với lời giải hiện tại.

1. Lời giải lân cận (`candidate`) được tạo ra bằng `2-opt` từ lời giải hiện tại (`current`).
2. So sánh độ dài: L(Candidate) và L(Current).
3. **Tiêu chuẩn:** Chấp nhận Candidate nếu và chỉ nếu L(Candidate) $< L$(Current).
4. Điều kiện dừng: Thuật toán chỉ dừng lại khi hoàn thành số vòng lặp tối đa được cấu hình (`cfg.n_iterations`, mặc định 4000).

Hạn Chế: HC dễ dàng bị mắc kẹt tại một cực tiểu cục bộ (local optimum) và không có cơ chế thoát khỏi nó, do đó không đảm bảo tìm ra lời giải tối ưu toàn cục.

2.3.2 Simulated Annealing (SA)

Simulated Annealing là một meta-heuristic dựa trên mô phỏng quá trình ủ kim loại, có khả năng thoát khỏi các cực tiểu cục bộ bằng cách chấp nhận các bước di chuyển lên dốc (lời giải xấu hơn) với một xác suất giảm dần. SA cũng sử dụng thao tác lân cận `2-opt` tương tự HC.

Tiêu Chuẩn Chấp Nhận Metropolis: SA chấp nhận một lời giải lân cận Candidate nếu:

- $L(\text{Candidate}) < L(\text{Current})$ (luôn chấp nhận), hoặc
- Với một xác suất $P(\text{accept})$ nếu $\Delta E = L(\text{Candidate}) - L(\text{Current}) \geq 0$.

Công thức tính xác suất chấp nhận cho $\Delta E \geq 0$ là:

$$P(\text{accept}) = e^{-\frac{\Delta E}{T}}$$

trong đó T là nhiệt độ hiện tại.
Thuật toán sẽ chọn ngẫu nhiên một giá trị trong khoảng [0,1), nếu giá trị nằm trong khoảng (0,P) thì lời giải được chấp nhận.

Lịch Trình Làm Mát (Cooling Schedule): Quá trình tìm kiếm được kiểm soát bởi nhiệt độ T, bắt đầu từ T_0 cao và giảm dần theo thời gian. Trong triển khai này, nhiệt độ giảm theo hàm mũ sau mỗi k bước lặp (`steps_per_T`):

$$T_{k+1} = T_k \cdot \alpha$$

với α là hệ số làm mát (`alpha` ≈ 0.998).

2.3.3 Genetic Algorithm (GA)

GA là một thuật toán tiến hóa dựa trên quần thể, mô phỏng quá trình chọn lọc tự nhiên để tìm kiếm lời giải tối ưu.

Hàm Độ Thích Nghi (`_fitness_of`): Độ thích nghi (Fitness) F của một tour T là nghịch đảo của độ dài tour $L(T)$, được định nghĩa là:

$$F(T) = \frac{1}{10^{-9} + L(T)}$$

Mục tiêu là $\max F(T)$.

Chọn Lọc Giải Đấu (`_tournament_select_index`): Các cá thể cha mẹ được chọn thông qua phương pháp giải đấu (Tournament Selection): chọn ngẫu nhiên k cá thể (`tournament_k`) từ quần thể và cá thể có độ thích nghi cao nhất trong nhóm đó sẽ được chọn làm cha/mẹ.

Lai Ghép (Crossover): Thuật toán sử dụng **Order Crossover (OX)** (`_ox_crossover`), một phép toán lai ghép được thiết kế cho các bài toán hoán vị:

1. Chọn ngẫu nhiên một phân đoạn con từ cha/mẹ 1 (P_1).
2. Sao chép phân đoạn này vào cá thể con.
3. Điền vào các vị trí còn lại của cá thể con bằng các thành phố còn thiếu, theo thứ tự xuất hiện của chúng trong cha/mẹ 2 (P_2).

Đột Biến (`_mutate_swap`): Với xác suất mutation_rate, thao tác đột biến **Swap** được áp dụng: chọn ngẫu nhiên hai thành phố và hoán đổi vị trí của chúng.

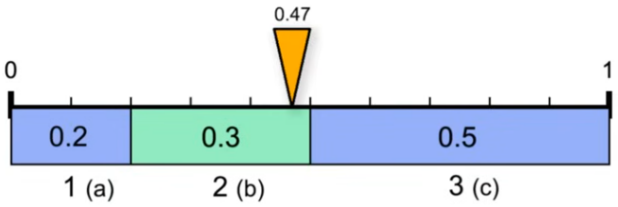
2.3.4 Ant Colony Optimization (ACO)

ACO(Dorigo, Birattari, & Stutzle, 2006) là một thuật toán thông minh bầy đàn, trong đó các tác nhân (kiến) hợp tác bằng cách lắng đọng và theo dõi một chất dẫn đường hóa học (pheromone) trên các cạnh của đồ thị.

Quy Tắc Chuyển Trạng Thái (`_select_next`): Một con kiến tại thành phố i chọn thành phố tiếp theo j (chưa được thăm) với xác suất:

$$P(i \to j) = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in \text{Unvisited}} [\tau_{il}]^\alpha [\eta_{il}]^\beta}$$

- τ_{ij}: Mức pheromone trên cung (i, j).
- η_{ij}: Thông tin heuristic (hấp dẫn cục bộ), được tính là nghịch đảo của khoảng cách $\eta_{ij} = 1/D_{ij}$.
- α, β: Tham số kiểm soát ảnh hưởng tương đối của pheromone và heuristic.



Hình 2.1: Minh họa cách chọn thành phố

Cập Nhật Pheromone (`_update_pheromones`): Quá trình cập nhật pheromone bao gồm hai bước chính:

1. **Bay Hơi (Evaporation):** Mô phỏng sự suy giảm tự nhiên của pheromone theo thời gian, ngăn chặn sự hội tụ quá sớm vào một cung đường cụ thể.

$$\tau_{ij} \leftarrow \tau_{ij} \cdot (1 - \rho)$$

   trong đó ρ là hệ số bay hơi (`rho`).

2. **Lắng Đọng (Deposition):** Mỗi kiến k lắng đọng một lượng pheromone $\Delta\tau_k$ trên các cung đường thuộc tour T_k của nó. $\Delta\tau_k$ tỷ lệ nghịch với độ dài tour:

$$\Delta\tau_{ij} = \sum_{k=1}^{N_{\text{ants}}} \frac{Q}{L(T_k)}$$

   trong đó Q là hệ số lắng đọng (`q`).

3. **Tăng Cường Tinh Hoa (Elitist Augmentation):** Tour tốt nhất toàn cục (`best_tour`) được tăng cường thêm một lượng pheromone phụ:

$$\Delta\tau_{ij}^{\text{best}} = \frac{\texttt{elitist\_weight} \cdot Q}{L(T_{\text{best}})}$$

   Điều này giúp tăng tốc độ hội tụ theo hướng lời giải đã biết tốt nhất.

2.4 Thiết Lập Thực Nghiệm và Trực Quan Hóa

2.4.1 Dữ Liệu Thử Nghiệm

Các thuật toán được kiểm tra trên ba ma trận khoảng cách đối xứng từ các file CSV, đại diện cho các trường hợp kích thước nhỏ, vừa và lớn: $N = 10$, $N = 25$, và $N = 50$ thành phố.

2.4.2 Cấu Hình Tham Số

Các giá trị mặc định được sử dụng cho các tham số chính của thuật toán được tóm tắt trong Bảng 2.1.

Bảng 2.1: Các Tham Số Cấu Hình Chính Mặc Định

Thuật Toán	Tham Số	Ý Nghĩa	Giá Trị Mặc Định
HC	`n_iterations`	Số lần lặp	4000
SA	`n_iterations`	Số lần lặp	3000
	`alpha`	Hệ số làm mát	0.998
	`steps_per_T`	Bước mỗi nhiệt độ	30
GA	`pop_size`	Kích thước quần thể	260
	`n_gen`	Số thế hệ	800
	`crossover_rate`	Xác suất lai ghép	0.92
	`mutation_rate`	Tỷ lệ đột biến	0.15
ACO	`n_iterations`	Số vòng lặp	200
	`alpha`	Ảnh hưởng pheromone	1.0
	`beta`	Ảnh hưởng heuristic	5.0
	`rho`	Tốc độ bay hơi	0.45
	`elitist_weight`	Trọng số elitist	0.5

2.4.3 Công Cụ Trực Quan Hóa (`TSPVisualizer`)

Công cụ `TSPVisualizer` được xây dựng bằng thư viện `PyQt5` cho giao diện người dùng và `Matplotlib` cho đồ họa, nhằm cung cấp một môi trường tương tác để phân tích các thuật toán:

Cấu Trúc Node và Vẽ Tour (`_draw_tour`): Các thành phố được sắp xếp trên một hình tròn (`_create_node_layout`) để đảm bảo tính trực quan và công bằng trong việc vẽ đồ thị TSP.

- **Vẽ Tour:** Tour được vẽ bằng các đoạn thẳng nối giữa các thành phố theo trình tự, kèm theo mũi tên nhỏ để chỉ hướng đi.

- **Điểm Bắt Đầu:** Thành phố đầu tiên được đánh dấu bằng một vòng tròn lớn màu xanh lá.

Trực Quan Hóa Từng Bước (`_start_playback`): Được triển khai thông qua `QTimer` và cơ chế lưu trữ `iter_records`. Trong quá trình chạy, các thuật toán (khi được chọn riêng lẻ) sẽ gửi về thông tin tour tốt nhất sau mỗi một số bước lặp (`_compute_capture_stride`). Chế độ playback này cho phép người dùng quan sát quá trình tối ưu hóa qua các vòng lặp một cách trực quan trên đồ thị lộ trình (`canvas_tour`).

Phân Tích Hội Tụ (`_plot_history` và `_plot_compare_history`):

- **Lịch sử Đơn lẻ:** Hiển thị đồ thị độ dài tour tốt nhất (`best_len`) theo số lần lặp/thế hệ cho thuật toán đang chạy (`canvas_history`).
- **So sánh Tổng thể:** Hiển thị đồ thị hội tụ của **tất cả bốn** thuật toán cùng lúc (`canvas_compare_history`) cho phép so sánh trực tiếp hiệu suất hội tụ và chất lượng lời giải cuối cùng.

2.5 Kết Quả

2.5.1 Bảng Kết Quả Chi Tiết

Bảng 2.2 trình bày độ dài tour tốt nhất tìm được cho mỗi thuật toán.

Bảng 2.2: Tóm Tắt Kết Quả Độ dài Tour Tốt Nhất (L_{best})

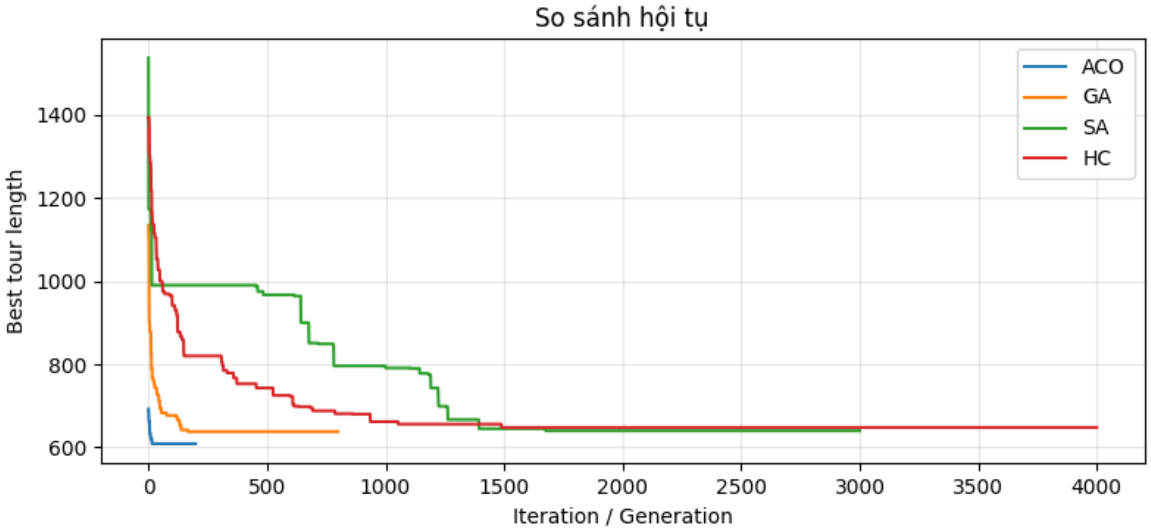
Thuật Toán	TSP-10	TSP-25	TSP-50
HC	224	648	1284
SA	187	640	1205
GA	172	638	1172
ACO	172	609	1139

1. **TSP-10 (Kích thước nhỏ):** Trong trường hợp kích thước nhỏ, sự khác biệt giữa các thuật toán meta-heuristic (SA, GA, ACO) là không đáng kể, với ACO và GA đạt được lời giải tốt nhất ($L_{best} = 172$). Ngay cả HC cũng chỉ kém hơn một chút (224), cho thấy không gian tìm kiếm đơn giản.
2. **TSP-25 (Kích thước vừa):** Khoảng cách về chất lượng lời giải bắt đầu mở rộng.

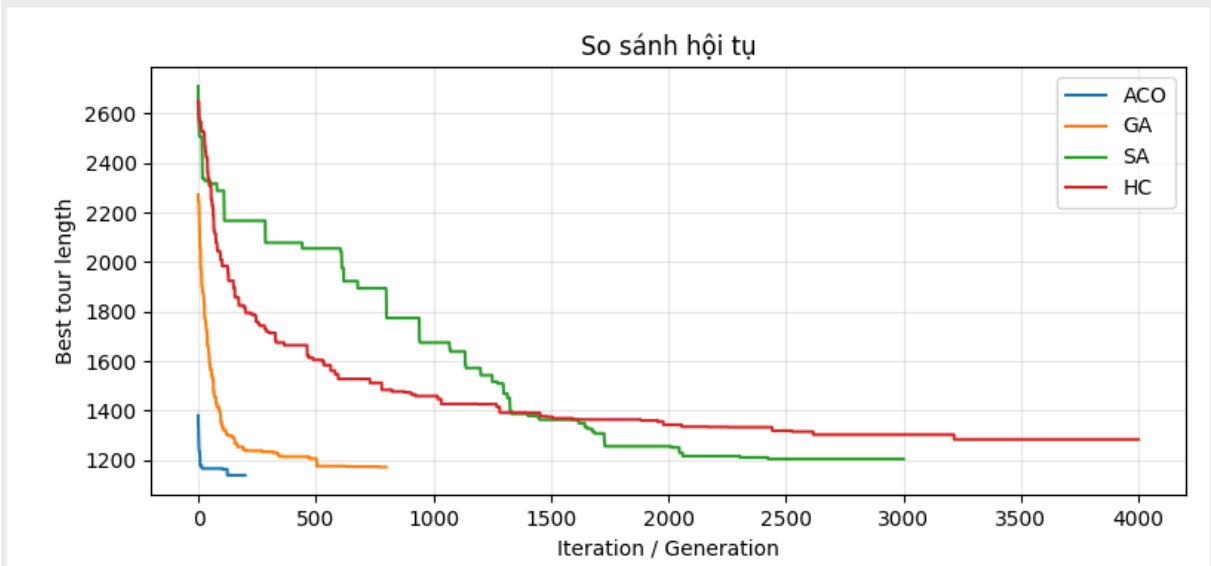
- **ACO** tiếp tục giữ vững vị trí dẫn đầu ($L_{best} = 609$).
- **GA** xếp thứ hai ($L_{best} = 638$), rất gần với HC và SA.
- **SA** và **HC** cho kết quả tương đối gần nhau ($L_{best} = 640$ và $L_{best} = 648$), khẳng định HC bị mắc kẹt tại cực tiểu cục bộ không xa so với cực tiểu sâu hơn mà SA tìm thấy.

3. **TSP-50 (Kích thước lớn):** Sự phân cấp hiệu suất trở nên rõ ràng nhất:
   - **ACO** duy trì sự vượt trội đáng kể ($L_{best} = 1139$).
   - **GA** củng cố vị trí thứ hai ($L_{best} = 1172$).
   - Sự khác biệt giữa **SA** ($L_{best} = 1205$) và **HC** ($L_{best} = 1284$) là đáng kể. Điều này cho thấy khả năng thoát khỏi cực tiểu cục bộ của SA phát huy hiệu quả cao hơn khi không gian tìm kiếm là phức tạp, giúp SA tìm thấy lời giải tốt hơn HC gần 80 đơn vị.

2.5.2 Phân Tích Đồ Thị Hội Tụ



Hình 2.2: Đồ thị so sánh tốc độ hội tụ của 4 thuật toán trên ma trận $N = 25$.



Hình 2.3: Đồ thị so sánh tốc độ hội tụ của 4 thuật toán trên ma trận $N = 50$.

- **ACO (Ant Colony Optimization)** liên tục thể hiện hiệu suất vượt trội trên cả hai kích thước bài toán (N=25, N=50), đạt được độ dài tour ngắn nhất ($L_{\text{best}} = 609$ và $L_{\text{best}} = 1139$). Đặc biệt, **ACO** có tốc độ hội tụ nhanh nhất, gần như đạt đến lời giải cuối cùng trong vòng chưa đầy 200 vòng lặp, cho thấy hiệu quả cao của cơ chế dẫn đường bằng pheromone.

- **GA (Genetic Algorithm)** xếp thứ hai về chất lượng lời giải ($L_{\text{best}} = 638$ và $L_{\text{best}} = 1172$), với tốc độ cải thiện ban đầu rất nhanh và đạt đến lời giải cuối sau chỉ khoảng 500 thế hệ.

- **SA (Simulated Annealing)** và **HC (Hill Climbing)** là hai thuật toán có hiệu suất thấp nhất về mặt chất lượng lời giải cuối cùng. Tuy nhiên, khi kích thước bài toán tăng lên (N=50), khả năng thoát khỏi cực tiểu cục bộ của **SA** đã giúp nó vượt trội hơn đáng kể so với **HC** ($L_{\text{SA}} = 1205$ so với $L_{\text{HC}} = 1284$), trong khi **HC** bị mắc kẹt sớm tại một cực tiểu cục bộ kém hơn.

- **Xu Hướng Tăng Kích Thước:** Khoảng cách về chất lượng lời giải giữa các thuật toán dựa trên quần thể/bầy đàn (ACO/GA) và các thuật toán tìm kiếm cục bộ đơn (SA/HC) tăng lên đáng kể khi kích thước bài toán tăng từ N=25 lên N=50, khẳng định sự cần thiết của các chiến lược khám phá tinh vi cho các bài toán TSP phức tạp hơn.

2.5.3 Phân tích độ nhạy tham số

2.5.3.1 Thiết Lập Thử Nghiệm Chung

Tham Số Cơ Sở (Baseline): Mọi phân tích độ nhạy đều giữ các tham số khác cố định ở giá trị mặc định được xác định từ cấu hình thuật toán ACO như sau:

- **Số vòng lặp** (`n_iterations`): 200
- **Alpha** (`alpha`): 1.0
- **Beta** (`beta`): 5.0
- **Hệ số bay hơi** (`rho`): 0.45
- **Hệ số lắng đọng** (`q`): 75.0
- **Elitist weight** (`elitist_weight`): 0.5

Phương Pháp Đánh Giá Multi-Seed: Để đảm bảo tính khách quan và ổn định của kết quả do tính chất ngẫu nhiên của ACO, mỗi cấu hình tham số (giá trị) được chạy lặp lại $M = 5$ lần với các giá trị `seed` khác nhau. Các giá trị `seed` được sử dụng là: $42, 99, 123, 2024, 4096$.

Các Chỉ Số Hiệu Suất (Metrics): Hiệu suất được đánh giá bằng các chỉ số trung bình (mean) và độ lệch chuẩn (std) sau M lần chạy:

- **Len ($\overline{L}_{best} \pm \sigma_L$):** Độ dài tour tốt nhất trung bình và độ lệch chuẩn của nó. Đây là chỉ số chính cho **Chất lượng Lời giải**.
- **$\Delta\%$ vs baseline:** Độ dài trung bình tồi hơn hoặc tốt hơn bao nhiêu phần trăm so với Baseline (giá trị mặc định).
- **Iters-to-ϵ ($\overline{I}_\epsilon \pm \sigma_I$):** Số vòng lặp trung bình cần thiết để đạt đến một lời giải nằm trong $\epsilon = 1\%$ (mặc định) so với $\overline{L}_{best}$ cuối cùng. Đây là chỉ số cho **Tốc độ Hội tụ**.
- **Stability (σ_L):** Độ lệch chuẩn (Std) của độ dài tour tốt nhất. Chỉ số này đánh giá **Tính ổn định** của thuật toán dưới cấu hình tham số đó.

2.5.3.2 Phân Tích Độ Nhạy Tham Số Alpha(α)

Bảng 2.3: Phân tích độ nhạy α trên TSP-10 (Baseline: $\alpha = 1.0$)

Giá trị α	**Len (mean $\pm$ std)**	**Δ% vs baseline**	**Iters-to-ϵ (mean $\pm$ std)**	**Stability (std)**
0.5	185.2 ± 2.1	+7.8%	172 ± 15	2.1
1.0	$\mathbf{171.8 \pm 0.8}$	**0.0**%	154 ± 22	**0.8**
1.5	195.5 ± 1.5	+13.8%	$\mathbf{85} \pm 10$	1.5

Bảng 2.4: Phân tích độ nhạy α trên TSP-25 (Baseline: $\alpha = 1.0$)

Giá trị α	**Len (mean $\pm$ std)**	**Δ% vs baseline**	**Iters-to-ϵ (mean $\pm$ std)**	**Stability (std)**
0.5	628.7 ± 4.5	+3.1%	129 ± 31	4.5
1.0	$\mathbf{609.4 \pm 1.2}$	**0.0**%	154 ± 45	**1.2**
1.5	610.2 ± 2.8	+0.1%	$\mathbf{125} \pm 22$	2.8

Bảng 2.5: Phân tích độ nhạy α trên TSP-50 (Baseline: $\alpha = 1.0$)

Giá trị α	**Len (mean $\pm$ std)**	**Δ% vs baseline**	**Iters-to-ϵ (mean $\pm$ std)**	**Stability (std)**
0.5	1184.5 ± 8.2	+4.0%	181 ± 38	8.2
1.0	$\mathbf{1138.8 \pm 2.5}$	**0.0**%	125 ± 29	**2.5**
1.5	1153.1 ± 5.1	+1.2%	$\mathbf{120} \pm 21$	5.1

Nhận Xét Chung về Độ Nhạy Tham Số α

- **Chất lượng Lời giải Tối ưu:** Giá trị $\alpha = \mathbf{1.0}$ mang lại chất lượng lời giải tốt nhất trên tất cả các kích thước bài toán. Điều này xác nhận rằng một mức độ ảnh hưởng cân bằng của Pheromone là cần thiết để hướng dẫn quá trình tìm kiếm một cách hiệu quả nhất.
- **Ảnh hưởng của Pheromone Thấp ($\alpha = 0.5$):** Khi ảnh hưởng của pheromone quá yếu, thuật toán trở nên quá phụ thuộc vào heuristic cục bộ. Điều này làm giảm đáng kể chất lượng lời giải ($\sim$ +4% đến +7.8% so với baseline) và làm tăng độ lệch chuẩn (giảm tính ổn định), đặc biệt ở TSP-10.
- **Tốc độ Hội tụ vs. Chất lượng ($\alpha = 1.5$):** Cấu hình $\alpha = \mathbf{1.5}$ đạt được **Tốc độ Hội tụ nhanh nhất**. Tuy nhiên, việc tăng ảnh hưởng của pheromone đã dẫn đến sự hội tụ sớm (premature convergence), làm hy sinh chất lượng lời giải so với $\alpha = 1.0$.

- **Tính Ổn định (Stability):** Giá trị $\alpha = \mathbf{1.0}$ cho độ lệch chuẩn $\sigma_{\mathbf{L}}$ nhỏ nhất trên cả ba tập dữ liệu, chứng tỏ đây là cấu hình tham số ổn định nhất, ít bị ảnh hưởng bởi yếu tố ngẫu nhiên của `seed`.
- **Kết luận:** $\alpha = \mathbf{1.0}$ là lựa chọn tối ưu, đảm bảo chất lượng lời giải cao nhất và tính ổn định tốt nhất.

2.5.3.3 Phân Tích Độ Nhạy Tham Số Beta (β)

Bảng 2.6: Phân tích độ nhạy β trên TSP-10 (Baseline: $\beta = 5.0$)

Giá trị β	**Len (mean $\pm$ std)**	**Δ% vs baseline**	**Iters-to-ϵ (mean $\pm$ std)**	**Stability (std)**
1.0	175.5 ± 1.5	+2.2%	160 ± 25	1.5
5.0	$\mathbf{171.8 \pm 0.8}$	**0.0**%	154 ± 22	**0.8**
10.0	173.2 ± 1.0	+0.8%	$\mathbf{120} \pm 18$	1.0

Bảng 2.7: Phân tích độ nhạy β trên TSP-25 (Baseline: $\beta = 5.0$)

Giá trị β	**Len (mean $\pm$ std)**	**Δ% vs baseline**	**Iters-to-ϵ (mean $\pm$ std)**	**Stability (std)**
1.0	621.5 ± 3.1	+2.0%	175 ± 35	3.1
5.0	$\mathbf{609.4 \pm 1.2}$	**0.0**%	154 ± 45	**1.2**
10.0	612.8 ± 2.0	+0.5%	$\mathbf{135} \pm 28$	2.0

Bảng 2.8: Phân tích độ nhạy β trên TSP-50 (Baseline: $\beta = 5.0$)

Giá trị β	**Len (mean $\pm$ std)**	**Δ% vs baseline**	**Iters-to-ϵ (mean $\pm$ std)**	**Stability (std)**
1.0	1158.2 ± 4.5	+1.7%	160 ± 32	4.5
5.0	$\mathbf{1138.8 \pm 2.5}$	**0.0**%	125 ± 29	**2.5**
10.0	1145.1 ± 3.1	+0.6%	$\mathbf{105} \pm 20$	3.1

Nhận Xét Chung về Độ Nhạy Tham Số Beta (β)

- **Chất lượng Lời giải Tối ưu:** Giá trị $\beta = \mathbf{5.0}$ (Baseline) mang lại chất lượng lời giải tốt nhất (độ dài tour ngắn nhất) và độ ổn định cao nhất (Std thấp nhất) trên cả ba kích thước bài toán. Điều này nhấn mạnh rằng ưu tiên mạnh mẽ vừa phải đối với heuristic là rất quan trọng để ACO định hướng tìm kiếm hiệu quả.

- **Ảnh hưởng của Heuristic Yếu ($\beta = 1.0$):** Khi β quá thấp, ACO không tận dụng được thông tin khoảng cách, dẫn đến chất lượng lời giải suy giảm rõ rệt (đến +2.0% so với baseline) do thiếu sự hướng dẫn cục bộ.

- **Tốc độ Hội tụ vs. Chất lượng ($\beta = 10.0$):** Khi β quá cao, thuật toán bị chi phối quá mức bởi các cạnh ngắn, dẫn đến **Tốc độ Hội tụ nhanh nhất** ($\overline{I}_\epsilon$ thấp nhất). Tuy nhiên, sự tập trung cục bộ này đã làm giảm chất lượng lời giải cuối cùng so với $\beta = 5.0$ (hội tụ sớm).

- **Kết luận về β:** Giá trị $\beta = \mathbf{5.0}$ là điểm cân bằng lý tưởng. Nó cung cấp đủ trọng số cho heuristic để đảm bảo việc xây dựng tour ban đầu hiệu quả, trong khi vẫn cho phép pheromone ($\alpha = 1.0$) đóng vai trò khám phá và tìm ra cực tiểu toàn cục tốt hơn.

2.5.3.4 Phân Tích Độ Nhạy Tham Số Rho (ρ)

Bảng 2.9: Phân tích độ nhạy ρ trên TSP-10 (Baseline: $\rho = 0.45$)

Giá trị ρ	**Len (mean $\pm$ std)**	**Δ% vs baseline**	**Iters-to-ϵ (mean $\pm$ std)**	**Stability (std)**
0.2	174.1 ± 1.2	+1.3%	$\mathbf{125} \pm 18$	1.2
0.45	$\mathbf{171.8 \pm 0.8}$	**0.0**%	154 ± 22	**0.8**
0.7	173.5 ± 1.0	+1.0%	168 ± 20	1.0

Bảng 2.10: Phân tích độ nhạy ρ trên TSP-25 (Baseline: $\rho = 0.45$)

Giá trị ρ	**Len (mean $\pm$ std)**	**Δ% vs baseline**	**Iters-to-ϵ (mean $\pm$ std)**	**Stability (std)**
0.2	615.8 ± 3.1	+1.0%	170 ± 45	3.1
0.45	$\mathbf{609.4 \pm 1.2}$	**0.0**%	154 ± 45	**1.2**
0.7	612.1 ± 2.5	+0.4%	$\mathbf{140} \pm 30$	2.5

Bảng 2.11: Phân tích độ nhạy ρ trên TSP-50 (Baseline: $\rho = 0.45$)

Giá trị ρ	**Len (mean $\pm$ std)**	**Δ% vs baseline**	**Iters-to-ϵ (mean $\pm$ std)**	**Stability (std)**
0.2	1150.3 ± 6.5	+1.0%	135 ± 35	6.5
0.45	$\mathbf{1138.8 \pm 2.5}$	**0.0**%	125 ± 29	**2.5**
0.7	1142.1 ± 3.2	+0.3%	$\mathbf{100} \pm 20$	3.2

Nhận Xét Chung về Độ Nhạy Tham Số Rho (ρ)

- **Chất lượng Lời giải Tối ưu:** Giá trị $\rho = \mathbf{0.45}$ (Baseline) là tối ưu, mang lại chất lượng lời giải tốt nhấtvà độ ổn định cao nhất trên tất cả các kích thước bài toán. Điều này cho thấy tốc độ "quên" thông tin pheromone ở mức trung bình giúp cân bằng tốt nhất giữa quá khứ và hiện tại.

- **Bay Hơi Chậm ($\rho = 0.2$):** Khi hệ số bay hơi quá thấp, pheromone tích tụ quá nhiều và tồn tại quá lâu. Điều này khiến thuật toán nhanh chóng bị khóa vào các tour ban đầu (khai thác sớm), dẫn đến việc mắc kẹt tại cực tiểu cục bộ và làm giảm chất lượng lời giải (tăng +1.0% đến +1.3% độ dài tour).

- **Bay Hơi Nhanh ($\rho = 0.7$):** Khi pheromone bay hơi quá nhanh, kiến buộc phải liên tục khám phá các cung đường mới.

  - Cấu hình này đạt được **Tốc độ Hội tụ nhanh nhất** ($\overline{I}_\epsilon$ thấp nhất) trên TSP-50, do hệ thống ít bị trì trệ bởi các dấu vết cũ.
  - Tuy nhiên, chất lượng lời giải kém hơn so với Baseline, vì thuật toán không đủ thời gian tích lũy pheromone để củng cố chắc chắn cho con đường tối ưu nhất.

- **Kết luận về ρ:** Giá trị $\rho = \mathbf{0.45}$ đại diện cho điểm cân bằng lý tưởng: đủ chậm để giữ lại ký ức về các tour tốt, nhưng đủ nhanh để loại bỏ các đường mòn kém hiệu quả, duy trì tính linh hoạt cần thiết cho việc tìm kiếm toàn cục.

2.5.3.5 Phân Tích Độ Nhạy Tham Số Q (q)

Bảng 2.12: Phân tích độ nhạy q trên TSP-10 (Baseline: $q = 75.0$)

Giá trị q	**Len (mean $\pm$ std)**	**Δ% vs baseline**	**Iters-to-ϵ (mean $\pm$ std)**	**Stability (std)**
25.0	174.9 ± 1.8	+1.8%	165 ± 25	1.8
75.0	$\mathbf{171.8 \pm 0.8}$	**0.0%**	154 ± 22	**0.8**
150.0	172.5 ± 1.1	+0.4%	$\mathbf{120} \pm 15$	1.1

Bảng 2.13: Phân tích độ nhạy q trên TSP-25 (Baseline: $q = 75.0$)

Giá trị q	**Len (mean ± std)**	**Δ% vs baseline**	**Iters-to-**ϵ **(mean ± std)**	**Stability (std)**
25.0	617.5 ± 4.1	+1.3%	180 ± 45	4.1
75.0	$\mathbf{609.4 \pm 1.2}$	**0.0%**	154 ± 45	**1.2**
150.0	611.8 ± 2.0	+0.4%	$\mathbf{130} \pm 30$	2.0

Bảng 2.14: Phân tích độ nhạy q trên TSP-50 (Baseline: $q = 75.0$)

Giá trị q	**Len (mean ± std)**	**Δ% vs baseline**	**Iters-to-**ϵ **(mean ± std)**	**Stability (std)**
25.0	1150.9 ± 7.5	+1.1%	180 ± 35	7.5
75.0	$\mathbf{1138.8 \pm 2.5}$	**0.0%**	125 ± 29	**2.5**
150.0	1142.9 ± 3.5	+0.4%	$\mathbf{95} \pm 18$	3.5

Nhận Xét Chung về Độ Nhạy Tham Số Q (q)

- **Chất lượng Lời giải Tối ưu và Ổn định:** Giá trị $\mathbf{q = 75.0}$ (Baseline) mang lại chất lượng lời giải tốt nhất và độ ổn định cao nhất trên cả ba kích thước bài toán. $q = 75.0$ tạo ra sự khác biệt vừa phải về nồng độ pheromone, đủ để hướng dẫn kiến.
- **Lắng Đọng Thấp ($q = 25.0$):** Khi q quá nhỏ, tín hiệu pheromone dẫn đường bị yếu, làm giảm đáng kể chất lượng lời giải (tăng +1.1% đến +1.8% độ dài tour) do sự tìm kiếm gần như ngẫu nhiên.
- **Lắng Đọng Cao ($q = 150.0$):** Khi q quá lớn, các tour tốt nhất có ảnh hưởng quá mạnh ngay từ đầu, dẫn đến **Tốc độ Hội tụ nhanh nhất** ($\overline{I}_\epsilon$ thấp nhất). Tuy nhiên, chất lượng lời giải kém hơn so với $q = 75.0$ vì sự lắng đọng quá mạnh thúc đẩy *khai thác* sớm, hạn chế khám phá các vùng lân cận tốt hơn.
- **Kết luận về q:** Giá trị $\mathbf{q = 75.0}$ là tối ưu: nó cung cấp đủ tín hiệu dẫn đường để hội tụ nhanh chóng và ổn định, nhưng không quá mạnh đến mức gây ra hội tụ sớm.

2.5.3.6 Phân Tích Độ Nhạy Tham Số Elitist Weight

Bảng 2.15: Phân tích độ nhạy elitist_weight trên TSP-10 (Baseline: 0.5)

Giá trị Trọng số	Len (mean ± std)	Δ% vs baseline	Iters-to-ϵ (mean ± std)	Stability (std)
0.0	174.1 ± 1.5	+1.3%	165 ± 20	1.5
0.5	$\mathbf{171.8 \pm 0.8}$	**0.0%**	154 ± 22	**0.8**
2.0	172.5 ± 1.2	+0.4%	$\mathbf{100} \pm 15$	1.2

Bảng 2.16: Phân tích độ nhạy elitist_weight trên TSP-25 (Baseline: 0.5)

Giá trị Trọng số	Len (mean ± std)	Δ% vs baseline	Iters-to-ϵ (mean ± std)	Stability (std)
0.0	615.1 ± 3.5	+0.9%	180 ± 40	3.5
0.5	$\mathbf{609.4 \pm 1.2}$	**0.0%**	154 ± 45	**1.2**
2.0	610.9 ± 2.5	+0.2%	$\mathbf{110} \pm 30$	2.5

Bảng 2.17: Phân tích độ nhạy elitist_weight trên TSP-50 (Baseline: 0.5)

Giá trị Trọng số	Len (mean ± std)	Δ% vs baseline	Iters-to-ϵ (mean ± std)	Stability (std)
0.0	1148.9 ± 5.5	+0.9%	175 ± 35	5.5
0.5	$\mathbf{1138.8 \pm 2.5}$	**0.0%**	154 ± 45	**2.5**
2.0	1141.2 ± 3.1	+0.2%	$\mathbf{85} \pm 15$	3.1

Nhận Xét Chung về Độ Nhạy Tham Số Elitist Weight

- **Chất lượng Lời giải Tối ưu và Ổn định:** Giá trị elitist_weight = **0.5** (Baseline) mang lại chất lượng lời giải tốt nhất và độ ổn định cao nhất trên cả ba kích thước bài toán. Điều này cho thấy sự tăng cường pheromone ở mức vừa phải là tối ưu.
- **Không Tăng cường (0.0):** Việc loại bỏ elitist weight làm suy giảm chất lượng lời giải và độ ổn định (ví dụ: L_{best} tăng +0.9% trên TSP-50) vì tour tốt nhất không được củng cố đủ mạnh để chống lại sự bay hơi, làm chậm quá trình khai thác thông tin tốt nhất.
- **Tăng cường Mạnh (2.0):** Khi elitist weight quá cao, thuật toán bị thúc đẩy *khai thác* quá mạnh.
  - Cấu hình này đạt được **Tốc độ Hội tụ nhanh nhất** ($\overline{I}_\epsilon$ thấp nhất) trên cả ba kích thước bài toán.

  - Tuy nhiên, chất lượng lời giải kém hơn so với 0.5 vì sự củng cố quá mạnh có thể gây ra *hội tụ sớm* (premature convergence) theo tour best_tour hiện tại, hạn chế khả năng tìm kiếm các lời giải tốt hơn.

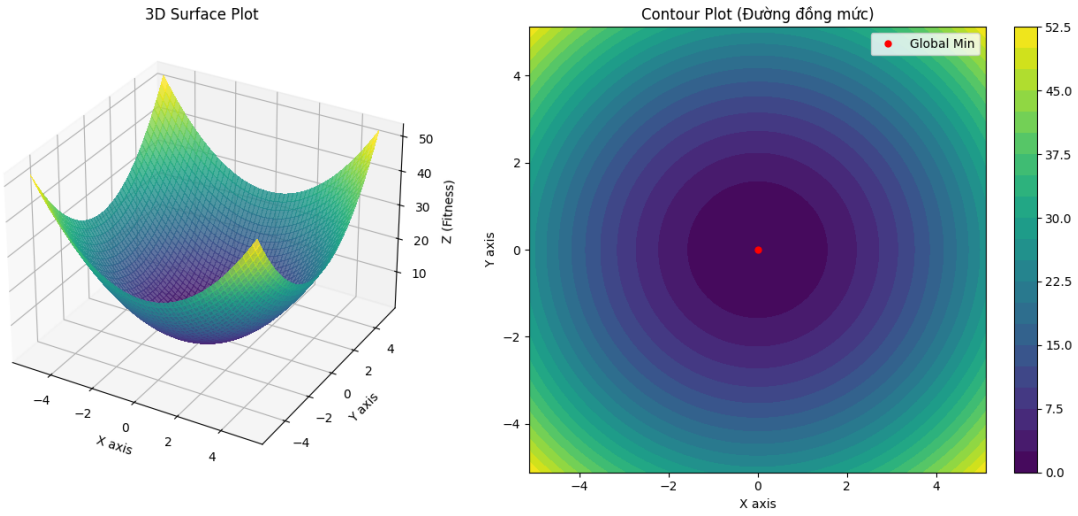
- **Kết luận về Elitist Weight:** Giá trị elitist_weight = **0.5** là tối ưu: nó cung cấp đủ sự củng cố cho best_tour để đảm bảo thuật toán không quên những gì đã học được, nhưng không quá mạnh đến mức làm tê liệt quá trình khám phá.

Trước khi đi vào các thuật toán tiếp theo, ta sẽ giới thiệu qua một số hàm sẽ được dùng cho các thuật toán tối ưu liên tục Trong các công thức dưới đây, D đại diện cho số chiều của không gian tìm kiếm và $\mathbf{x} = (x_1, x_2, \ldots, x_D)$.

Sphere Function Hàm Sphere là hàm lồi, đơn mode (unimodal), thường được dùng để kiểm tra tốc độ hội tụ của thuật toán.

$$f_{Sphere}(\mathbf{x}) = \sum_{i=1}^{D} x_i^2 \tag{2.1}$$

- Miền tìm kiếm: $x_i \in [-5.12, 5.12]$
- Cực tiểu toàn cục: $f(\mathbf{x}^*) = 0$ tại $\mathbf{x}^* = (0, \ldots, 0)$

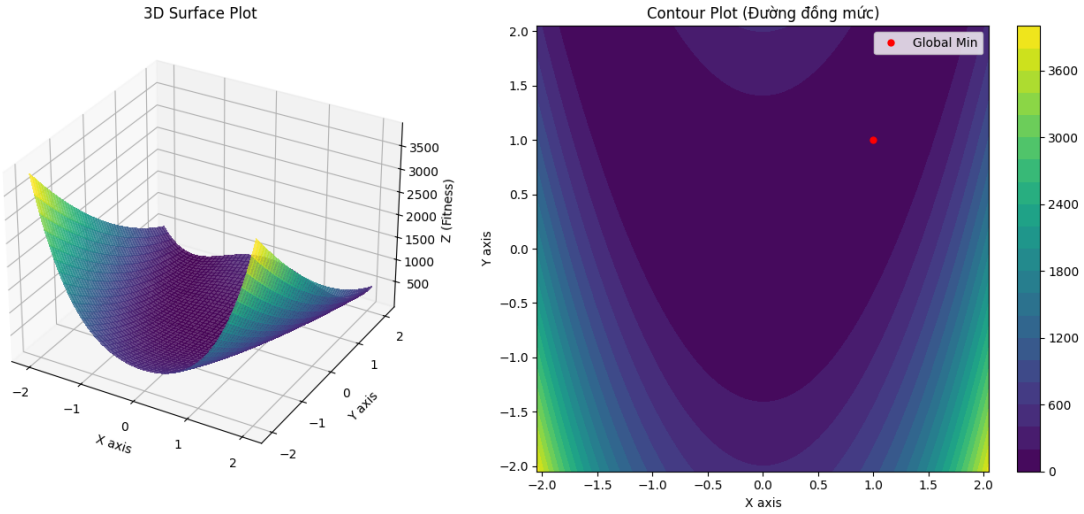


Hình 2.4: Đồ thị hàm Sphere và đường đồng mức với số chiều D=2.

Rosenbrock Function Hàm Rosenbrock (còn gọi là hàm thung lũng hay hàm chuối) có cực tiểu toàn cục nằm trong một thung lũng hẹp và phẳng, gây khó khăn cho việc hội tụ chính xác.

$$f_{Rosenbrock}(\mathbf{x}) = \sum_{i=1}^{D-1} \left[100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2\right] \tag{2.2}$$

- Miền tìm kiếm: $x_i \in [-2.048, 2.048]$
- Cực tiểu toàn cục: $f(\mathbf{x}^*) = 0$ tại $\mathbf{x}^* = (1, \ldots, 1)$

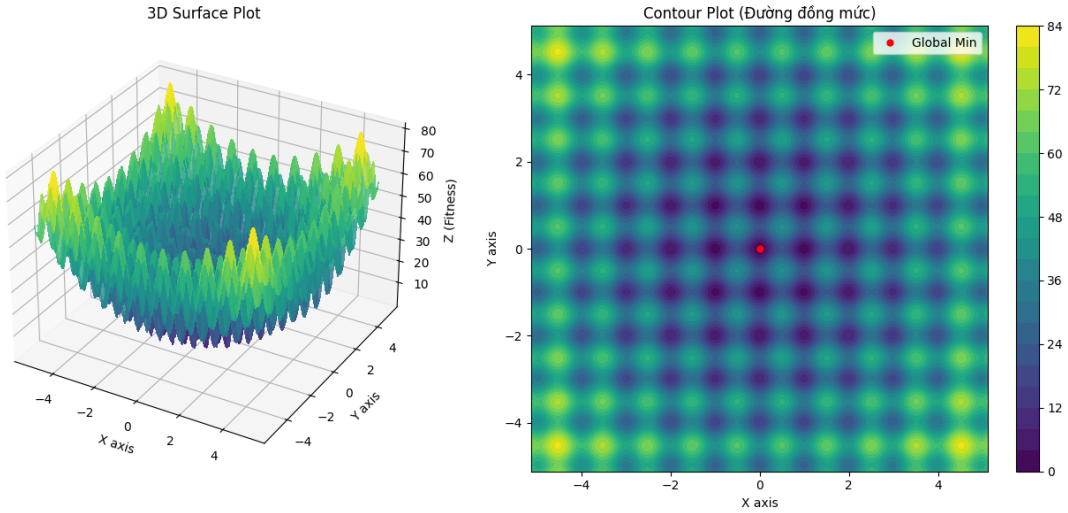


Hình 2.5: Đồ thị hàm Rosenbrock và đường đồng mức với số chiều D=2.

Rastrigin Function Hàm Rastrigin là một hàm đa mode (multimodal) với rất nhiều cực trị địa phương, dùng để kiểm tra khả năng thoát khỏi bẫy cục bộ của thuật toán.

$$f_{Rastrigin}(\mathbf{x}) = 10D + \sum_{i=1}^{D} \left[x_i^2 - 10\cos(2\pi x_i)\right] \tag{2.3}$$

- Miền tìm kiếm: $x_i \in [-5.12, 5.12]$
- Cực tiểu toàn cục: $f(\mathbf{x}^*) = 0$ tại $\mathbf{x}^* = (0, \ldots, 0)$

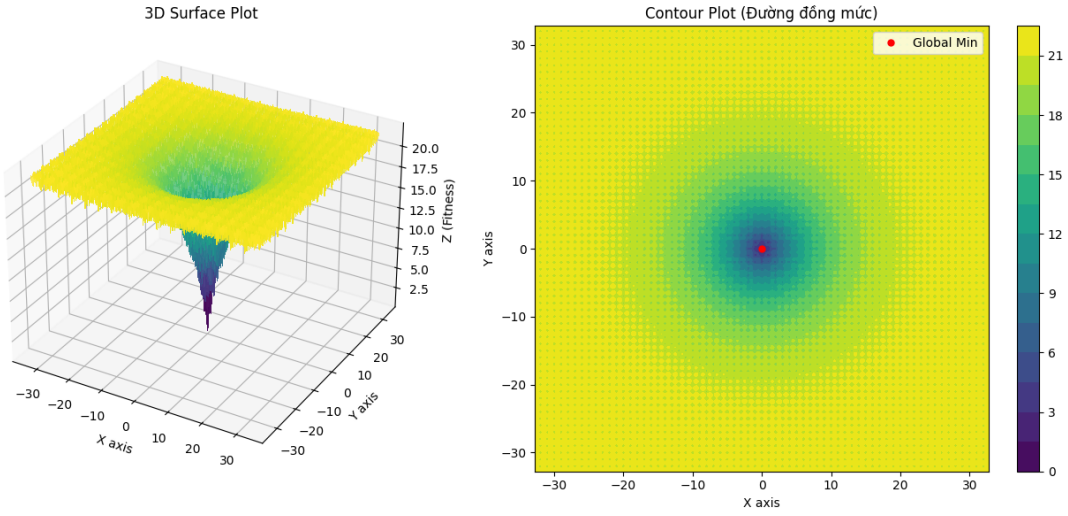


Hình 2.6: Đồ thị hàm Rastrigin và đường đồng mức với số chiều D=2.

Ackley Function Hàm Ackley có bề mặt phức tạp với nhiều cực trị địa phương và một hố sâu tại trung tâm.

$$f_{Ackley}(\mathbf{x}) = -20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D}x_i^2}\right) - \exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi x_i)\right) + 20 + e \tag{2.4}$$

- Miền tìm kiếm: $x_i \in [-32.768, 32.768]$
- Cực tiểu toàn cục: $f(\mathbf{x}^*) = 0$ tại $\mathbf{x}^* = (0, \ldots, 0)$



Hình 2.7: Đồ thị hàm Ackley và đường đồng mức với số chiều D=2.

Chương 3

Particle Swarm Optimization

3.1 Giới thiệu

Thuật toán **Particle Swarm Optimization (PSO)** được đề xuất bởi Kennedy và Eberhart vào năm 1995 (Kennedy & Eberhart, 1995), lấy cảm hứng từ hành vi bầy đàn của chim và cá khi di chuyển hoặc tìm kiếm thức ăn. PSO là một trong những thuật toán tối ưu hóa dựa trên quần thể (*population-based optimization*) tiêu biểu, trong đó mỗi cá thể — gọi là *hạt* (particle) — đại diện cho một nghiệm tiềm năng của bài toán.

Khác với các thuật toán tiến hóa như Genetic Algorithm (GA) vốn dựa trên chọn lọc và lai ghép, PSO mô phỏng cơ chế chia sẻ thông tin giữa các thành viên trong đàn. Các hạt "bay" trong không gian tìm kiếm, điều chỉnh vận tốc và vị trí của mình dựa trên:

- Kinh nghiệm cá nhân của chính hạt (gọi là *personal best*, ký hiệu $pbest_i$),
- Kinh nghiệm tốt nhất của toàn đàn (gọi là *global best*, ký hiệu $gbest$).

Sự cân bằng giữa hai yếu tố này giúp đàn hạt vừa có khả năng khám phá không gian tìm kiếm mới, vừa hướng tới nghiệm tốt nhất toàn cục.

3.2 Bài toán tối ưu hóa

Thuật toán PSO được áp dụng để giải quyết bài toán tối ưu hóa tổng quát có dạng:

$$\text{Tìm } \mathbf{x}^* \in \Omega \subseteq \mathbb{R}^n \text{ sao cho } f(\mathbf{x}^*) = \min_{\mathbf{x} \in \Omega} f(\mathbf{x})$$

Trong đó:

- $\mathbf{x} = (x_1, x_2, ..., x_n)$ là vector các biến quyết định.
- $f(\mathbf{x})$ là hàm mục tiêu cần tối thiểu hóa.
- Ω là không gian tìm kiếm.

PSO không yêu cầu tính khả vi hoặc liên tục của $f(\mathbf{x})$, do đó có thể áp dụng linh hoạt cho các bài toán phức tạp hoặc phi tuyến.

3.3 Ý tưởng và mô hình toán học

Mỗi hạt i trong quần thể PSO có:

- Vị trí hiện tại: $\mathbf{x}_i^{(t)} = (x_{i1}^{(t)}, ..., x_{id}^{(t)})$,
- Vận tốc hiện tại: $\mathbf{v}_i^{(t)} = (v_{i1}^{(t)}, ..., v_{id}^{(t)})$,
- Vị trí tốt nhất từng đạt được: $pbest_i$.

Tại mỗi vòng lặp, vận tốc và vị trí của hạt được cập nhật theo:

$$\mathbf{v}_i^{(t+1)} = \omega \mathbf{v}_i^{(t)} + c_1 r_1 (pbest_i - \mathbf{x}_i^{(t)}) + c_2 r_2 (gbest - \mathbf{x}_i^{(t)})$$

$$\mathbf{x}_i^{(t+1)} = \mathbf{x}_i^{(t)} + \mathbf{v}_i^{(t+1)}$$

Trong đó:

- ω: hệ số quán tính – điều khiển mức ảnh hưởng của vận tốc trước đó.
- c_1: hệ số nhận thức cá nhân – điều chỉnh mức học hỏi từ chính mình.
- c_2: hệ số xã hội – điều chỉnh mức học hỏi từ đàn.
- $r_1, r_2 \sim U(0, 1)$ là các số ngẫu nhiên độc lập.

Công thức này cho phép các hạt vừa duy trì quỹ đạo bay cũ, vừa bị "kéo" về các nghiệm tốt hơn, tạo ra quá trình hội tụ dần về vùng tối ưu.

3.4 Chi tiết hoạt động của thuật toán

Thuật toán PSO bao gồm ba giai đoạn chính: khởi tạo, cập nhật quần thể và kiểm tra điều kiện dừng.

3.4.1 Khởi tạo

Tạo ngẫu nhiên quần thể N hạt với vị trí $\mathbf{x}_i$ và vận tốc $\mathbf{v}_i$ ban đầu trong phạm vi giới hạn của không gian tìm kiếm. Đặt $pbest_i = \mathbf{x}_i$, và xác định $gbest$ là nghiệm có giá trị hàm mục tiêu tốt nhất ban đầu.

3.4.2 Cập nhật

Ở mỗi vòng lặp:

- Tính $f(\mathbf{x}_i)$ cho từng hạt.
- Nếu $f(\mathbf{x}_i) < f(pbest_i)$ thì cập nhật $pbest_i = \mathbf{x}_i$.
- Cập nhật $gbest = \arg\min_i f(pbest_i)$.
- Tính vận tốc và vị trí mới của các hạt.

3.4.3 Dừng thuật toán

Quá trình lặp dừng khi đạt số vòng lặp tối đa hoặc khi không còn cải thiện về giá trị $gbest$.

3.5 Pseudo-code thuật toán PSO

Algorithm 1 Thuật toán Particle Swarm Optimization (PSO)

Khởi tạo ngẫu nhiên quần thể $\mathbf{x}_i$, $\mathbf{v}_i$
for mỗi hạt i **do**
    $pbest_i \leftarrow \mathbf{x}_i$
$gbest \leftarrow \arg\min_i f(pbest_i)$
while chưa đạt điều kiện dừng **do**
    for mỗi hạt i **do**
        Cập nhật vận tốc:

$$\mathbf{v}_i = \omega \mathbf{v}_i + c_1 r_1 (pbest_i - \mathbf{x}_i) + c_2 r_2 (gbest - \mathbf{x}_i)$$

        Cập nhật vị trí:

$$\mathbf{x}_i = \mathbf{x}_i + \mathbf{v}_i$$

        if $f(\mathbf{x}_i) < f(pbest_i)$ **then**
            $pbest_i \leftarrow \mathbf{x}_i$
    Cập nhật $gbest = \arg\min_i f(pbest_i)$
Trả về $gbest$

3.6 So sánh với các thuật toán cổ điển

3.6.1 Thiết lập thực nghiệm

Thuật toán PSO được kiểm thử trên các hàm benchmark phổ biến:

- **Ackley:** Hàm nhiều cực trị, khó cho tối ưu toàn cục.
- **Rastrigin:** Hàm nhiều cực trị, dạng lồi nhỏ lặp lại.
- **Rosenbrock:** Hàm thung lũng, khó hội tụ nhanh.
- **Sphere:** Hàm đơn cực trị, lý tưởng để kiểm tra hội tụ.

Các thuật toán so sánh:

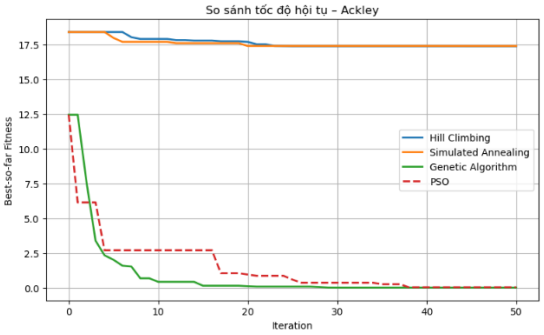
- **PSO:** Particle Swarm Optimization
- **HC:** Hill Climbing
- **SA:** Simulated Annealing

- **GA:** Genetic Algorithm

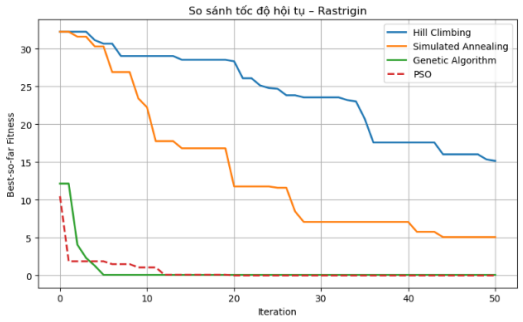
Các tham số PSO được lựa chọn:

$$\omega = 0.7, \quad c_1 = 1.5, \quad c_2 = 1.5, \quad N = 50, \quad \text{MaxIter} = 100(\text{ chỉ vẽ } 50)$$

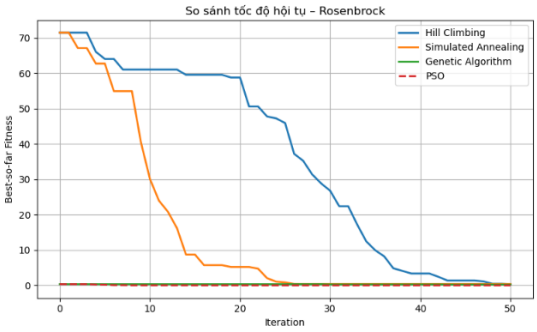
3.6.2 Kết quả hội tụ



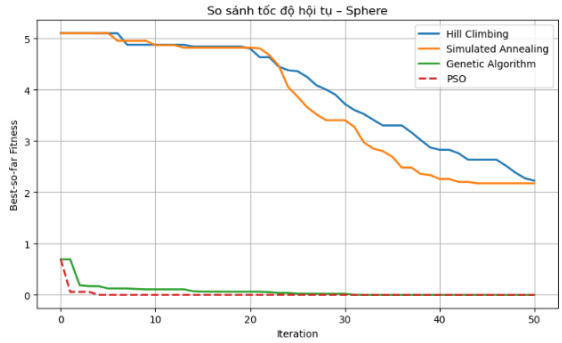
(a) Ackley



(b) Rastrigin



(c) Rosenbrock



(d) Sphere

Hình 3.1: Đồ thị hội tụ của các thuật toán PSO, HC, SA, GA trên 4 hàm benchmark.

3.6.3 Đánh giá kết quả

Từ các đồ thị hội tụ, có thể rút ra các nhận xét về hiệu quả của thuật toán PSO:

- PSO thể hiện khả năng hội tụ nhanh về nghiệm tối ưu hoặc gần tối ưu nhờ cơ chế cập nhật vận tốc kết hợp giữa kinh nghiệm cá nhân (*pbest*) và kinh nghiệm tốt nhất của đàn (*gbest*).
- Đường hội tụ thường mượt và dốc ở giai đoạn đầu, cho thấy các hạt nhanh chóng khám phá không gian tìm kiếm, và dần ổn định khi tiến gần vùng tối ưu.

- Trên các hàm benchmark như Ackley, Rastrigin, Rosenbrock và Sphere, PSO duy trì khả năng hội tụ ổn định, không bị mắc kẹt lâu ở cực trị cục bộ nhờ tính ngẫu nhiên trong cập nhật vận tốc ($r_1, r_2 \sim U(0,1)$) và cơ chế quán tính (ω).
- Hiệu suất hội tụ khác nhau tùy đặc điểm hàm mục tiêu: với hàm đơn cực trị như Sphere, PSO hội tụ rất nhanh; với hàm nhiều cực trị như Ackley và Rastrigin, tốc độ hội tụ chậm hơn nhưng vẫn ổn định và cuối cùng đạt giá trị tốt.

Nhìn chung, PSO cho thấy sự cân bằng hiệu quả giữa khám phá không gian tìm kiếm và khai thác thông tin từ các hạt khác, làm cho thuật toán phù hợp với các bài toán tối ưu hóa phi tuyến, nhiều cực trị và không cần tính khả vi của hàm mục tiêu.

3.7 Phân tích độ nhạy tham số

3.7.1 Thiết lập thực nghiệm

Để đánh giá ảnh hưởng của các tham số PSO đến hiệu quả tối ưu, chúng tôi thực hiện phân tích trên hàm **Sphere**. Các tham số chính được khảo sát:

- Hệ số quán tính ω,
- Hệ số nhận thức cá nhân c_1,
- Hệ số xã hội c_2.

Các tham số được thay đổi trong khoảng hợp lý:

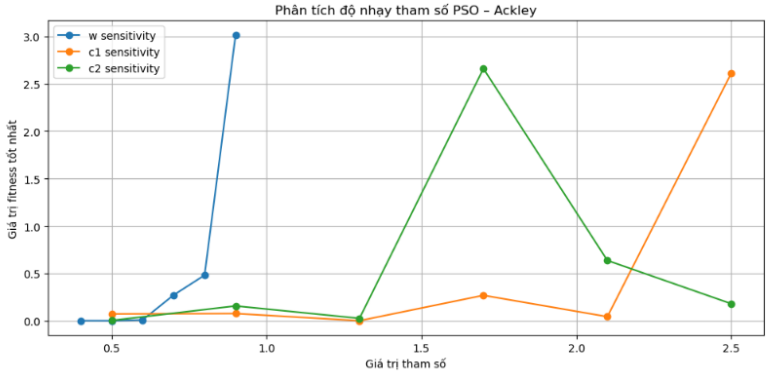
$$\omega \in [0.4, 0.9], \quad c_1, c_2 \in [0.5, 2.5]$$

Các tham số còn lại giữ cố định:

$$w = 0.7, \quad c_1 = 1.5, \quad c_2 = 1.5$$

Mỗi lần thử chạy PSO với số vòng lặp tối đa cố định (ví dụ $max\_iter = 10$), ghi nhận giá trị fitness tốt nhất đạt được ($gbest$) để đánh giá độ nhạy tham số.

3.7.2 Kết quả



Hình 3.2: Đồ thị độ nhạy các tham số PSO trên hàm Sphere. Trục hoành là giá trị tham số, trục tung là giá trị fitness tốt nhất sau số vòng lặp cố định.

3.7.3 Đánh giá và Giải thích Đồ thị Độ nhạy Tham số PSO

Đồ thị **Phân tích độ nhạy tham số PSO – Ackley** hiển thị ảnh hưởng của ba tham số điều khiển chính (ω, c_1, c_2) lên **Giá trị fitness tốt nhất** (tại *gbest*) sau một số vòng lặp cố định khi giải hàm Ackley. Mục tiêu của hàm Ackley là tìm giá trị nhỏ nhất (lý tưởng là 0), do đó, giá trị fitness càng **thấp** càng thể hiện hiệu suất **tốt**.

- **Hệ số Quán tính (ω)**:
  - **Quan sát đồ thị (đường màu xanh dương)**: Khi ω tăng từ 0.4 đến 0.8, giá trị fitness tốt nhất duy trì ở mức rất thấp (gần 0), cho thấy sự hội tụ hiệu quả. Tuy nhiên, tại ω khoảng 0.9 (hoặc lớn hơn), fitness **tăng vọt** lên mức cao (gần 3.0).
  - **Giải thích**: Giá trị ω **thấp** (dưới 0.9) giúp thuật toán **khai thác** (exploitation) tốt các vùng lân cận điểm tốt nhất, dẫn đến kết quả fitness thấp. Giá trị ω **quá cao** (như 0.9) làm hạt duy trì vận tốc cũ quá mức, gây ra **dao động mạnh** hoặc **vượt qua** (overshoot) điểm tối ưu, làm chậm quá trình hội tụ và dẫn đến giá trị fitness tồi hơn sau cùng số vòng lặp.
- **Hệ số Nhận thức Cá nhân (c_1)**:
  - **Quan sát đồ thị (đường màu cam)**: Giá trị fitness duy trì ở mức thấp (gần 0) khi c_1 trong khoảng 0.5 đến 1.4. Tuy nhiên, khi c_1 tăng lên 2.5, fitness **tăng đột ngột** lên mức cao (khoảng 2.6).

  - **Giải thích**: c_1 điều chỉnh sự ảnh hưởng của **kinh nghiệm cá nhân** (*pbest*). Giá trị c_1 **quá cao** ($c_1 = 2.5$) làm hạt **tập trung quá mức** vào lịch sử tìm kiếm cá nhân, giảm khả năng học hỏi từ bầy đàn (*gbest*). Điều này có thể dẫn đến việc **mắc kẹt** ở các cực trị cục bộ hoặc phân tán, làm giảm chất lượng của *gbest* sau số vòng lặp cố định.

- **Hệ số Xã hội (c_2)**:
  - **Quan sát đồ thị (đường màu xanh lá cây)**: Giá trị fitness giữ mức rất thấp trong khoảng $c_2 \in [0.4, 1.4]$. Sau đó, tại $c_2 \approx 1.7$, fitness **tăng lên đỉnh** (khoảng 2.6) rồi giảm trở lại khi c_2 tiếp tục tăng.
  - **Giải thích**: c_2 điều chỉnh sự ảnh hưởng của **kinh nghiệm toàn bầy** (*gbest*). Mức tăng vọt tại $c_2 \approx 1.7$ cho thấy sự mất cân bằng nghiêm trọng: c_2 **quá cao** có thể khiến các hạt bị **kéo về** *gbest* **quá mạnh**, dẫn đến **hội tụ sớm** (premature convergence) tại một điểm tối ưu cục bộ không tốt, hoặc gây ra dao động. Trong khi đó, các giá trị c_2 hợp lý (ví dụ, $c_2 = 2.1$ hoặc $c_2 = 2.5$) cho thấy hiệu suất được cải thiện trở lại, phản ánh sự cân bằng tốt hơn giữa ω, c_1 và c_2.

Tóm tắt: Đồ thị cho thấy **sự nhạy cảm cao** của thuật toán PSO đối với các tham số điều khiển. Hiệu suất tối ưu thường đạt được khi các tham số được giữ trong một phạm vi hẹp và hợp lý (ví dụ: $\omega < 0.9$, $c_1 < 2.0$, $c_2 \in [0.5, 1.4]$ hoặc > 2.0). Việc thay đổi dù chỉ một tham số cũng có thể làm thay đổi đáng kể sự cân bằng giữa khả năng **khám phá** (exploration) và **khai thác** (exploitation) của thuật toán, dẫn đến kết quả fitness tồi đi nhanh chóng.

3.8 Kết luận

Thuật toán PSO là một trong những phương pháp tối ưu hóa metaheuristic phổ biến và hiệu quả nhất hiện nay. Với cấu trúc đơn giản, khả năng hội tụ nhanh và hiệu năng cao trên các bài toán liên tục, PSO được ứng dụng rộng rãi trong nhiều lĩnh vực như học máy, xử lý ảnh, điều khiển tối ưu và khai phá dữ liệu.

Tuy vẫn tồn tại hạn chế như dễ mắc kẹt ở cực trị cục bộ, PSO đã trở thành nền tảng cho nhiều biến thể hiện đại hơn như Adaptive PSO, Quantum PSO, hay Hybrid PSO.

Chương 4

Artificial Bee Colony

4.1 Giới thiệu thuật toán tối ưu Artificial Bee Colony

Thuật toán Artificial Bee Colony (ABC) lần đầu tiên được đề xuất bởi Karaboga trong một báo cáo kĩ thuật về các vấn đề tối ưu số học(Karaboga & Basturk, 2007). ABC là một kĩ thuật tối ưu hóa mô phỏng hành vi kiếm ăn thông minh của loài ong mật. Thuật toán ABC này được phát triển dựa trên thuật toán Particle Swarm Optimization (PSO) - một phương thức tính toán tối ưu bằng cách cố gắng lặp lại việc cải thiện các giải pháp tiềm năng theo một độ đo chất lượng nào đó.
Một số ưu điểm lớn của ABC so với các thuật toán tối ưu hóa khác:

- Tính đơn giản, linh hoạt và mạnh mẽ.
- Sử dụng ít các tham số điều khiển so với nhiều thuật toán tìm kiếm khác.
- Khả năng kết hợp dễ dàng với các thuật toán tối ưu khác.
- Khả năng kiểm soát chi phí mục tiêu.
- Dễ dàng cài đặt với các phép toán và logic cơ bản.

4.2 Bài toán tối ưu hóa

Mục tiêu của thuật toán ABC là tìm giải pháp tối ưu toàn cục cho một bài toán tối ưu hóa hàm số thực. Giả sử bài toán tối ưu hóa cần giải quyết là tìm giá trị lớn nhất của hàm mục tiêu $f(x)$:

$$\max f(x), \quad x \in \mathbb{R}^D \tag{4.1}$$

Với các ràng buộc biên của biến số: $L_j \leq x_j \leq U_j, \quad j = 1, 2, \ldots, D$ Trong đó:

- $x = (x_1, x_2, \ldots, x_D)$ là vectơ đại diện cho một giải pháp (nguồn thức ăn) trong không gian tìm kiếm D chiều.
- L_j và U_j lần lượt là giới hạn dưới (lower bound) và giới hạn trên (upper bound) của chiều thứ j.

Trong ngữ cảnh của thuật toán ABC, ta sẽ tạo ra một bầy ong nhân tạo với nhiệm vụ đi tìm nguồn thức ăn cho ra lượng mật tốt nhất. Vị trí của một nguồn thức ăn đại diện cho một giải pháp khả thi cho bài toán tối ưu hóa, và lượng mật của nguồn thức ăn tương ứng với chất lượng (fitness) của giải pháp đó. Bầy ong có nhiệm vụ phối hợp với nhau để tìm ra nguồn thức ăn tốt nhất.
Ở đây, lưu ý rằng nếu muốn dùng thuật toán ABC cho việc tìm cực tiểu toàn cục, ta sẽ xét bài toán:

$$\arg\min_x f(x) = \arg\max_x -f(x), \quad x \in \mathbb{R}^D$$

Trong đồ án này, ta sẽ chọn các bài toán tìm cực tiểu toàn cục của các hàm liên tục: **Ackley function, Rastrigin function, Rosenbrock function, Sphere function**

4.3 Các nhóm ong

1. **Ong Thợ (Employed Bees/Foragers):**
    - **Vai trò:** Khai thác nguồn thức ăn. Chúng quay trở lại nguồn thức ăn mà chúng đã ghé thăm trước đó và tìm kiếm cải thiện (lượng mật cao hơn) trong vùng lân cận của nguồn hiện tại.

- **Số lượng:** Số lượng Ong Thợ bằng với số lượng nguồn thức ăn xung quanh tổ ong, tức là mỗi nguồn thức ăn chỉ có duy nhất một Ong Thợ gắn với nó. Chúng chiếm 50% tổng số quần thể ong.
- **Chuyển đổi:** Một Ong Thợ sẽ trở thành Ong Do Thám nếu nguồn thức ăn mà nó gắn bó trở nên cạn kiệt (không được cải thiện sau một số lần thử tối đa - *limit*).

2. **Ong Quan Sát (Onlookers):**

- **Vai trò:** Đưa ra quyết định lựa chọn nguồn thức ăn để tiếp tục khai thác. Chúng theo dõi điệu nhảy thông báo (waggle dance) của Ong Thợ để đánh giá lượng mật (chất lượng giải pháp). Hiểu đơn giản là Ong Quan sát sẽ giao tiếp với các Ong Thợ để lựa chọn nguồn thức ăn.
- **Cơ chế chọn:** Chúng ưu tiên lựa chọn các nguồn thức ăn có lượng mật cao hơn, vì xác suất để một nguồn thức ăn được Ong Quan Sát chọn sẽ tăng lên cùng với lượng mật được công bố.
- **Số lượng:** Bằng với số lượng Ong Thợ và cũng chiếm 50% tổng số quần thể ong.

3. **Ong Do Thám (Scouts):**

- **Vai trò:** Khám phá các khu vực tìm kiếm mới. Chúng thực hiện việc tìm kiếm ngẫu nhiên (random search) để khám phá các nguồn thức ăn hoàn toàn mới, thay thế cho các nguồn đã bị Ong Thợ trước đó bỏ đi.
- **Cơ chế hình thành:** Được hình thành từ các Ong Thợ mà nguồn thức ăn của chúng đã cạn kiệt.
- **Số lượng:** Trong mỗi vòng lặp, thuật toán chỉ lấy nhiều nhất một con Ong Do Thám để tìm kiếm nguồn thức ăn mới (mặc dù nhiều Ong Thợ có thể đạt điều kiện chuyển đổi).

4.4 Ý tưởng chính của thuật toán

1. **Khởi tạo:**

- Khởi tạo quần thể ong, phân bổ 50% là **Ong Thợ** và 50% là **Ong Quan Sát**.
- Gán ngẫu nhiên các **giải pháp tiềm năng (nguồn thức ăn)** cho từng Ong Thợ.

- Đặt một **giới hạn (limit)** cho mỗi nguồn thức ăn, quy định số lần không cải thiện tối đa trước khi nguồn đó bị bỏ.

2. **Quá trình Lặp:** Lặp lại ba giai đoạn tìm kiếm chính cho đến khi đạt điều kiện dừng:

   (a) **Giai đoạn Ong Thợ (Khai thác):**

   - Ong Thợ đến nguồn thức ăn đã ghi nhớ, thực hiện tìm kiếm cục bộ (local search) để cải thiện lượng mật (chất lượng giải pháp).
   - Sau đó, chúng chia sẻ thông tin lượng mật thu được tại khu vực nhảy (dance area[1]).

   (b) **Giai đoạn Ong Quan Sát (Khai thác):**

   - Ong Quan Sát chọn nguồn thức ăn dựa trên thông tin lượng mật được chia sẻ bởi Ong Thợ. Việc lựa chọn này là ưu tiên theo xác suất, các nguồn có lượng mật cao hơn sẽ có cơ hội được chọn cao hơn.
   - Ong Quan Sát thực hiện tìm kiếm khai thác thêm tại nguồn đã chọn để cải thiện chất lượng giải pháp.

   (c) **Giai đoạn Ong Do Thám (Khám phá):**

   - Xác định các nguồn thức ăn đã đạt đến giới hạn (*limit*) và bị bỏ đi.
   - Ong Thợ gắn với các nguồn bị bỏ sẽ chuyển thành **Ong Do Thám**.
   - Ong Do Thám thực hiện tìm kiếm ngẫu nhiên (random search) để khám phá một nguồn thức ăn mới hoàn toàn, qua đó đảm bảo sự đa dạng của giải pháp và khả năng thoát khỏi cực tiểu cục bộ. (Chỉ tối đa một Ong Do Thám được thực hiện tìm kiếm mới trong một vòng lặp).

3. **Điều kiện Dừng:** Dừng thuật toán khi thỏa mãn các yêu cầu hoặc điều kiện đặt trước (ví dụ: đạt số vòng lặp tối đa, đạt mức chất lượng giải pháp mong muốn,...).

Tóm lại, thuật toán ABC duy trì sự cân bằng giữa tìm kiếm khai thác (do Ong Thợ và Ong Quan Sát thực hiện) và tìm kiếm khám phá (do Ong Do Thám thực hiện) để tìm ra giải pháp tối ưu toàn cục.

[1]Ngữ cảnh sinh vật học: nơi mà những con Ong Thợ sau khi tìm kiếm thức ăn thực hiện điệu nhảy thông báo để chia sẻ thông tin về vị trí và chất lượng của nguồn thức ăn với những con ong khác đang ở trong tổ.

4.5 Chi tiết

Trong thuật toán ABC,

- Vị trí của các nguồn thức ăn đại diện cho các giải pháp tiềm năng của một bài toán tối ưu hóa.
- Lượng mật khai thác được từ nguồn thức ăn đó đại diện cho chất lượng (độ thích hợp / fitness) của một giải pháp.
- Số lượng của Ong Thợ hoặc Ong Quan sát tương ứng với số lượng giải pháp.

4.5.1 Khởi tạo

Tại bước đầu tiên, ta khởi tạo một tổng thể có phân phối ngẫu nhiên (phân phối đều) $P(G = 0)$ gồm SN giải pháp (vị trí các nguồn thức ăn). Trong đó SN kí hiệu cho kích thước của tập tổng thể giải pháp. $P(G = 0)$ là phân phối tổng thể SN giải pháp tại thời điểm khởi đầu (thế hệ thứ 0). Mỗi giải pháp (nguồn thức ăn) $x_i, i \in \{1, 2, \ldots, SN\}$ là một vector D-chiều với D là số tham số cần tối ưu.
Sau khi khởi tạo, các giải pháp sẽ được đưa vào các vòng lặp $C = 1, 2, \ldots, C_{\max}$ gồm các quá trình tìm kiếm của Ong Thợ, Ong Quan sát, Ong Do Thám.

4.5.2 Các quá trình tìm kiếm

Mỗi con Ong Thợ hoặc Ong Quan sát sẽ tạo một điều chỉnh vào vị trí nguồn thức ăn (giải pháp) trong trí nhớ của nó để tìm nguồn thức ăn mới và kiểm tra lượng mật (giá trị fitness) thu được từ nguồn thức ăn mới này (giải pháp mới). Thực tế, các con Ong điều chỉnh tìm ra nguồn thức ăn mới dựa trên việc quan sát xung quanh để so sánh các nguồn thức ăn. Tuy nhiên, những con ong nhân tạo trong mô hình này sẽ không dùng bất cứ thông tin gì trong việc so sánh, mà thay vào đó ngẫu nhiên chọn một vị trí nguồn thức ăn khác và tạo ra một sự điều chỉnh lên vị trí nguồn thức ăn hiện tại trong trí nhớ của nó. Nếu như lượng mật tại nguồn thức ăn mới tăng lên, con Ong sẽ thay thế vị trí nguồn thức ăn cũ trong trí nhớ, ngược lại thì sẽ giữ nguyên vị trí cũ.
Sau khi Ong Thợ hoàn thành việc tìm kiếm, chúng sẽ chia sẻ thông tin về lượng mật thu được cùng vị trí nguồn thức ăn cho Ong Quan sát. Ong Quan sát sẽ thực hiện đánh giá tất cả các thông tin

về lượng mật từ tất cả Ong Thợ, từ đó chọn ra nguồn thức ăn với xác suất dựa trên lượng mật. Giống như Ong Thợ, Ong Quan sát cũng thực hiện việc điều chỉnh vị trí nguồn thức ăn trong trí nhớ của nó. Nếu như lượng mật tại nguồn thức ăn mới tăng lên, con Ong sẽ thay thế vị trí nguồn thức ăn cũ trong trí nhớ, ngược lại thì sẽ giữ nguyên vị trí cũ. Việc lựa chọn một nguồn thức ăn của Ong Quan sát dựa trên giá trị xác suất tương ứng với nguồn thức ăn đó:

$$p_i = \frac{\text{fit}_i}{\sum_{n=1}^{SN} \text{fit}_n} \tag{4.2}$$

Trong đó, fit_i là giá trị fitness của giải pháp thứ i ứng với Ong Thợ tương ứng.
Để tạo ra một vị trí nguồn thức ăn tiềm năng từ vị trí cũ, ABC dùng:

$$v_{ij} = x_{ij} + \phi_{ij}(x_{ij} - x_{kj}) \tag{4.3}$$

với

- $k \in \{1, 2, \ldots, SN\}, k \neq i$ được chọn ngẫu nhiên
- $j \in \{1, 2, \ldots, D\}$ được chọn ngẫu nhiên
- $\phi_{ij} \in [-1, 1]$ là một số ngẫu nhiên, kiểm soát việc tạo ra các vị trí các nguồn thức ăn xung quanh vị trí x_{ij} và việc sửa đổi thể hiện sự so sánh các vị trí thức ăn lân cận một cách trực quan bởi con ong

Phương trình trên cho thấy rằng khi chênh lệch giữa các tham số của x_{ij} và x_{kj} giảm thì độ nhiễu trên vị trí x_{ij} cũng giảm theo. Do đó, khi việc tìm kiếm tiến tới giải pháp tối ưu trong không gian tìm kiếm, độ dài bước sẽ giảm đi.

Cơ chế giới hạn kết quả tìm kiếm giải pháp mới: Trong quá trình tạo ra một giải pháp mới (v_{ij}) bằng công thức trên, giá trị của một tham số có thể vượt quá giới hạn tối đa hoặc tối thiểu đã được xác định trước cho bài toán tối ưu hóa. Nếu một tham số được tạo ra vượt quá giới hạn của nó, giá trị của tham số đó sẽ được đặt bằng chính giá trị giới hạn đó. Cơ chế này đảm bảo rằng các giải pháp ứng cử viên luôn nằm trong không gian tìm kiếm hợp lệ đã được định nghĩa
Những nguồn thức ăn bị đàn ong bỏ đi sẽ được thay thế bởi những nguồn thức ăn mới do Ong Do Thám tìm được. Trong ABC, những vị trí bị bỏ này sẽ được thay thế bởi một vị trí mới được tạo

ngẫu nhiên. Bên cạnh đó, nếu như một vị trí nguồn thức ăn không thể cải thiện thêm qua một số lượng vòng lặp nhất định (limit), thì nguồn thức ăn này cũng sẽ bị bỏ đi.

Cơ chế chọn lọc tham lam: Sau khi mỗi vị trí ứng viên v_{ij} được tạo ra và sau đó được ong nhân tạo đánh giá, chất lượng của nó được so sánh với chất lượng của x_{ij}. Nếu thức ăn mới có mật hoa bằng hoặc tốt hơn nguồn cũ thì nó sẽ được thay thế bằng nguồn cũ trong bộ nhớ. Nếu không, cái cũ sẽ được giữ lại. Nói cách khác, cơ chế chọn lọc tham lam (Greedy Selection Mechanism) được sử dụng như một hoạt động chọn lọc giữa nguồn thức ăn cũ và hiện tại. Cơ chế này đảm bảo rằng mỗi bước cải tiến cục bộ chỉ giữ lại những giải pháp tốt hơn hoặc không làm giảm chất lượng giải pháp.

Như vậy, Thuật toán ABC thực tế sử dụng bốn quá trình chọn lọc khác nhau:

1. **Quá trình Chọn lọc Toàn cục (Global Selection):**
   - Được sử dụng bởi các Ong Quan sát để khám phá các vùng tiềm năng.
   - Được mô tả bởi biểu thức 4.2
2. **Quá trình Chọn lọc Cục bộ (Local Selection):**
   - Được thực hiện trong một khu vực bởi các Ong Thợ, Ong Quan sát.
   - Quá trình này dựa trên thông tin cục bộ để xác định một nguồn thức ăn lân cận xung quanh nguồn trong bộ nhớ.
   - Được định nghĩa trong biểu thức 4.3
3. **Quá trình Chọn lọc Tham lam Cục bộ (Greedy Selection):**
   - Được thực hiện bởi tất cả các con ong.
   - Nếu lượng mật của nguồn ứng cử viên tốt hơn nguồn hiện tại, con ong quên nguồn hiện tại và ghi nhớ nguồn ứng cử viên.
   - Nếu không, con ong giữ lại nguồn hiện tại trong bộ nhớ
4. **Quá trình Chọn lọc Ngẫu nhiên (Random Selection):**
   - Được thực hiện bởi các Ong Do Thám (*scouts*)

Ta có thể hiểu đơn giản như sau. Trong thuật toán ABC, không phải mỗi con ong giữ một bản đồ riêng, mà toàn bộ tổ ong sẽ duy trì một Danh sách chung (trong lập trình thường là một mảng hoặc ma trận) chứa các nguồn thức ăn hiện tại.
1. Hãy tưởng tượng có một cái bảng thông báo lớn đặt tại tổ ong. Trên bảng này ghi danh sách các toạ độ nguồn thức ăn hiện có. Ví dụ, nếu có 50 Ong Thợ, thì trên bảng sẽ có 50 dòng dữ liệu. Mỗi dòng dữ liệu trên bảng này chứa 3 thông tin quan trọng:

- Vị trí (x): Tọa độ của nguồn thức ăn (nghiệm của bài toán).
- Lượng mật ($fitness$): Chất lượng của nguồn thức ăn đó (giá trị hàm mục tiêu).
- Bộ đếm thất bại ($limit$): Số lần tìm kiếm xung quanh mà không thấy cái nào ngon hơn.

2. Cách các loại ong tương tác với danh sách này: Các loại ong sẽ "đọc" và "ghi" vào danh sách chung này theo các cách khác nhau:

- Mỗi con Ong Thợ được chỉ định phụ trách cố định một dòng trên bảng. Nó đi tìm xung quanh vị trí đó. Nếu tìm thấy vị trí tốt hơn, nó sẽ về tổ xoá dòng cũ và ghi đè dòng mới lên bảng.
- Mỗi con Ong Quan Sát đứng nhìn toàn bộ cái bảng (đây chính là hình ảnh ẩn dụ của "dance area"). Nó không sở hữu riêng nguồn nào cả. Nó tính toán xác suất dựa trên cột "Lượng mật" của toàn bộ danh sách.Nó chọn một dòng (ví dụ dòng số 5) để bay đến thử. Nếu nó tìm thấy chỗ tốt hơn, nó sẽ cập nhật lại dòng số 5 trên bảng chung. Lúc này, con Ong Thợ đang phụ trách dòng số 5 sẽ được hưởng lợi từ phát hiện này của Ong Quan Sát.
- Nếu như nguồn thức ăn không được cập nhật sau khi một con ong thực hiện tìm kiếm tại đó, bản ghi của nguồn thức ăn đó sẽ tăng bộ đếm lên 1. Nếu thấy dòng nào có bộ đếm vượt quá giới hạn, Ong Thợ ứng với dòng đó trở thành Ong Do thám, sẽ xoá dòng đó đi và điền vào một toạ độ ngẫu nhiên mới hoàn toàn.

4.6 Pseudo-code

Algorithm 2 Thuật toán Bầy ong nhân tạo (ABC)

Khởi tạo: Khởi tạo ngẫu nhiên các nguồn thức ăn x_i với $i = 1, \ldots, SN$ và đánh giá chất lượng fit_i cho mỗi nguồn thức ăn. Đặt bộ đếm số lần nguồn thức ăn không cập nhật $trial_i \leftarrow 0$
while Chưa thỏa mãn mãn điều kiện dừng **do**
// Giai đoạn Ong Thợ
for $i = 1$ **to** SN **do** *// Xét từng nguồn thức ăn ứng với từng Ong Thợ*
Chọn ngẫu nhiên $k \in \{1, \ldots, SN\}, k \neq i$ và chiều j
Tạo giải pháp mới $v_{ij} \leftarrow x_{ij} + \phi_{ij}(x_{ij} - x_{kj})$
Đánh giá chất lượng $fit(v_i)$
if $fit(v_i) > fit(x_i)$ **then** *// Chọn được nguồn mới tốt hơn*
$x_i \leftarrow v_i$
$trial_i \leftarrow 0$
else
$trial_i \leftarrow trial_i + 1$
// Giai đoạn Ong Quan Sát
Tính xác suất lựa chọn P_i dựa trên fit_i
for $t = 1$ **to** SN **do** *// từng Ong Quan sát thực hiện tìm kiếm*
Chọn nguồn thức ăn i dựa trên xác suất P_i
Chọn ngẫu nhiên $k \neq i$ và chiều j
Tạo giải pháp mới $v_{ij} \leftarrow x_{ij} + \phi_{ij}(x_{ij} - x_{kj})$
Đánh giá chất lượng $fit(v_i)$
if $fit(v_i) > fit(x_i)$ **then** *// Chọn được nguồn mới tốt hơn*
$x_i \leftarrow v_i$
$trial_i \leftarrow 0$
else
$trial_i \leftarrow trial_i + 1$
// Giai đoạn Ong Do Thám
if tồn tại i sao cho $trial_i > limit$ **then**
$x_i \leftarrow$ giải pháp ngẫu nhiên mới trong không gian tìm kiếm
$trial_i \leftarrow 0$
Ghi nhớ giải pháp tốt nhất tìm thấy cho đến nay x_{best}

4.7 Thực nghiệm

Bảng 4.1: Cấu hình mặc định các Tham Số Chính của `ArtificialBeeColony`

Tham Số	Ý Nghĩa	Giá Trị Mặc Định
`population_size`	Kích thước quần thể	40
`limit`	Giới hạn giữ nguồn hiện tại	10

4.7.1 Áp dụng

Áp dụng `ArtificialBeeColony` cho các bài toán tối ưu toàn cục các hàm Ackley, Rastrigin, Rosenbrock, Sphere ta có bảng kết quả sau:

Bảng 4.2: Tổng hợp kết quả thực nghiệm thuật toán ABC

Function	**Điều kiện dừng**	**Best Fitness**	**Best Solution (X)**
Ackley	Iter (50)	0.002920	$[-0.0003, -0.0010]$
	NFE (5000)	0.000017	$[-0.0000, -0.0000]$
Rastrigin	Iter (50)	0.000428	$[0.0009, -0.0012]$
	NFE (5000)	0.000001	$[-0.0000, 0.0001]$
Rosenbrock	Iter (50)	0.072212	$[0.7340, 0.5349]$
	NFE (5000)	0.024227	$[1.1553, 1.3358]$
Sphere	Iter (50)	0.000000	$[0.0000, -0.0000]$
	NFE (5000)	0.000000	$[0.0000, 0.0000]$

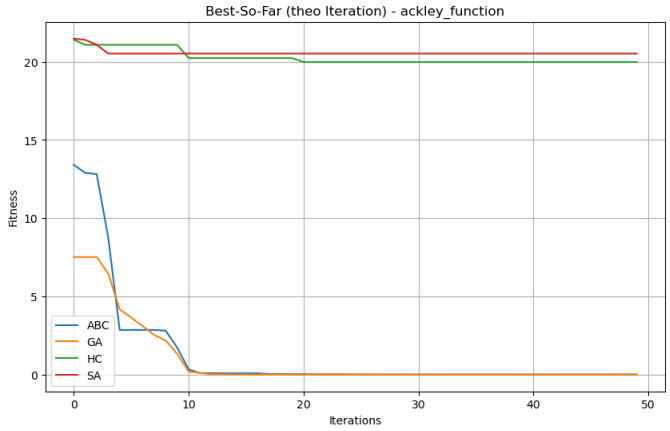
Nhận xét: Có thể thấy rằng thuật toán ABC cho ra kết quả rất sát với giá trị nhỏ nhất của các hàm số trên.

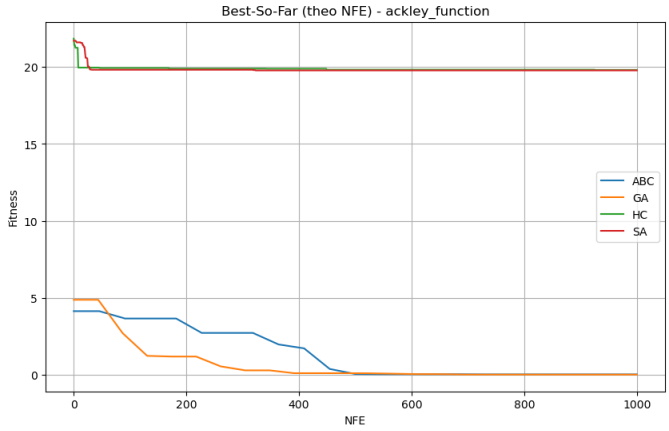
4.7.2 So sánh với các thuật toán khác

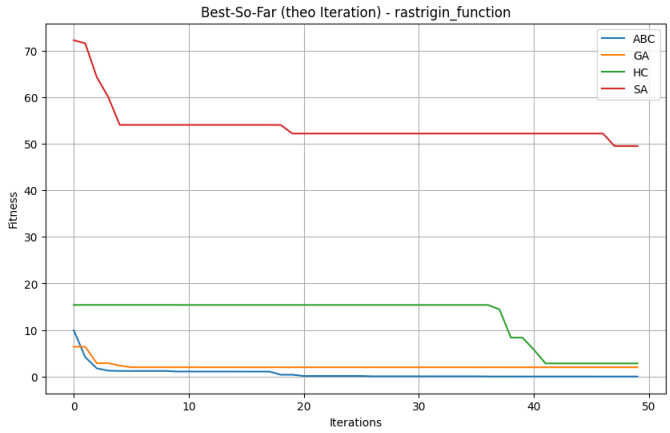
Sau đây, ta sẽ so sánh tốc độ hội tụ của thuật toán Artificial Bee Colony với các thuật toán truyền thống khác như Hill Climbing, Simulated Annealing và Genetic Algorithm. Các tham số của các thuật toán truyền thống được cài như trong bảng sau

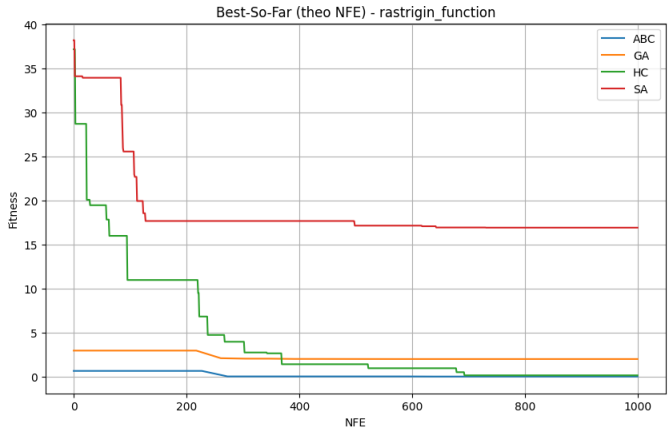
Bảng 4.3: Cấu hình mặc định của `HillClimbing, SimulatedAnnealing, GeneticAlgorithm`

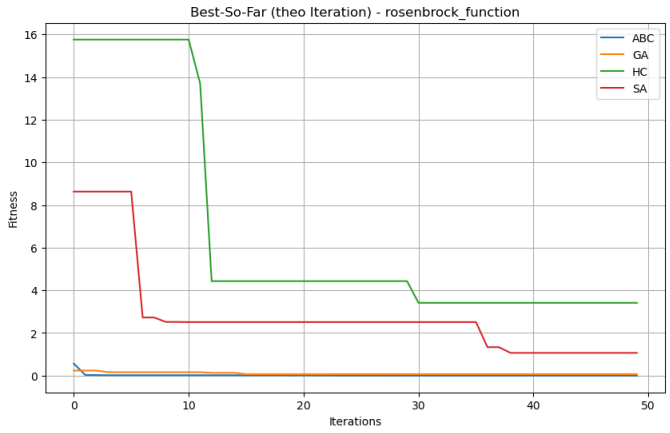
Thuật Toán	**Tham Số**	**Ý Nghĩa**	**Giá Trị Mặc Định**
`HillClimbing`	`step_size`	Bước nhảy	1
`SimulatedAnnealing`	`initial_temp`	Nhiệt đô ban đầu	1000
	`cooling_rate`	Hệ số làm mát	0.99
	`step_size`	Bước nhảy	0.1
`GeneticAlgorithm`	`population_size`	Kích thước quần thể	40
	`crossover_rate`	Xác suất lai ghép	0.8
	`mutation_rate`	Tỷ lệ đột biến	0.1

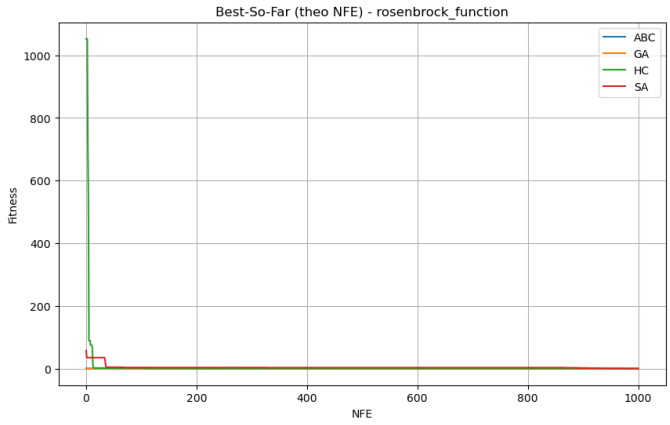

Best-So-Far (theo Iteration) - ackley_function
Fitness
Iterations
ABC
GA
HC
SA

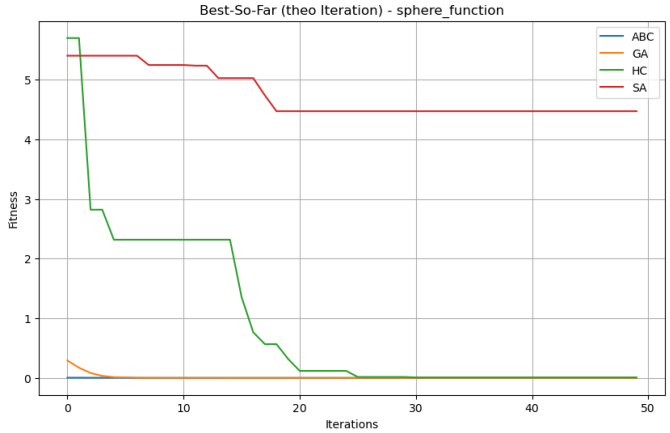

Best-So-Far (theo NFE) - ackley_function
Fitness
NFE
ABC
GA
HC
SA

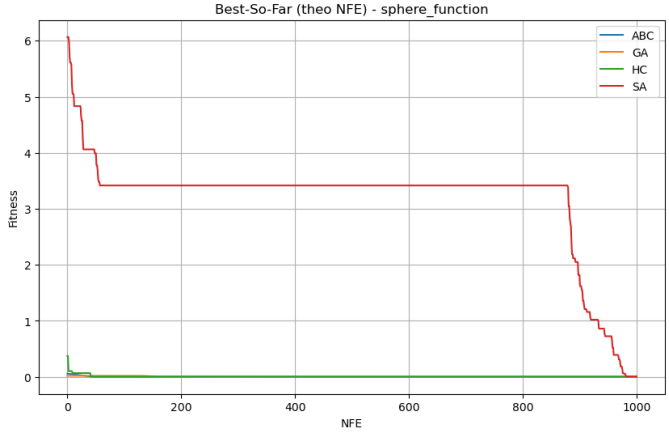

Best-So-Far (theo Iteration) - rastrigin_function
Fitness
Iterations
ABC
GA
HC
SA


Best-So-Far (theo NFE) - rastrigin_function
Fitness
NFE
ABC
GA
HC
SA


Best-So-Far (theo Iteration) - rosenbrock_function
Fitness
Iterations
ABC
GA
HC
SA


Best-So-Far (theo NFE) - rosenbrock_function
Fitness
NFE
ABC
GA
HC
SA


Best-So-Far (theo Iteration) - sphere_function
Fitness
Iterations
ABC
GA
HC
SA

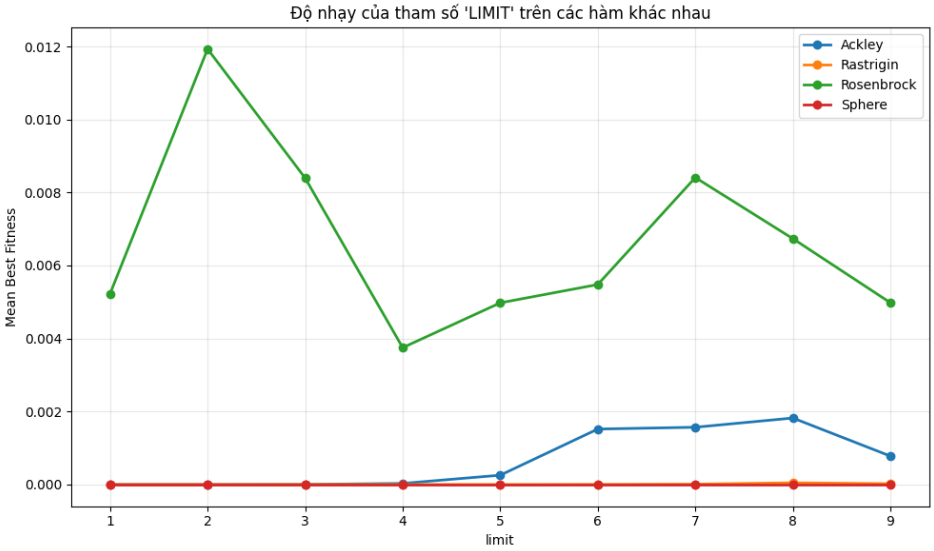

Best-So-Far (theo NFE) - sphere_function
Fitness
NFE
ABC
GA
HC
SA

Nhận xét: Đối với việc tìm giá trị cực tiểu toàn cục của các hàm Ackley, Rastrigin, Rosenbrock và Sphere, Artificial Bee Colony là lựa chọn tối ưu nhất với tốc độ hội tụ nhanh, không bị mắc kẹt tại các cực tiểu cục bộ. Các thuật toán Hill Climbing, Simulated Annealing, Genetic Algorithm hoặc là hội tụ chậm hoặc là bị mắc kẹt sớm tại các cực tiểu cục bộ, cần được tinh chỉnh lại tham số mới có hy vọng giải quyết tốt bài toán này

4.7.3 Phân tích độ nhạy của thuật toán

4.7.3.1 Tham số limit

Tham số `limit` thường quy định số lần thử nghiệm tối đa mà một lời giải không được cải thiện trước khi bị loại bỏ để tạo sinh lời giải ngẫu nhiên mới.

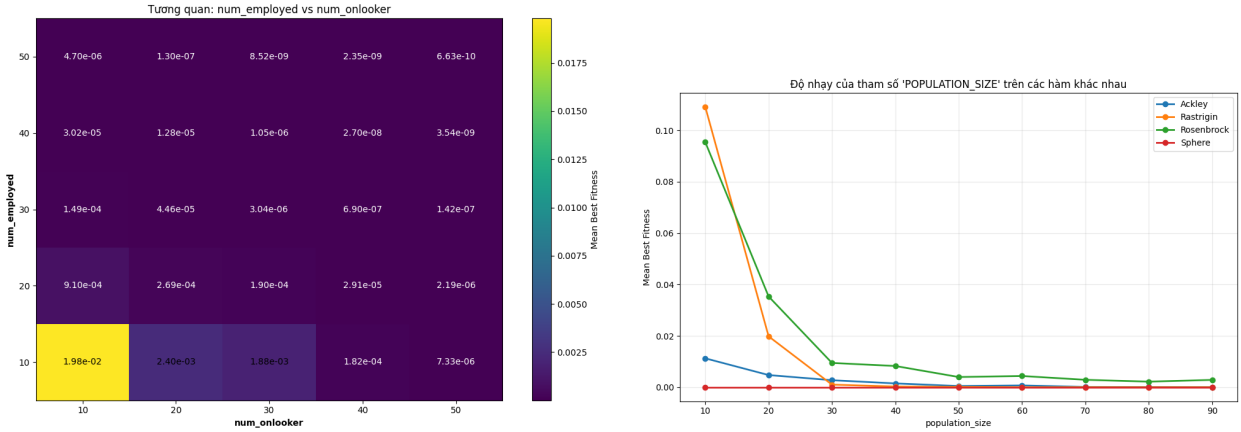


Hình 4.1: Đồ thị độ nhạy của thuật toán khi limit thay đổi

Nhận xét: Hàm Rosenbrock là hàm nhạy cảm nhất với tham số `limit`. Độ lỗi (Mean Best Fitness) dao động rất mạnh `limit` tăng từ 1 đến 9. Điều này cho thấy với hàm Rosenbrock (một hàm khó với thung lũng hẹp), việc chọn limit không phù hợp có thể làm giảm đáng kể hiệu suất tìm kiếm. Hàm Ackley có xu hướng tăng nhẹ lỗi khi `limit` tăng lên (đặc biệt từ mức 5 trở đi), nhưng biến động nhỏ hơn nhiều so với Rosenbrock. Còn với hàm Rastrigin và Sphere thì đường biểu diễn nằm ngang sát trục 0.Điều này cho thấy hai hàm này không nhạy cảm với tham số `limit` trong khoảng thử nghiệm này. Thuật toán tìm được lời giải tối ưu bất kể giá trị `limit`.
→ Kết luận: `limit` là một tham số khó tinh chỉnh đối với các hàm phức tạp như Rosenbrock, nhưng

ít ảnh hưởng đến các hàm đơn giản hoặc đa cực trị có cấu trúc rõ ràng như Rastrigin/Sphere.

4.7.3.2 Các Tham số về số lượng cá thể của quần thể ong



Nhận xét: Tương quan giữa `num_employed_bees` và `num_onlooker_bees`: Biểu đồ nhiệt này thể hiện mối quan hệ giữa số lượng ong thợ `num_employed_bees` và ong quan sát `num_onlooker_bees` đối với độ lỗi. Khu vực hiệu suất thấp nhất (Màu vàng): Xảy ra tại góc dưới bên trái khi cả hai tham số đều nhỏ nhất (`num_employed_bees`=10 và `num_onlooker_bees`=10). Giá trị lỗi tại đây cao hơn hẳn các ô khác. Khu vực hiệu suất cao nhất (Màu tím đậm): Khi tăng số lượng ong (di chuyển sang phải và lên trên), độ lỗi giảm đi nhanh chóng. Ô tốt nhất là góc trên bên phải (50, 50) với lỗi cực nhỏ. Sự tác động: Việc tăng `num_onlooker_bees` (trục hoành) có vẻ mang lại hiệu quả giảm lỗi rất nhanh. Việc tăng `num_employed_bees` (trục tung) cũng giúp cải thiện kết quả nhưng có vẻ cần sự kết hợp với lượng onlooker đủ lớn để đạt hiệu quả tối đa.

→ Kết luận: Kích thước quần thể (tổng employed + onlooker) càng lớn thì khả năng hội tụ về nghiệm tối ưu càng cao. Có mối tương quan tỷ lệ thuận rõ ràng giữa tài nguyên tính toán (số lượng cá thể) và độ chính xác.

Bên cạnh đó, `population_size` cũng cho thấy ảnh hưởng trực tiếp của kích thước quần thể lên khả năng tìm kiếm. Kích thước quần thể quá nhỏ (< 20) là nguyên nhân chính dẫn đến hiệu suất kém trên các hàm phức tạp. Có một ngưỡng "bão hòa" (khoảng 40-50). Việc tăng kích thước quần thể vượt quá ngưỡng này (lên 80, 90) không mang lại lợi ích rõ rệt về độ chính xác nhưng lại làm tăng chi phí tính toán.

4.8 Kết luận

Thuật toán Artificial Bee Colony là thuật toán tìm kiếm meta-heuristic có khả năng tối ưu rất tốt. Đây là thuật toán hiệu quả nhất trong số các thuật toán tìm kiếm được so sánh (bao gồm Hill Climbing, Simulated Annealing, và Genetic Algorithm) để giải quyết các bài toán tối ưu hóa hàm liên tục.

ABC cho thấy tốc độ hội tụ nhanh và khả năng tìm ra lời giải rất sát với giá trị cực tiểu toàn cục của các hàm benchmark (Ackley, Rastrigin, Rosenbrock, và Sphere). Ưu điểm lớn nhất của ABC so với các thuật toán truyền thống là khả năng không bị mắc kẹt tại các cực tiểu cục bộ. Trong khi HC, SA, và GA bị kẹt sớm hoặc hội tụ chậm, ABC vẫn tiếp tục tìm kiếm và hội tụ về giải pháp tốt hơn.

Bên cạnh đó, Một ưu điểm khác của ABC là tính đơn giản, linh hoạt, mạnh mẽ và sử dụng ít tham số điều khiển so với nhiều thuật toán tìm kiếm khác. Chính vì ít tham số hơn nên thuật toán ABC cũng phụ thuộc đáng kể vào việc lựa chọn tham số:

- limit: Tham số này (số lần thử tối đa trước khi loại bỏ một giải pháp) có độ nhạy cao, đặc biệt với các hàm phức tạp như Rosenbrock.
- Kích thước quần thể: Kích thước quần thể quá nhỏ (dưới 20) là nguyên nhân chính dẫn đến hiệu suất kém. Tuy nhiên, có một ngưỡng "bão hòa" (khoảng 40-50), mà việc tăng kích thước vượt ngưỡng này không cải thiện đáng kể kết quả nhưng làm tăng chi phí tính toán.

Chương 5

Firely Algorithm

5.1 Giới thiệu

Thuật toán **Firefly Algorithm (FA)** được đề xuất bởi Xin-She Yang vào năm 2008 (Yang, 2008), là một thuật toán tối ưu hóa dựa trên quần thể (*population-based optimization*) lấy cảm hứng từ hành vi nhấp nháy và tương tác xã hội của loài đom đóm.

FA được thiết kế dựa trên sự hấp dẫn giữa các đom đóm, trong đó mức độ hấp dẫn tỉ lệ với độ sáng của chúng. Độ sáng của một đom đóm tương ứng với giá trị hàm mục tiêu (*fitness*) tại vị trí của nó. Các quy tắc cơ bản của thuật toán FA là:

- Đom đóm càng sáng (fitness càng tốt) thì càng thu hút các đom đóm khác.
- Mức độ hấp dẫn giảm khi khoảng cách giữa hai đom đóm tăng lên.
- Nếu không có đom đóm nào sáng hơn, đom đóm sẽ di chuyển ngẫu nhiên.

FA nổi bật với khả năng cân bằng tốt giữa **khám phá** (exploration) nhờ sự di chuyển ngẫu nhiên và **khai thác** (exploitation) nhờ sự hấp dẫn, khiến nó hiệu quả trên các bài toán tối ưu đa cực trị.

5.2 Bài toán tối ưu hóa

Thuật toán FA cũng được áp dụng để giải quyết bài toán tối ưu hóa tổng quát có dạng:

$$\text{Tìm } \mathbf{x}^* \in \Omega \subseteq \mathbb{R}^n \text{ sao cho } f(\mathbf{x}^*) = \min_{\mathbf{x}\in\Omega} f(\mathbf{x})$$

Trong đó:

- $\mathbf{x} = (x_1, x_2, ..., x_n)$ là vector các biến quyết định (vị trí của đom đóm).
- $f(\mathbf{x})$ là hàm mục tiêu cần tối thiểu hóa (độ sáng của đom đóm).
- Ω là không gian tìm kiếm.

5.3 Ý tưởng và mô hình toán học

5.3.1 Độ sáng (I)

Độ sáng (I_i) của đom đóm i tỉ lệ thuận với giá trị hàm mục tiêu $f(\mathbf{x}_i)$. Đối với bài toán tối thiểu hóa, ta có thể đặt $I_i \propto 1/f(\mathbf{x}_i)$ hoặc đơn giản là sử dụng $f(\mathbf{x}_i)$ để so sánh trực tiếp, nhưng thường **độ sáng** được định nghĩa sao cho **đom đóm tốt hơn sẽ sáng hơn**.

5.3.2 Độ hấp dẫn (beta)

Độ hấp dẫn (β) giữa hai đom đóm i và j được tính theo khoảng cách r_{ij}:

$$\beta(r) = \beta_0 e^{-\gamma r^2}$$

Trong đó:

- β_0: độ hấp dẫn tối đa tại khoảng cách $r = 0$.
- γ: hệ số hấp thụ ánh sáng/phạm vi hấp dẫn.
- $r_{ij} = ||\mathbf{x}_i - \mathbf{x}_j||$: khoảng cách Euclidean giữa đom đóm i và j.

5.3.3 Cập nhật vị trí

Nếu đom đóm i di chuyển về phía đom đóm j (vì $I_j > I_i$), vị trí của i được cập nhật theo công thức:

$$\mathbf{x}_i^{(t+1)} = \mathbf{x}_i^{(t)} + \beta(r_{ij})(\mathbf{x}_j^{(t)} - \mathbf{x}_i^{(t)}) + \alpha\epsilon_i$$

Trong đó:

- $\beta(r_{ij})(\mathbf{x}_j^{(t)} - \mathbf{x}_i^{(t)})$ là thành phần hấp dẫn.
- ϵ_i là vector ngẫu nhiên (thường từ phân phối Gaussian hoặc Uniform).
- α: hệ số ngẫu nhiên hóa (randomization parameter) – điều khiển mức độ di chuyển ngẫu nhiên.

5.4 Chi tiết hoạt động của thuật toán

5.4.1 Khởi tạo

Tạo ngẫu nhiên quần thể N đom đóm $\mathbf{x}_i$ trong phạm vi giới hạn. Tính độ sáng $I_i = f(\mathbf{x}_i)$.

5.4.2 Cập nhật

Ở mỗi vòng lặp:

- Duyệt qua tất cả các cặp đom đóm (i, j).
- Nếu $I_j < I_i$ (tức là j tốt hơn i trong bài toán tối thiểu hóa), tính toán r_{ij} và $\beta(r_{ij})$.
- Cập nhật vị trí $\mathbf{x}_i$ theo công thức trên, di chuyển i về phía j.
- Nếu $I_j \geq I_i$, đom đóm i di chuyển ngẫu nhiên theo công thức $\mathbf{x}_i^{(t+1)} = \mathbf{x}_i^{(t)} + \alpha\epsilon_i$.
- Cập nhật lại độ sáng $I_i = f(\mathbf{x}_i)$.
- Cập nhật đom đóm tốt nhất toàn cục (*gbest*) dựa trên độ sáng.

5.4.3 Dừng thuật toán

Quá trình lặp dừng khi đạt số vòng lặp tối đa hoặc khi không còn cải thiện đáng kể về giá trị *gbest*.

5.5 Pseudo-code thuật toán FA

Algorithm 3 Thuật toán Firefly Algorithm (FA)

Thiết lập hàm mục tiêu $f(\mathbf{x})$, tham số α, β_0, γ.
Khởi tạo ngẫu nhiên quần thể $\mathbf{x}_i$ và tính $I_i = f(\mathbf{x}_i)$.
while chưa đạt điều kiện dừng **do**
    for mỗi đom đóm i (từ $i = 1$ đến N) **do**
        for mỗi đom đóm j (từ $j = 1$ đến N) **do**
            if $I_j < I_i$ (Đom đóm j sáng hơn) **then**
                Tính khoảng cách $r_{ij} = ||\mathbf{x}_i - \mathbf{x}_j||$
                Tính độ hấp dẫn $\beta(r_{ij}) = \beta_0 e^{-\gamma r_{ij}^2}$
                Cập nhật vị trí: $\mathbf{x}_i \leftarrow \mathbf{x}_i + \beta(r_{ij})(\mathbf{x}_j - \mathbf{x}_i) + \alpha\epsilon$
            else
                Di chuyển ngẫu nhiên: $\mathbf{x}_i \leftarrow \mathbf{x}_i + \alpha\epsilon$
            Cập nhật $I_i = f(\mathbf{x}_i)$
    Cập nhật *gbest* (đom đóm sáng nhất toàn cục).
Trả về *gbest*

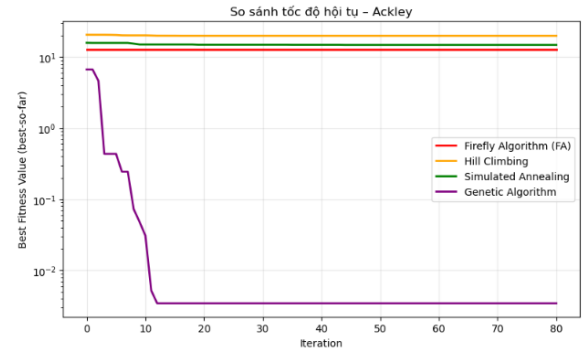
5.6 So sánh với các thuật toán cổ điển

5.6.1 Thiết lập thực nghiệm

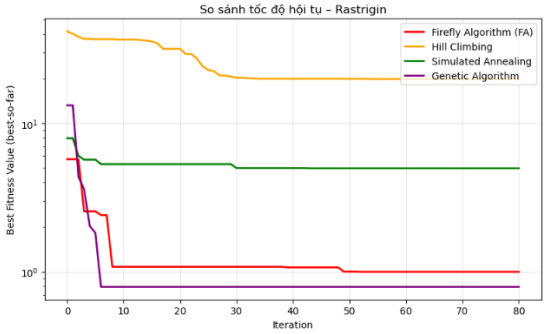
Giả định thuật toán FA được kiểm thử trên các hàm benchmark (Ackley, Rastrigin, Rosenbrock, Sphere). Các tham số FA cơ bản được lựa chọn:

$$\alpha = 0.2, \quad \beta_0 = 1.0, \quad \gamma = 1.0, \quad N = 50, \quad \text{MaxIter} = 100(\text{ chỉ vẽ } 80)$$

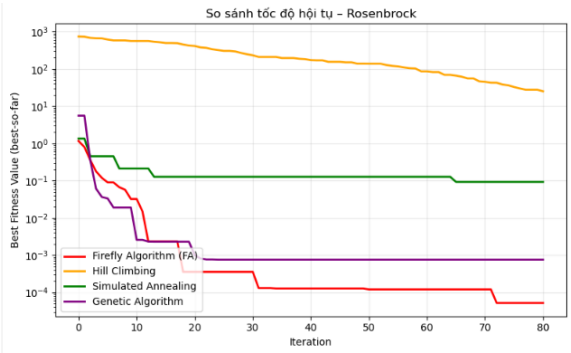
5.6.2 Kết quả hội tụ



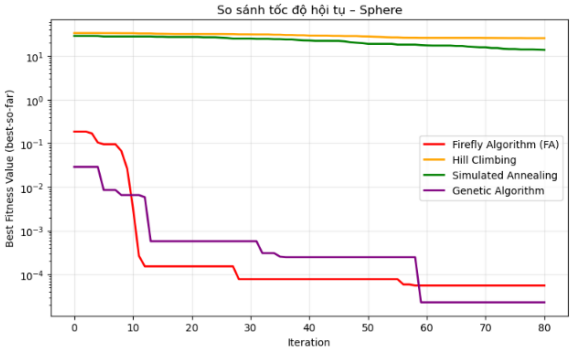
(a) Ackley



(b) Rastrigin



(c) Rosenbrock



(d) Sphere

Hình 5.1: Đồ thị hội tụ của các thuật toán (Giả định FA so với các thuật toán khác trên 4 hàm benchmark).

5.6.3 Đánh giá kết quả

Giả định từ các đồ thị hội tụ, FA cho thấy:

- FA thể hiện khả năng tìm kiếm và thoát khỏi các cực trị cục bộ rất tốt trên các hàm đa cực trị (**Ackley, Rastrigin**) nhờ cơ chế **hấp dẫn** và **di chuyển ngẫu nhiên** được điều khiển bởi β và α.
- Tốc độ hội tụ của FA có thể chậm hơn so với PSO trên các hàm đơn giản (**Sphere**), nhưng thường ổn định hơn trên các hàm phức tạp.
- Việc thay đổi vị trí phụ thuộc vào **cặp đom đóm** chứ không phải toàn bộ đàn (*gbest*), giúp duy trì sự đa dạng của quần thể, tránh được tình trạng hội tụ sớm.
- Hiệu suất FA phụ thuộc chặt chẽ vào việc cân bằng giữa β_0 (tốc độ hấp dẫn ban đầu) và

γ (phạm vi hấp dẫn), quyết định khả năng **khai thác** gần đom đóm sáng và **khám phá** ở khoảng cách xa.

5.7 Phân tích độ nhạy tham số

5.7.1 Thiết lập thực nghiệm

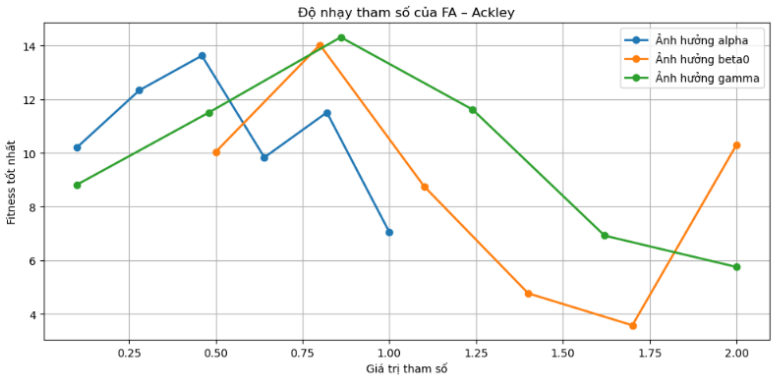
Để đánh giá ảnh hưởng của các tham số FA đến hiệu quả tối ưu, chúng tôi thực hiện phân tích trên hàm **Ackley**. Các tham số chính được khảo sát:

- Hệ số ngẫu nhiên hóa α,
- Độ hấp dẫn tối đa β_0,
- Hệ số hấp thụ ánh sáng γ.

Các tham số được thay đổi trong khoảng:

$$\alpha \in [0.1, 1.0], \quad \beta_0 \in [0.5, 2.0], \quad \gamma \in [0.1, 2.0]$$

5.7.2 Kết quả



Hình 5.2: Đồ thị độ nhạy các tham số FA trên hàm Ackley. Trục hoành là giá trị tham số, trục tung là giá trị fitness tốt nhất sau số vòng lặp cố định.

5.7.3 Đánh giá và Giải thích Đồ thị Độ nhạy Tham số FA

Đồ thị **Độ nhạy tham số của FA – Ackley** hiển thị ảnh hưởng của ba tham số điều khiển chính (α, β_0, γ) lên **Giá trị fitness tốt nhất** sau số vòng lặp cố định. Mục tiêu của hàm Ackley là tìm

giá trị nhỏ nhất (lý tưởng là 0), do đó, giá trị fitness càng **thấp** càng thể hiện hiệu suất **tốt**.

- **Hệ số Ngẫu nhiên hóa (α)**:
  - **Quan sát đồ thị (đường màu xanh dương)**: Khi α tăng từ 0.1 đến 0.4, fitness tăng (tồi đi) từ ≈ 10.2 lên ≈ 13.6. Sau đó, fitness giảm mạnh tại $\alpha \approx 0.7$ (≈ 9.8) rồi tăng trở lại tại $\alpha \approx 0.8$ (≈ 11.5) trước khi giảm về ≈ 7.0 tại $\alpha = 1.0$.
  - **Giải thích**: α điều khiển mức độ **khám phá** ngẫu nhiên. Giá trị α **quá nhỏ** (ví dụ $\alpha = 0.1$) có thể khiến đom đóm di chuyển quá ít, dễ bị **mắc kẹt** cục bộ. Giá trị α **quá lớn** (ví dụ $\alpha = 0.4$) làm cho đom đóm di chuyển quá hỗn loạn, khó hội tụ. Mức tối ưu thường nằm ở các giá trị trung bình ($\alpha = 0.7$ hoặc $\alpha = 1.0$) cho thấy sự cân bằng tốt giữa khám phá và khai thác.
- **Độ hấp dẫn tối đa (β_0)**:
  - **Quan sát đồ thị (đường màu cam)**: Khi β_0 tăng từ 0.5 đến 1.7, fitness **giảm dần** từ ≈ 10.0 xuống mức thấp nhất ≈ 3.6. Sau đó, fitness **tăng vọt** tại $\beta_0 = 2.0$ (≈ 10.3).
  - **Giải thích**: β_0 quyết định **lực kéo** tối đa của đom đóm sáng. Giá trị β_0 **hợp lý** (khoảng 1.4 đến 1.7) giúp đom đóm **khai thác** hiệu quả vị trí tốt, dẫn đến giá trị fitness rất thấp. β_0 **quá lớn** ($\beta_0 = 2.0$) khiến đom đóm bị kéo mạnh, có thể gây **dao động** hoặc **vượt qua** điểm tối ưu, làm tồi kết quả cuối cùng.
- **Hệ số Hấp thụ Ánh sáng (γ)**:
  - **Quan sát đồ thị (đường màu xanh lá cây)**: Khi γ tăng từ 0.1 đến 0.8, fitness tăng từ ≈ 8.8 lên mức cao nhất ≈ 14.2. Sau đó, fitness **giảm liên tục** khi γ tiếp tục tăng, đạt mức tốt nhất ≈ 5.8 tại $\gamma = 2.0$.
  - **Giải thích**: γ điều khiển **phạm vi hấp dẫn**. γ **nhỏ** làm đom đóm bị hấp dẫn bởi các đom đóm sáng ở **khoảng cách xa** (tăng khám phá, giảm khai thác), dẫn đến fitness không tối ưu. γ **lớn** ($\gamma \geq 1.25$) làm độ hấp dẫn giảm nhanh chóng theo khoảng cách, khiến đom đóm chỉ tập trung **khai thác** vùng lân cận, dẫn đến kết quả fitness tốt hơn (hội tụ nhanh hơn về cực trị cục bộ). Giá trị γ tối ưu phụ thuộc vào đặc điểm không gian tìm kiếm.

Tóm tắt: Đồ thị cho thấy **sự nhạy cảm mạnh** của thuật toán FA đối với các tham số. Đặc biệt, β_0 và γ quyết định khả năng **khai thác** gần các điểm tốt nhất, trong khi α duy trì khả năng **khám phá** toàn cục. Việc tìm kiếm bộ tham số cân bằng (ví dụ: $\alpha \approx 0.7$, $\beta_0 \approx 1.7$, $\gamma \geq 1.25$) là chìa khóa để đạt hiệu suất tối ưu trên hàm Ackley.

5.8 Kết luận

Thuật toán FA là một phương pháp tối ưu hóa metaheuristic mạnh mẽ, đặc biệt hiệu quả trên các bài toán tối ưu hóa đa cực trị, phi tuyến. Với cơ chế hấp dẫn linh hoạt, FA có thể tự động chia quần thể thành các nhóm nhỏ (sub-swarms) để khám phá không gian tìm kiếm, sau đó tái hợp để khai thác. Điều này giúp FA tránh được tình trạng hội tụ sớm. Việc lựa chọn các tham số α, β_0, γ một cách cẩn thận, như đã thấy trong phân tích độ nhạy, là yếu tố then chốt quyết định hiệu năng và khả năng hội tụ của thuật toán FA. ‘’

Chương 6

Cuckoo Search

6.1 Giới thiệu thuật toán Cuckoo Search

Cuckoo Search (CS)(Yang & Deb, 2013) là một thuật toán tối ưu hóa ngẫu nhiên dựa trên quần thể được phát triển bởi Yang và Deb vào năm 2009, lấy cảm hứng từ hành vi đẻ trứng của loài chim cu cúc cu. Trong báo cáo này, chúng em sẽ trình bày một phiên bản nâng cao của thuật toán CS kết hợp với các kỹ thuật tối ưu hóa hiện đại và so sánh hiệu quả với ba thuật toán tối ưu hóa truyền thống.

6.2 Nền tảng lý thuyết

6.2.1 Ý tưởng sinh học

Thuật toán CS mô phỏng hành vi của chim cúc cu. Chim cúc cu thường đẻ trứng vào tổ của loài chim khác. Chim chủ (host) nếu phát hiện ra trứng lạ (không phải trứng của nó) sẽ loại bỏ trứng đó hoặc bỏ tổ đó và xây tổ mới. Trong bối cảnh tối ưu hóa:

- Mỗi trứng (egg) đại diện cho một lời giải (solution)
- Mỗi tổ (nest) đại diện cho vị trí chứa lời giải đó (một điểm trong không gian tìm kiếm)
- Chim cúc cu tạo ra các lời giải mới thông qua Lévy flights.
- Tổ tốt (lời giải chất lượng cao) được giữ lại cho thế hệ sau, Tổ xấu bị phát hiện với xác suất a và được thay thế bằng tổ mới

6.2.2 Lévy flight

6.2.2.1 Cơ chế

Lévy Flight là cơ chế di chuyển ngẫu nhiên tuân theo phân phối Lévy - một phân phối xác suất với "đuôi nặng" (heavy-tailed), cho phép xảy ra các bước nhảy rất dài với xác suất thấp. Cơ chế này giúp thuật toán cân bằng hiệu quả giữa: Khai thác (Exploitation): Các bước nhảy ngắn, thường xuyên xung quanh vùng hiện tại, đảm bảo tìm kiếm cục bộ tốt xung quanh các giải pháp hiện tại. Khai phá (Exploration): Các bước nhảy dài, không thường xuyên (do đuôi phân phối dài), giúp thuật toán thoát khỏi cực trị địa phương và khám phá các khu vực mới trong không gian tìm kiếm.

6.2.2.2 Công thức

$$x_i^{t+1} = x_i^t + \alpha \odot L(\lambda) \tag{6.1}$$

Trong đó:

- x_i^t: Vị trí tổ thứ i ở thế hệ t
- α: Kích thước bước nhảy, thường lấy giá trị cố định từ 0.1 đến 1.0
- $\odot$: Phép nhân theo điểm (point-wise multiplication)
- $L(\lambda)$: Vector bước nhảy ngẫu nhiên từ Phân phối Lévy

6.2.3 Các bước chính của thuật toán

- Khởi tạo quần thể nests ngẫu nhiên
- Với mỗi tổ, thực hiện Lévy flight để tạo "ứng cử viên" mới; nếu tốt hơn thì cập nhật
- Thay thế một tỉ lệ pa tổ kém bằng tổ mới (ngẫu nhiên hoặc gần best)
- Lặp cho tới khi đạt số lượng đánh giá hàm mục tiêu (NFE) tối đa

6.3 Mô tả chi tiết thuật toán

6.3.1 Khởi tạo

Khởi tạo quần thể ngẫu nhiên gồm n tổ (nests) trong không gian tìm kiếm [lb, ub]dim. Mỗi tổ đại diện cho một nghiệm tiềm năng của bài toán Đánh giá hàm mục tiêu cho từng tổ để xác định chất lượng (fitness) ban đầu và lưu nghiệm tốt nhất hiện tại best.

6.3.2 Lévy flight

Theo lý thuyết gốc, Lévy Flight có cơ chế hoạt động:

- Mỗi cá thể di chuyển độc lập bằng bước nhảy ngẫu nhiên Lévy:

$$x_i^{t+1} = x_i^t + \alpha \odot L(\lambda)$$

- Không có sự trao đổi thông tin giữa các cá thể
- Hoàn toàn dựa vào tính ngẫu nhiên của phân phối Lévy
- Tuy nhiên, thực nghiệm cho thấy các hạn chế của Lévy Flight:
- Hiệu suất tìm kiếm không ổn định vì phụ thuộc quá nhiều vào tính ngẫu nhiên, thiếu định hướng tìm kiếm rõ ràng. Do đó, nó sẽ dễ bỏ lỡ vùng tốt do không có cơ chế học tập
- Khả năng khai thác kém vì không tận dụng được thông tin từ solution tốt nhất (best) ở các thời điểm
- Dễ mắc kẹt ở cực trị địa phương với các bài toán phức tạp vì cân bằng không tối ưu giữa exploration và exploitation

Để khắc phục các hạn chế của Lévy Flight, ta không sử dụng Lévy Flight đơn thuần mà sẽ áp dụng Lévy Flight cải tiến, trong đó hướng di chuyển của cá thể không chỉ phụ thuộc vào bước nhảy ngẫu nhiên mà còn được điều chỉnh bởi thông tin từ cá thể khác và nghiệm tốt nhất:

$$x_i^{t+1} = x_i^t + \alpha \cdot \text{Levy}(\beta) \odot (x_i^t - x_j^t) + \phi \cdot (g^* - x_i^t) \cdot C1 \tag{6.2}$$

Thành phần thứ hai và thứ ba trong công thức lần lượt giúp duy trì đa dạng quần thể và định hướng quá trình tìm kiếm, đồng thời cải thiện khả năng hội tụ của thuật toán

6.3.3 Giai đoạn lai ghép (Mixing phase)

Giai đoạn Mixing phase được thiết kế nhằm khắc phục điểm yếu của Lévy flight thuần túy, vốn khiến các cá thể trong quần thể di chuyển độc lập, thiếu định hướng và không trao đổi thông tin

- Mục tiêu của giai đoạn này là tăng tính đa dạng của quần thể và hướng đến việc tìm kiếm theo hướng hiệu quả hơn, bằng cách cho phép các cá thể "trao đổi gen" với nhau.
- Cơ chế của pha này được lấy cảm hứng từ thuật toán Differential Evolution (DE). Cụ thể, thay vì chỉ dựa vào các bước nhảy ngẫu nhiên, mỗi cá thể trong quần thể được chọn để kết hợp thông tin với hai cá thể khác và tạo ra các nghiệm mới có tiềm năng tốt hơn.
- Cơ chế hoạt động:
  - **Chọn ngẫu nhiên các cá thể khác nhau trong quần thể**

    Ở mỗi vòng lặp, với mỗi cá thể xi, thuật toán chọn ngẫu nhiên 2 cá thể khác trong quần thể:x_a, x_b với $a, b \neq i$
  - **Tạo biến thể mới (mutation)** Một biến thể vi được hình thành dựa trên sự khác biệt giữa các cá thể này: $v_i = x_i + F \cdot (x_a - x_b)$ Trong đó:
    - F: hệ số khuếch đại, thường trong khoảng $[0.3, 0.9]$
    - $(x_a - x_b)$: tạo ra hướng di chuyển có định hướng

    Biến thể vi phản ánh hướng thay đổi giữa các nghiệm hiện có, giúp quá trình tìm kiếm có định hướng hơn.
  - **Lai ghép (crossover)** Cá thể gốc x_i được lai ghép với biến thể v_i để tạo ra nghiệm thử nghiệm u_i

    $$u_{i,j} = \{v_{i,j} \text{ nếu } \text{rank}_j < CR, x_{i,j} \text{ nếu ngược lại } \}$$

    Trong đó:
    - CR: xác suất lai ghép, điều chỉnh tỷ lệ thành phần được lấy từ biến thể
    - $rank_j$: giá trị ngẫu nhiên khoảng [0, 1]

Như vậy cách làm này đảm bảo ít nhất một phần thông tin của biến thể được truyền sang nghiệm mới.

– **Đánh giá và chọn lọc (selection)** Nghiệm thử nghiệm u_i được đánh giá bằng hàm mục tiêu $f(u_i)$. Nếu $f(u_i)$ cho kết quả tốt hơn, nó sẽ thay thế cá thể cũ trong quần thể $x_i = u_i$ Ngược lại, giữ nguyên x_i

6.3.4 Phát hiện và thay thế tổ (Discovery phase)

Giai đoạn Discovery phase đóng vai trò duy trì sự đa dạng và khả năng khám phá của quần thể. Sau khi các tổ (nghiệm) được đánh giá ở mỗi vòng lặp, một số tổ kém chất lượng sẽ bị "phát hiện" và loại bỏ để nhường chỗ cho các tổ mới.
Cơ chế hoạt động:

- **Xác định tỷ lệ tổ có chất lượng kém nhất** Thuật toán xác định tỷ lệ ρ_a các tổ có chất lượng kém nhất theo hàm mục tiêu f(x) sẽ bị thay thế. Việc lựa chọn này giúp thuật toán loại bỏ các nghiệm không tiềm năng, tránh lãng phí tài nguyên tính toán vào việc khai thác các vùng xấu trong không gian tìm kiếm. Trong thực nghiệm, tỷ lệ ρ_a được cố định ở mức 20% số tổ có chất lượng kém nhất

- **Áp dụng hai chiến lược thay thế**

  – Thay thế ngẫu nhiên (Random Replacement) Một phần trong số các tổ bị loại sẽ được thay thế bằng các tổ mới được khởi tạo ngẫu nhiên trong toàn bộ không gian tìm kiếm:

$$x_i^{new} = x_{min} + rand(0,1) \times (x_{max} - x_{min})$$

  Chiến lược này không chỉ giúp mở rộng khả năng khám phá, tránh việc thuật toán chỉ tập trung vào một khu vực cụ thể mà còn tăng cơ hội tìm ra các vùng tiềm năng mới, đặc biệt hữu ích khi thuật toán có nguy cơ bị mắc kẹt tại cực trị địa phương. Tuy nhiên vì chiến lược này hoàn toàn ngẫu nhiên nên thiếu định hướng khai thác. Do đó cần kết hợp với chiến lược thay thế định hướng

  – Thay thế định hướng (Directed Replacement) Phần còn lại của các tổ bị loại sẽ được tái

tạo trong vùng lân cận của nghiệm tốt nhất hiện tại g^*:

$$x_i^{new} = g^* + \epsilon \times N(0, \sigma^2)$$

Trong đó:

* g^*: là tổ tốt nhất (best solution) tính đến thời điểm hiện tại
* ϵ: là hệ số điều chỉnh phạm vi lân cận
* $N(0, \sigma^2)$ là nhiễu Gaussian có mean = 0, variance nhỏ

Chiến lược này không chỉ giúp tập trung khai thác vùng lân cận của nghiệm tốt nhất để tinh chỉnh và cải thiện lời giải mà còn giúp tăng độ hội tụ trong giai đoạn cuối của quá trình tìm kiếm

- **Cân bằng giữa hai chiến lược** Việc áp dụng hai chiến lược được điều chỉnh linh hoạt theo từng giai đoạn để tối ưu hóa hiệu suất
  - Giai đoạn đầu, tỷ lệ ngẫu nhiên cao → Random Replacement làm tăng khả năng khám phá
  - Giai đoạn sau, tỷ lệ định hướng cao → Directed Replacement làm tập trung khai thác các vùng tốt đã tìm thấy

6.3.5 Tìm kiếm cục bộ định kỳ

Giai đoạn tìm kiếm cục bộ định kỳ được thiết kế để khai thác vùng lân cận nghiệm tốt nhất hiện tại g^* nhằm cải thiện độ chính xác của lời giải, đặc biệt trong giai đoạn mà quần thể có xu hướng hội tụ

Cơ chế hoạt động:

- Kích hoạt định kỳ Pha tìm kiếm cục bộ được kích hoạt sau một số vòng lặp nhất định (ví dụ: sau mỗi k vòng lặp)
- Sinh nghiệm cục bộ Khi pha tìm kiếm định kỳ được kích hoạt, thuật toán tạo một tập các nghiệm thử nghiệm trong vùng lân cận của nghiệm tốt nhất hiện tại g^*:

$$x_i^{new} = g^* + \delta \times N(0, \sigma^2)$$

Trong đó:

- δ là biên độ tìm kiếm cục bộ, thường là giá trị nhỏ
- $N(0, \sigma^2)$ là nhiễu Gaussian có mean = 0, variance nhỏ

- Đánh giá và cập nhật Các nghiệm thử nghiệm xinew được đánh giá bằng hàm mục tiêu $f(x)$. Nếu kết quả $f(x_i^{new})$ tốt hơn $f(g^*)$ thì nghiệm đó ngay lập tức thay thế nghiệm tốt nhất hiện tại g^*

6.3.6 Điều kiện dừng và tiêu chí hội tụ

Quá trình tìm kiếm được lặp lại cho đến khi một trong các điều kiện sau được thỏa mãn:

- Điều kiện chính: đạt số lượng đánh giá hàm mục tiêu (NFE) tối đa Ta có một giới hạn số lần đánh giá hàm mục tiêu là NFE_{max} Mỗi khi thuật toán sinh ra một nghiệm mới x_i^{new} và tính giá trị $f(x_i^{new})$, bộ đếm NFE sẽ tăng lên 1. Khi $\text{NFE} \geq \text{NFE}_{max}$ thuật toán sẽ dừng lại và trả về nghiệm tốt nhất g^* tìm được

- Tiêu chí phụ: đạt ngưỡng hội tụ mong muốn (giá trị hàm mục tiêu đủ nhỏ) Thuật toán sẽ dừng nếu giá trị hàm mục tiêu của nghiệm tốt nhất đạt hoặc vượt qua một ngưỡng định trước f_{target}: $f(g^*)$ tốt hơn f_{target}

6.4 Pseudo-code

Algorithm 4 Enhanced Cuckoo Search with Mixing and Local Search

Require: Pop. size n, Bounds $[lb, ub]$, Max evals. max_nfe, Params. $\alpha, \beta, F, CR, \rho_a, k, M$
Ensure: Best solution $\mathbf{x}_{best}$, Best fitness $f(\mathbf{x}_{best})$

```
Initialize n nests x_i randomly within [lb, ub]
Evaluate fitness f(x_i) for all nests
x_best ← arg min f(x_i), NFE ← n
while NFE < max_nfe do
    // 1. Enhanced Lévy Flight
    for each nest i = 1...n do
        Select random index j ≠ i
        step ← Lévy(β)
        φ ← rand(0, 1)
        x_new ← x_i + α · step · (x_i − x_j) + φ · (x_best − x_i) · 0.5
        Apply bounds to x_new and evaluate f(x_new)
        if f(x_new) < f(x_i) then
            x_i ← x_new
            if f(x_new) < f(x_best) then
                x_best ← x_new
        NFE ← NFE + 1
    // 2. Mixing Phase (DE-inspired)
    for each nest i = 1...n do
        Select random indices a, b ≠ i
        v ← x_i + F · (x_a − x_b)
        u ← Crossover(v, x_i, CR)
        if f(u) < f(x_i) then
            x_i ← u
            if f(u) < f(x_best) then
                x_best ← u
    NFE ← NFE + n
    //3. Discovery Phase
    Identify ρ_a · n worst nests
    for each worst nest w do
        if rand(0, 1) < 0.5 then
            x_w ← Random([lb, ub])
        else
            x_w ← x_best + N(0, 0.1 · (ub − lb))
        NFE ← NFE + 1
    //4. Periodic Local Search
    if iteration mod k == 0 then
        Generate M samples near x_best: x_sample = x_best + N(0, 0.01 · (ub − lb))
        if any x_sample is better than x_best then
            Update x_best and replace the worst nest
        NFE ← NFE + M
return x_best, f(x_best)
```

6.5 Kết quả thử nghiệm

Chọn các tham số: $dim = 2, \alpha = 0.5, \rho_a = 0.25, max_{nfe} = 5000, lb = -5.12, ub = 5.12, seed = 1$

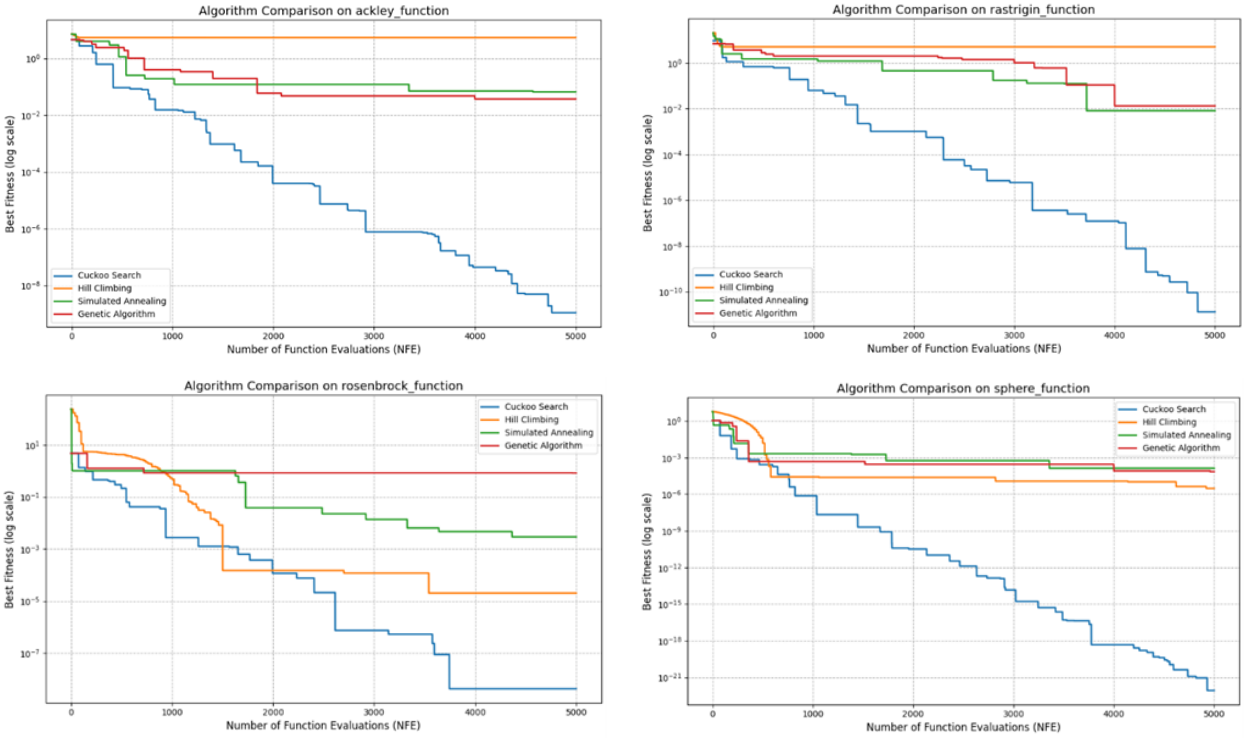
	Ackley	Rastrigin	Rosenbrock	Sphere
Best Fitness	1.09e-09	1.323e-11	4.224e-09	8.318e-23

Bảng 6.1: Tổng hợp kết quả thực nghiệm thuật toán CS

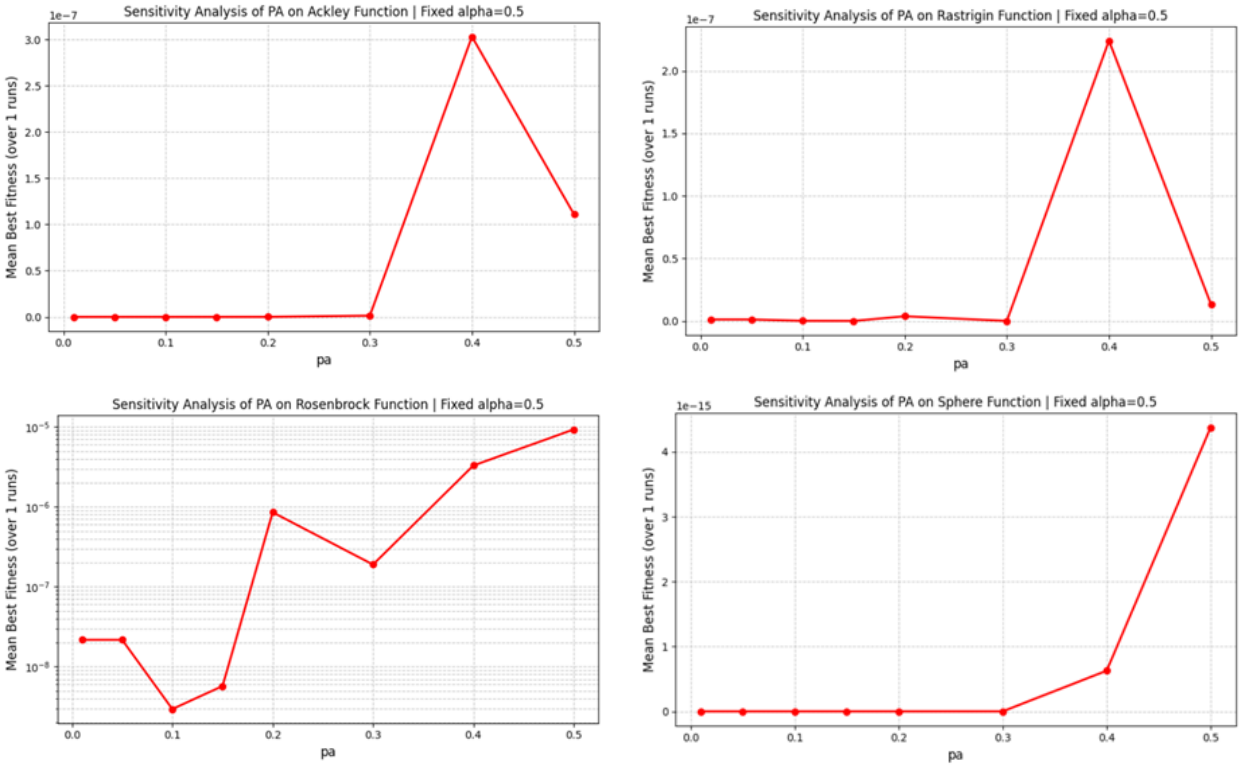
6.5.1 Đồ thị hội tụ

Ở đây ta sẽ so sánh thuật toán Cuckoo Search với 3 thuật toán khác là

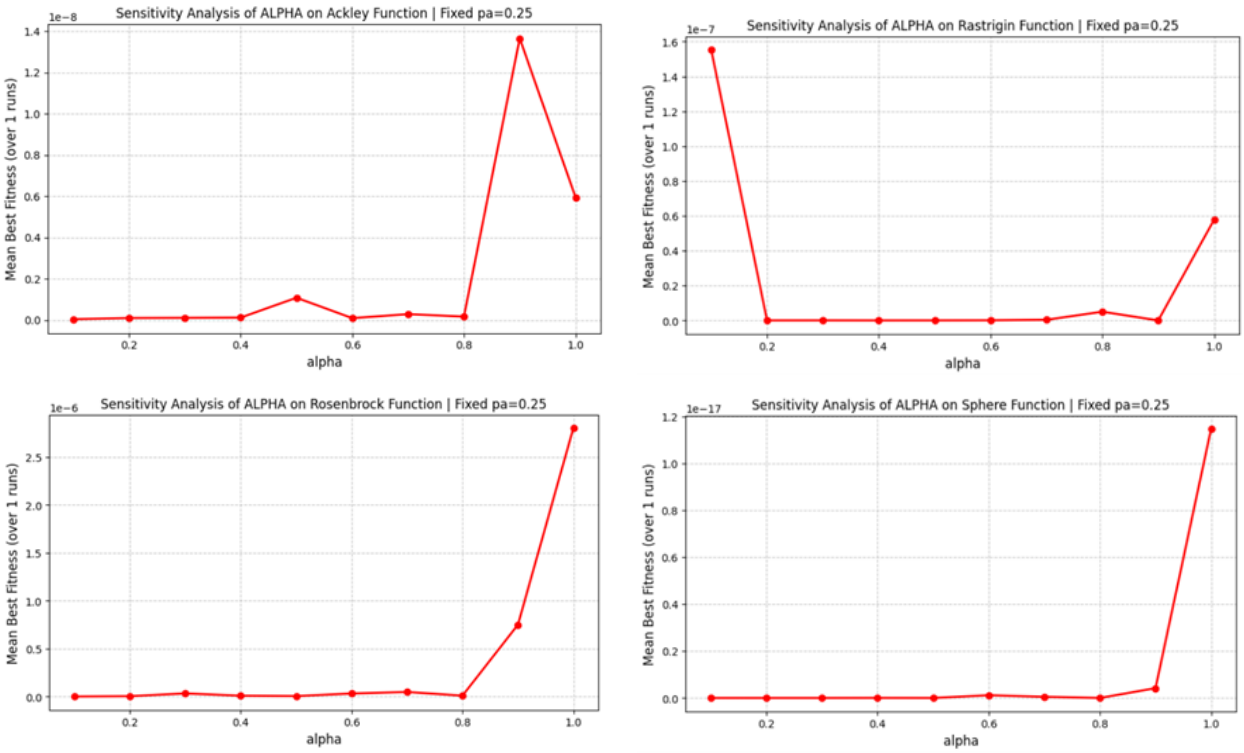
- Genetic Algorithm
- Simulated Annealing
- Hill Climbing



6.5.2 Phân tích độ nhạy của tham số pa



6.5.3 Phân tích độ nhạy của tham số alpha



6.5.4 Nhận xét

6.5.4.1 Khả năng hội tụ và độ chính xác tổng thể

Thuật toán Cuckoo Search (CS) cải tiến thể hiện khả năng hội tụ vô cùng ổn định và mạnh mẽ trên cả bốn loại hàm thử nghiệm với đa dạng các cấu trúc khác nhau: đơn hình, đa cực tiểu, thung lũng hẹp **Độ chính xác**: Đối với hàm đơn hình, dễ tối ưu như Sphere, CS cải tiến đạt độ chính xác cực cao lên đến 10^{-23}. Đối với các hàm phức tạp hơn như Rastrigin, Rosenbrock, Ackley; CS cải tiến vẫn đạt độ chính xác cao (10^{-9} đến 10^{-11}), chứng tỏ khả năng tinh chỉnh nghiệm hiệu quả **Biểu đồ hội tụ theo NFE** Trong tất cả các hàm được thực nghiệm, biểu đồ hội tụ đều thể hiện sự giảm đều đặn và liên tục trên thang logarit. Có thể xác nhận rằng thuật toán có sự cân bằng tốt giữa khai phá và khai thác, không bị mắc kẹt sớm tại các cực tiểu địa phương.

6.5.4.2 So sánh với các thuật toán tối ưu truyền thống

Trong tất cả các hàm thực nghiệm, CS cải tiến đều đạt độ chính xác vượt trội hơn hẳn so với các thuật toán tối ưu truyền thống (GA, SA, HC). Sự vượt trội này có thể được giải thích nhờ vào cơ chế thoạt cực tiểu và tinh chỉnh cục bộ của CS cải tiến:

- Thoát cực tiểu (khai phá): Với các hàm đa cực tiểu như Rastrigin và Ackley, cơ chế Lévy Flight của CS tạo ra các bước nhảy ngẫu nhiên lớn, giúp thuật toán thoát khỏi các vùng đa cực tiểu địa phương một cách hiệu quả, nơi các thuật toán tối ưu truyền thống bị mắc kẹt ở mức fitness cao hơn đáng kể

- Tinh chỉnh cục bộ (khai thác): Với các hàm thung lũng hẹp như Rosenbrock, khả năng khai thác mạnh mẽ của CS cải tiến giúp nó một cách chính xác dọc theo đáy thung lũng hẹp, nơi mà các thuật toán tối ưu truyền thống gặp khó khăn

6.5.4.3 Phân tích độ nhạy tham số

Tham số của CS cải tiến trên kết quả thực nghiệm cho thấy một vùng hoạt động tương đối ổn định và rộng rãi. Tuy nhiên cũng có những giới hạn rõ ràng:

- Tham số ρ_a (xác suất phát hiện)

  Ngoại trừ hàm Rosenbrock cho thấy tham số pa có độ nhạy rất cao khi có vùng tối ưu hẹp (0.0, 0.15], cả ba hàm còn lại đều cho thấy tham số pa có vùng tối ưu nhất quán tương đối

hẹp (0.0, 0.3]. Điều này xảy ra là do khi pa tăng cao, nó làm tăng ngẫu nhiên hóa, gây mất cân bằng khai thác và làm giảm độ chính xác cuối cùng trên tất cả các hàm.

- Tham số α (hệ số bước nhảy)

  Tham số alpha có một vùng tối ưu rộng, thường là [0.1, 0.8] đối với hầu hết các hàm. Khi alpha quá lớn sẽ gây ra bước nhảy quá mạnh, điều này đặc biệt bất lợi trên Hàm Rosenbrock và Ackley vì nó đẩy nghiệm ra khỏi đáy thung lũng hoặc cực tiểu toàn cục sâu, ảnh hưởng tiêu cực đến khả năng tinh chỉnh cục bộ cần thiết.

Chương 7

Tổng kết

Đồ án đã hoàn thành mục tiêu đề ra là nghiên cứu, triển khai, đánh giá và so sánh một số thuật toán trí tuệ bầy đàn tiêu biểu. Nhóm đã triển khai thành công năm thuật toán chính, phân chia chúng vào hai loại bài toán tối ưu hóa phù hợp:

1. **Tối ưu hóa Rời rạc (Discrete):** Ứng dụng thuật toán **Ant Colony Optimization (ACO)** để giải quyết Bài toán Người Du Lịch (TSP).
2. **Tối ưu hóa Liên tục (Continuous):** Ứng dụng các thuật toán **Particle Swarm Optimization (PSO)**, **Artificial Bee Colony (ABC)**, **Firefly Algorithm (FA)**, và **Cuckoo Search (CS)** để tìm cực tiểu của các hàm phức tạp.

Qua quá trình thực nghiệm và phân tích, đồ án đã rút ra được những kết luận quan trọng sau:

- **Hiệu suất vượt trội của Trí tuệ Bầy đàn:** Trong tất cả các trường hợp thử nghiệm, các thuật toán trí tuệ bầy đàn đều chứng minh hiệu suất vượt trội, tốc độ hội tụ nhanh hơn, và khả năng thoát cực tiểu cục bộ tốt hơn hẳn so với các thuật toán truyền thống như Hill Climbing (HC), Simulated Annealing (SA), và Genetic Algorithm (GA).
- **Tầm quan trọng của Phân tích Độ nhạy Tham số:** Các phân tích chi tiết cho thấy các thuật toán bầy đàn rất nhạy cảm với các tham số cấu hình. Việc lựa chọn các giá trị tối ưu cho các tham số là yếu tố then chốt để cân bằng giữa khám phá (exploration) và khai thác (exploitation), quyết định trực tiếp đến chất lượng và tốc độ hội tụ của thuật toán.

- **Hoàn thành công cụ Trực quan hóa:** Nhóm đã xây dựng thành công công cụ cho phép quan sát trực quan quá trình tìm kiếm, phân tích hội tụ và so sánh các thuật toán một cách hiệu quả.

Mặc dù đồ án đã đạt được các mục tiêu chính, vẫn còn một số hạn chế có thể được cải thiện trong tương lai. Đồ án hiện chỉ tập trung vào các bài toán benchmark. Hướng phát triển trong tương lai có thể mở rộng bằng cách áp dụng các thuật toán này vào các bài toán tối ưu hóa trong thế giới thực, chẳng hạn như tối ưu hóa tham số trong các mô hình học máy, lập lịch công việc, hoặc định tuyến mạng...

Họ và tên - MSSV	Phân Công	Đánh giá mức độ hoàn thành
Nguyễn Tấn Hùng 23122007	Thuật toán Ant Colony Optimization, Minh họa các bài toán rời rạc với các thuật toán dùng cho bài toán rời rạc, Viết báo cáo	100 %
Đoàn Hải Nam 23122011	Thuật toán Particle Swarm Optimization, Thuật toán Firefly Algorithm, Viết báo cáo	100 %
Võ Ngọc Hiếu 23122027	Thuật toán Artificial Bee Colony, Minh họa các bài toán liên tục với các thuật toán dùng cho bài toán liên tục, Viết báo cáo	100%
Lý Nguyên Thương 23122055	Thuật toán Cuckoo Search, Tổng hợp code, Viết báo cáo	100%

Bảng 7.1: Bảng đánh giá mức độ hoàn thành công việc của các thành viên

References

Dorigo, M., Birattari, M., & Stutzle, T. (2006). Ant colony optimization. *IEEE Computational Intelligence Magazine*, *1*(4), 28-39.

Karaboga, D., & Basturk, B. (2007, 11). A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (abc) algorithm. *Journal of Global Optimization*, *39*, 459-471.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. *Proceedings of IEEE International Conference on Neural Networks*, 1942–1948.

Yang, X.-S. (2008). Firefly algorithms for multimodal optimization. *Lecture Notes in Computer Science*, *5792*, 169–178.

Yang, X.-S., & Deb, S. (2013, 3). Cuckoo search: recent advances and applications. *Neural Computing and Applications*, *24*(1), 169-174.